Solo Forth

Version 0.14.0-rc.124+20201123 for G+DOS

Marcos Cruz
(programandala.net)

Solo Forth

Version 0.14.0-rc.124+20201123 for G+DOS

Marcos Cruz

2020-11-21

Table of Contents

Description

Solo Forth is a Forth system for the ZX Spectrum 128 and compatible computers, with disk drives and +3DOS, G+DOS, or TR-DOS.

Solo Forth cannot run on the original ZX Spectrum 48, but could be used to develop programs for it.

Solo Forth can be used as a stand-alone Forth system (either on an emulator or on the real computer), or as part of a cross-development environment on a GNU/Linux operating system (in theory, other type of operating systems could be used as well).

Main features

- Fast DTC (Direct Threaded Code) implementation.
- A kernel as small as possible.
- Name space in banked memory, separated from code and data space.
- Easy access to banked memory.
- Big library of useful source code.
- Modular DOS support.
- Fully documented source code.
- Detailed documentation.
- Conform to the Forth standard (not fully tested yet).

Minimum requirements

- 128 KiB RAM.
- One double-sided 80-track disk drive (two recommended), for 800-KiB disks.

Motivation, history and current status

The motivation behind Solo Forth is double:

1. **I wanted to program the ZX Spectrum with a modern Forth system**: In 2015, my detailed disassembly of ZX Spectrum's Abersoft Forth, a popular tape-based implementation of fig-Forth ported to several 8-bit home computers in the 1980's (and the Forth system I started learning Forth with in 1984), helped me understand the inner working of the fig-Forth model, including its by-design limitations, compared to modern Forths, and discover some bugs of the ZX Spectrum port. At the same time I wrote the Afera library in order to make Abersoft Forth more stable, powerful and comfortable for cross development. The objective was reached but, after a certain point, further improvements weren't feasible without making radical changes in the system. The need for a new Forth system arised: a Forth designed from the start to use disk drives and banked memory, and useful for cross-development.
2. **Nobody had written such a Forth system before:** In 2015 there was no disk-based Forth for the ZX Spectrum platform, and the only Forth written for ZX Spectrum 128 (the first model with banked memory) was Lennart Benschop's Forth-83 (1988). But despite being more powerful than fig-Forth, it is still tape-based and keeps the block sources in a RAM disk. Beside, the system is built by metacompilation, what makes it difficult to adapt to disk drives.

The development of Solo Forth started on 2015-05-30, from the disassembled code of Abersoft Forth. Some ideas and code were reused also from the Afera library and from a previous abandoned project called DZX-Forth (a port of CP/M DX-Forth to ZX Spectrum +3e).

The GitHub repository was created on 2016-03-13 from the development backups, in order to preserve the evolution of the code from the very beginning.

Solo Forth is very stable, and it's being used to develop two projects in Forth: Nuclear Waste Invaders and Black Flag.

About this manual

This is the G+DOS variant of the Solo Forth manual. Nevertheless, some details about the rest of supported DOSes are included as well, when the comparison is useful.

This manual is built automatically from the sources. It contains mainly a description of the Forth system, the basic usage information and a complete glossary with cross references.

Platforms

Table 1. Supported platforms

Computer	Disk interface	DOS
Pentagon 128		TR-DOS
Pentagon 512		TR-DOS
Pentagon 1024		TR-DOS
Scorpion ZS 256		TR-DOS
ZX Spectrum 128	Beta 128	TR-DOS
ZX Spectrum 128	Plus D	G+DOS
ZX Spectrum +2	Beta 128	TR-DOS
ZX Spectrum +2	Plus D	G+DOS
ZX Spectrum +2A	(External disk drive)	+3DOS
ZX Spectrum +2B	(External disk drive)	+3DOS
ZX Spectrum +3		+3DOS
ZX Spectrum +3e		+3DOS

Porting the G+DOS version of Solo Forth to its close relatives GDOS, Beta DOS and Uni-DOS would require only minor changes, beside adding some library code to support their specific features.

Supporting IDEDOS, ResiDOS, esxDOS or NextZXOS would let Solo Forth use hard drives, flash cards, and a lot of memory.

Table 2. Platforms that could be supported in future versions

Computer	Disk interface	DOS
ZX Evolution TS-Conf		TR-DOS
ZX Spectrum +3e	divIDE/divMMC/ZXATASP/ZXCF/ZXMMC…	IDEDOS
ZX Spectrum +3e	divIDE/divMMC/ZXATASP/ZXCF/ZXMMC…	ResiDOS
ZX Spectrum 128/+2	DISCiPLE	GDOS
ZX Spectrum 128/+2	DISCiPLE/Plus D	Uni-DOS
ZX Spectrum 128/+2	Plus D	Beta DOS
ZX Spectrum 128/+2/+2A/+2B/+3	divIDE/divMMC/ZXATSP/ZXCF/ZXMMC…	esxDOS
ZX Spectrum Next		NextZXOS
ZX-UNO/ZX-UNO+/ZX-DOS		esxDOS

Comparative of DOS support

Block disks are fully supported on G+DOS, but some file-management words are not implemented yet, especially the standard Forth words.

The following table shows the main disk-management words implemented on each DOS:

Table 3. Main disk-related words implemented

Word	**+3DOS**	**G+DOS**	**TR-DOS**	**Description**
`2-block-drives`	YES	YES	YES	Use the first two drives as block drives
`3-block-drives`	n/a	n/a	YES	Use the first three drives as block drives
`4-block-drives`	n/a	n/a	YES	Use the first four drives as block drives
`>file`		YES	YES	Save memory zone to a file
`acat`	YES	YES	YES	Display an abbreviated disk catalogue
`bank-read-file`	YES			Read from a file with a bank paged in
`bank-write-file`	YES			Write to a file with a bank paged in
`bin`	YES	YES		Standard Forth
`cat`	YES	YES	YES	Display a detailed disk catalogue
`close-file`	YES			Standard Forth
`create-file`	YES			Standard Forth
`delete-file`	YES	YES	YES	Standard Forth
`drive-unused`	YES	YES		Return the number of unused kibibytes in a drive
`file-dir#`			YES	Return the directory number of a file
`file-exists?`		YES	YES	Return a flag: does a files exists?
`file-length`		YES	YES	Return the length of a file
`file-position`	YES			Standard Forth
`file-sector`			YES	Return the first sector of a file
`file-sectors`			YES	Return the number of sectors occupied by a file
`file-size`	YES			Standard Forth
`file-start`		YES	YES	Return the start address of a file
`file-status`		YES	YES	Standard Forth
`file-track`			YES	Return the first track of a file
`file-type`		YES	YES	Return the type of a file
`file>`		YES	YES	Load file contents to memory zone
`find-file`		YES	YES	Find a file
`flush-drive`	YES			Write all pending data for a drive
`flush-file`				Standard Forth
`get-block-drives`	YES	YES	YES	Get the drives used as block drives
`get-drive`	YES	YES	YES	Get the number of the current drive

Word	+3DOS	G+DOS	TR-DOS	Description
`include-file`				Standard Forth
`include`				Standard Forth
`included`				Standard Forth
`open-file`	YES			Standard Forth
`r/o`	YES			Standard Forth
`r/w`	YES			Standard Forth
`read-byte`	YES			Read byte from file
`read-file`	YES			Standard Forth
`read-line`	YES			Standard Forth
`rename-file`	YES	YES	YES	Standard Forth
`reposition-file`	YES			Standard Forth
`require`				Standard Forth
`required`				Standard Forth
`resize-file`				Standard Forth
`set-block-drives`	YES	YES	YES	Set the drives used as block drives
`set-drive`	YES	YES	YES	Set the current drive
`undelete-file`			YES	Undelete a file
`w/o`	YES			Standard Forth
`wacat`	YES	YES		Display an abbreviated disk catalogue with wildcards
`wcat`	YES	YES		Display a detailed disk catalogue with wildcards
`write-byte`	YES			Write a byte to a file
`write-file`	YES			Standard Forth
`write-line`	YES			Standard Forth

Download

Solo Forth can be downloaded from two sites:

- Solo Forth home page
- Solo Forth repository in GitHub

Project directories

Directory	Subdirectory	Description
backgrounds		Version background images
bin		ZX Spectrum binary files for disk 0
bin	addons	Code loaded from disk, not assembled in the library yet
bin	dos	DOS files
bin	fonts	Fonts for the supported screen modes
disks		Disk images
disks	gplusdos	G+DOS disk images
disks	plus3dos	+3DOS disk images
disks	trdos	TR-DOS disk images
doc		Manuals in DocBook, EPUB, HTML and PDF
make		Files used by `make` to build the system
screenshots		Version screenshots
src		Sources
src	addons	Code to be loaded from disk. Not used yet.
src	doc	Files used to build the documentation
src	inc	Z80 symbols
src	lib	Library
src	loader	BASIC loader for disk 0
tmp		Temporary files created by `make`
tools		Development and user tools
vim		Vim files
vim	ftplugin	Filetype plugin
vim	syntax	Syntax highlighting

Disks

The <disks/gplusdos> directory of the directory tree contains the following disk images:

```
disks/gplusdos/disk_0_boot.mgt
disks/gplusdos/disk_1_library.mgt
disks/gplusdos/disk_2_programs.mgt
disks/gplusdos/disk_3_workbench.mgt
```

- Disk 0 is the boot disk. It contains the BASIC loader, the Solo Forth binary, some addons (i.e. compiled code that is not part of the library yet) and fonts for the supported screen modes.
- Disk 1 contains the library.
- Disk 2 contains some programs: little sample games, most of them under development, two block editors and one sound editor.
- Disk 3 contains tests and benchmarks. Most of them were used during the development and their only documentation is the commented source.

WARNING Disks 1, 2 and 3 are Forth block disks: They contain the source Forth blocks directly on the disk sectors, without any file system. Therefore their contents cannot be accessed with ordinary DOS commands.

The MGT disk image format

The MGT disk images (used for G+DOS and other systems) do not include format-describing metadata: The MGT file is just a dump of the original 800-KiB disk. Beside, G+DOS does not need its own metadata (the directory tracks) be present in order to read or write sectors, making the whole disk usable for Forth blocks.

In fact the Solo Forth's MGT disk images that contain Forth blocks are identical to Forth block files. Therefore they can be browsed or edited using a Forth block editor.

How to run

1. Run a ZX Spectrum emulator and select a ZX Spectrum 128 (or ZX Spectrum +2) with the Plus D disk interface.
2. “Insert” the disk image file <disks/gplusdos/disk_0_boot.mgt> as disk 1 of the Plus D disk interface.
3. Choose "128 BASIC" from the computer start menu.
4. Type `run` in BASIC. G+DOS will be loaded from disk, and Solo Forth as well.

How to use the library

1. Run Solo Forth.
2. “Insert” the file <disks/gplusdos/disk_1_library.mgt> as disk 2 of the Plus D disk interface. Type `2 set-drive throw` to make drive 2 the current one.
3. Type `1 load` to load block 1 from the library disk. By convention, block 0 cannot be loaded (it is used for comments), and block 1 is used as a loader. In Solo Forth, block 1 contains `2 load`, in order to load the `need` tool from block 2.
4. Type `need name`, were “name” is the name of the word or tool you want to load from the library.

How to make a library index

The `need` word and its related words search the index line (line 0) of all blocks of the disk for the first occurence of the required word, within a configurable range of blocks (using the variables `first-locatable` and `last-locatable`). Of course, nested `need` are resolved the same way: searching the library from the beginning. This can be slow. This is not a problem, because the goal of Solo Forth is cross development, and therefore only the last step of the development loop, i.e., the compilation of the sources from the disk images created in the host system, compilation that includes all the slow searching of library blocks, is done in the real (actually, emulated) machine. But the system includes a tool to create an index of the library, which is used to locate their contents instantaneously, what makes things more comfortable when the Forth system is used interactively.

How to use the library index:

1. Load the indexer with `need make-thru-index`.
2. Make the index and activate it with `make-thru-index`.
3. The default behaviour (no index) can be restored with `use-no-index`. The index can be reactivated with `use-thru-index`.

The indexer creates an index (actually, a Forth word list whose definitions use no code or data space) and changes the default behaviour of `need` and related words to use it. Then `need name` will automatically start loading the first block where the “name” is defined

Table 4. Time and name-space memory required to make the library index

Computer	DOS	Block drives	Library blocks	Seconds	Bytes
ZX Spectrum 128	G+DOS	1	791 (8..799)	357	19515
ZX Spectrum +3	+3DOS	1	710 (8..718)	323	18636
Pentagon 128	TR-DOS	2	1263 (8..1271)	255	18437
Scorpion ZS 256	TR-DOS	2	1263 (8..1271)	291	18437

NOTE The name space is in far memory, a virtual 64-KiB space formed by 4 configurable memory banks (see `far-banks`). No code or data space is used by the indexer.

An alternative indexer is under development. It’s activated with `use-fly-index` and does not make and index in advance: Instead, it indexes the blocks on the fly, when they are searched the first time. This indexer was included in Solo Forth 0.12.0 but it’s not finished yet.

How to load a program that needs the library

The programs included in disk 2, and the tests and benchmarks included in disk 3, need words from the library, which is in disk 1. Therefore, two disk drives must be configured first as block drives, using `2-block-drives`.

Let’s see an example, how to load the game called Tetris for Terminals, which is in disk 2.

1. Run Solo Forth.
2. Insert the library disk image (disk 1) into the first drive.
3. Insert the programs disk image (disk 2) into the second drive.
4. Execute command `1 load` in order to `load` the `need` utility from the library disk.
5. Execute the command `need 2-block-drives`, which loads `2-block-drives` from the library disk and then executes it, setting the first and the second drives as block drives.
6. Execute the command `need tt`, which locates the first block of the game (in disk 2) and loads it, loading its requirements from the library (disk 1) as needed.
7. Follow the instructions.

When `2-block-drives` is executed, the blocks of the first two disk drives are seen as one single set, i.e. `200 list` will list block 200 from the first disk, but `850 list` will list the block from the second disk:

Table 5. Range of blocks per drive on every DOS, in normal order

DOS	1st drive	2nd drive	3rd drive	4th drive
+3DOS	0-718	719-1437	n/a	n/a
G+DOS	0-799	800-1599	n/a	n/a
TR-DOS	0-635	636-1271	1272-1908	1909-2544

`2-block-drives` is a layer above `set-block-drives`, which can configure any number of block drives in any order. Examples:

```
2 1 2 set-block-drives \ identical to ``2-block-drives``
1 2 2 set-block-drives \ use both drives in reverse order
```

How to search the source files

A shell script is included in order to make searching the Forth sources for a regular expression a bit easier: <tools/seek>.

The script uses `ack` by default; if `ack` is not installed, `grep` is used instead. All parameters are passed to them.

Usage examples:

```
./tools/seek use-thru-index
./tools/seek use-thru-index -l
./tools/seek color
./tools/seek ";\s:\s"
./tools/seek "\-bank"
./tools/seek "code\s+\S+\s+\("
```

How to test and benchmark

Disk 3 (called "workbench") contains many little specific tests and benchmarks used during the development of Solo Forth, probably not interesting for the application programmer. But it also contains an adapted version of the Hayes test and some known benchmarks.

First, set the required block disks

1. Run Solo Forth or enter `cold` to start from scratch.
2. "Insert" the file <disks/gplusdos/disk_1_library.mgt> into disk drive 1 of your emulated machine.
3. "Insert" the file <disks/gplusdos/disk_3_workbench.mgt> into disk drive 2 of your emulated machine.
4. Enter command `1 load` to `load` the `need` tool.
5. Enter command `need 2-block-drives` to set both disk drives as block drives in their normal order, making `need` search both of them: first drive 1 (the library), then drive 2 (the benchmarks and tests). Note `need 2-block-drives` not only loads the word `2-block-drives`, but also executes it. This is equivalent to the command `need set-block-drives 2 1 2 set-block-drives`

Second, load the desired code

Depending on the code you want to run, enter the corresponding command:

1. `need hayes-test`
2. `need byte-magazine-benchmark`
3. `need interface-age-benchmark`
4. `need vector-loop-benchmark`
5. `need all-benchmarks` to run all the three benchmarks above

How to write Forth programs

Briefly, the steps of cross development are the following:

1. Edit the sources of the Forth program on the host operating system, using the simple FSB format described in the documentation of fsb and fsb2.
2. Convert the sources into Forth block disk images using fsb2.
3. Run Solo Forth on a ZX Spectrum emulator and compile the Forth program from the disk image. Further testing and debugging can be done in the Forth system.

In order to use Solo Forth to write programs for ZX Spectrum, programmers already acquainted with Forth and GNU/Linux systems can extract all the required information from the <Makefile> of Solo Forth.

The only difference between building Solo Forth and building a Forth program is the content of disk 0 (the boot disk), if needed, and the library modules included in the library disk, which usually also contains the source of the program at the end. If the program doesn't need to use the disk at run-time, you can simply copy the default disk 0, and boot it to `load` your program from block 1 of your customized disk 1, with a simple `1 load`. When the loading finishes, you can save a system snapshot, in SZX format, using the corresponding option of your ZX Spectrum emulator.

Some games are provided as examples, in disk 2. In order to try, improve and fix the Forth system during its development, two more complex game projects are being developed at the same time:

- Black Flag (Black Flag in GitHub).
- Nuclear Waste Invaders (Nuclear Waste Invaders in GitHub).

They are not finished yet, but they can be useful as examples of program development with Solo Forth. See how the useful `load-program` is used in block 1 of their sources.

How to rebuild Solo Forth

If you modify the sources, you have to build new disk images for your DOS of choice. Also the manual depends on the documentation included in the sources.

First, see the requirements listed in the header of the <Makefile> file and install the required programs. Then enter the project directory and use one of the following commands to build the disk images or the manual for your DOS of choice:

Table 6. Commands to rebuild the disk images

DOS	Computer	
+3DOS	All	`make plus3dos`
G+DOS	All	`make gplusdos`
TR-DOS	All	`make trdos`
TR-DOS	128-KiB models	`make trdos128`
TR-DOS	Pentagon 512/1024	`make pentagon`
TR-DOS	Scorpion ZS 256	`make scorpion`
All	All	`make`

Table 7. Commands to rebuild the manual

Format	+3DOS	G+DOS	TR-DOS	All
DocBook	`make plus3dosdbk`	`make gplusdosdbk`	`make trdosdbk`	`make dbk`
EPUB	`make plus3dosepub`	`make gplusdosepub`	`make trdosepub`	`make epub`
HTML	`make plus3doshtml`	`make gplusdoshtml`	`make trdoshtml`	`make html`
ODT	`make plus3dosodt`	`make gplusdosodt`	`make trdosodt`	`make odt`
PDF	`make plus3dospdf`	`make gplusdospdf`	`make trdospdf`	`make pdf`
All	`make plus3dosdoc`	`make gplusdosdoc`	`make trdosdoc`	`make doc`

NOTE

Only the EPUB, HTML and PDF built directly from the Asciidoctor source are included in the release files. Other formats like ODT and DocBook, or the EPUB and HTML variants obtained from DocBook, are optional and can be built from the sources.

The disk images will be created in the <disks> directory. The manual will be created in the <doc> directory.

Exception codes

Exception codes (also called `throw` codes of `throw` values) are used as prescribed by the Forth-2012 standard: codes -255..-1 are used only as assigned by the standard, and codes -4095..-256 are reserved for Solo Forth. Therefore, programs shall not define values for use with `throw` in the range -4095..-1.

Table 8. Exception code ranges

Range	Reserved for
1..32767	Programs
-255..-1	Standard Forth
-999..-256	Solo Forth
-1127..-1000	Solo Forth: G+DOS
-1154..-1128	Solo Forth: ZX Spectrum OS (BASIC)
-4095..-1155	Solo Forth
-32768..-4096	Programs

The original ZX Spectrum OS error codes are included (in range -1154..-1128) because a few of them may be returned by some DOS words in special cases.

The way errors are displayed is configurable. By default only the exception code is displayed by an uncatched `throw`, because the default action of `.throw`, which is a deferred word, is `.throw#`. In order to display also the error description, the alternative action `.throw-message` must be loaded from the library with `need .throw-message`.

Table 9. Exception code assignments

Exception code	Meaning
#-01	ABORT
#-02	ABORT"
#-03	stack overflow
#-04	stack underflow
#-05	return stack overflow
#-06	return stack underflow
#-07	do-loops nested too deeply during execution
#-08	dictionary overflow
#-09	invalid memory address
#-10	division by zero
#-11	result out of range

Exception code	Meaning
#-12	argument type mismatch
#-13	undefined word
#-14	interpreting a compile-only word
#-15	invalid FORGET
#-16	attempt to use zero-length string as a name
#-17	pictured numeric output string overflow
#-18	parsed string overflow
#-19	definition name too long
#-20	write to a read-only location
#-21	unsupported operation
#-22	control structure mismatch
#-23	address alignment exception
#-24	invalid numeric argument
#-25	return stack imbalance
#-26	loop parameters unavailable
#-27	invalid recursion
#-28	user interrupt
#-29	compiler nesting
#-30	obsolescent feature
#-31	>BODY used on non-CREATEd definition
#-32	invalid name argument
#-33	block read exception
#-34	block write exception
#-35	invalid block number
#-36	invalid file position
#-37	file I/O exception
#-38	non-existent file
#-39	unexpected end of file
#-40	invalid BASE for floating point conversion
#-41	loss of precision
#-42	floating-point divide by zero
#-43	floating-point result out of range
#-44	floating-point stack overflow

Exception code	Meaning
#-45	floating-point stack underflow
#-46	floating-point invalid argument
#-47	compilation word list deleted
#-48	invalid POSTPONE
#-49	search-order overflow
#-50	search-order underflow
#-51	compilation word list changed
#-52	control-flow stack overflow
#-53	exception stack overflow
#-54	floating-point underflow
#-55	floating-point unidentified fault
#-56	QUIT
#-57	exception in sending or receiving a character
#-58	[IF], [ELSE], or [THEN] exception
#-59	ALLOCATE
#-60	FREE
#-61	RESIZE
#-62	CLOSE-FILE
#-63	CREATE-FILE
#-64	DELETE-FILE
#-65	FILE-POSITION
#-66	FILE-SIZE
#-67	FILE-STATUS
#-68	FLUSH-FILE
#-69	OPEN-FILE
#-70	READ-FILE
#-71	READ-LINE
#-72	RENAME-FILE
#-73	REPOSITION-FILE
#-74	RESIZE-FILE
#-75	WRITE-FILE
#-76	WRITE-LINE
#-77	malformed xchar

Exception code	Meaning
#-78	SUBSTITUTE
#-79	REPLACES
#-256	not a word nor a number
#-257	warning: is not unique
#-258	stack imbalance
#-259	trying to load from block 0
#-260	wrong digit
#-261	deferred word is uninitialized
#-262	assertion failed
#-263	execution only
#-264	definition not finished
#-265	loading only
#-266	off current editing block
#-267	warning: not present, though needed
#-268	needed, but not located
#-269	relative jump too long
#-270	text not found
#-271	immediate word not allowed in this structure
#-272	array index out of range
#-273	invalid assembler condition
#-274	command line history overflow
#-275	wrong number
#-276	dictionary reached the zone of memory banks
#-277	needed, but not indexed
#-278	empty block found: quit indexing
#-279	user area overflow
#-280	user area underflow
#-281	escaped strings search-order overflow
#-282	escaped strings search-order underflow
#-283	assembly label number out of range
#-284	assembly label number already used
#-285	too many unresolved assembly label references
#-286	not located

Exception code	Meaning
#-287	wrong number of drives
#-288	too many files open
#-289	input source exhausted
#-290	invalid UDG scan
#-291	out of OS memory
#-292	file access method not supported
#-293	string too long
#-1000	G+DOS: Nonsense in G+DOS
#-1001	G+DOS: Nonsense in GNOS
#-1002	G+DOS: Statement end error
#-1003	G+DOS: Break requested
#-1004	G+DOS: Sector error
#-1005	G+DOS: Format data lost
#-1006	G+DOS: Check disk in drive
#-1007	G+DOS: No +SYS file
#-1008	G+DOS: Invalid file name
#-1009	G+DOS: Invalid station
#-1010	G+DOS: Invalid device
#-1011	G+DOS: Variable not found
#-1012	G+DOS: Verify failed
#-1013	G+DOS: Wrong file type
#-1014	G+DOS: Merge error
#-1015	G+DOS: Code error
#-1016	G+DOS: Pupil set
#-1017	G+DOS: Invalid code
#-1018	G+DOS: Reading a write file
#-1019	G+DOS: Writing a read file
#-1020	G+DOS: O.K. G+DOS
#-1021	G+DOS: Network off
#-1022	G+DOS: Wrong drive
#-1023	G+DOS: Disk write protected
#-1024	G+DOS: Not enough space on disk
#-1025	G+DOS: Directory full

Exception code	**Meaning**
#-1026	G+DOS: File not found
#-1027	G+DOS: End of file
#-1028	G+DOS: File name used
#-1029	G+DOS: No G+DOS loaded
#-1030	G+DOS: STREAM used
#-1031	G+DOS: CHANNEL used
#-1128	OS: OK
#-1129	OS: NEXT without FOR
#-1130	OS: Variable not found
#-1131	OS: Subscript wrong
#-1132	OS: Out of memory
#-1133	OS: Out of screen
#-1134	OS: Number too big
#-1135	OS: RETURN without GO SUB
#-1136	OS: End of file
#-1137	OS: STOP statement
#-1138	OS: Invalid argument
#-1139	OS: Integer out of range
#-1140	OS: Nonsense in BASIC
#-1141	OS: BREAK - CONT repeats
#-1142	OS: Out of DATA
#-1143	OS: Invalid file name
#-1144	OS: No room for line
#-1145	OS: STOP in INPUT
#-1146	OS: FOR without NEXT
#-1147	OS: Invalid I/O device
#-1148	OS: Invalid colour
#-1149	OS: BREAK into program
#-1150	OS: RAMTOP no good
#-1151	OS: Statement lost
#-1151	OS: Invalid stream
#-1152	OS: FN without DEF
#-1153	OS: Parameter error

Exception code	Meaning
#-1154	OS: Tape loading error

Notation

Stack notation

Stack parameters input to and output from a definition are described using the notation:

```
( stack-id: before -- after )
```

where *stack-id:* specifies which stack is being described, *before* represents the stack-parameter data types before execution of the definition and *after* represents them after execution. The symbols used in *before* and *after* are shown in table Stack notation symbols for numbers.

The control-flow-stack stack-id is "C:", the data-stack stack-id is "S:", and the return-stack stack-id is "R:". When there is no confusion, the data-stack stack-id is omitted. In Solo Forth, the data stack is used as control-flow stack.

When there are several items of the same type, a numerical suffix is added: *(n1 n2 — n3)*; sometimes with brackets: *(n[1] n[2] — n[3])*; sometimes with hashes: *(n#1 n#2 — n#3)*. Eventually, the format will be unified.

When there are alternate *after* representations, they are described by *after[1] | after[2]*, e.g. *(n — ca len true | false)*.

When there are alternate items, they are described by *item[1]|item[2]*, e.g. *(n[1]|u[1] n[2]|u[2] — n[3]|u[3])* .

The top of the stack is to the right. Only those stack items required for or provided by execution of the definition are shown.

Table 10. Stack notation symbols for numbers

Symbol	Data type	Size	Range
a	address	1 cell	0 .. 65535
aa	aligned address[1]	1 cell	0 .. 65535
ca	character-aligned address[1]	1 cell	0 .. 65535
fa	float-aligned address[1]	1 cell	0 .. 65535
f	well-formed flag (false: 0; true: -1)	1 cell	-1 .. 0
0f	zero flag (false: 0; true: non-zero)	1 cell	-32768 .. 65535
true	true flag (-1)	1 cell	-1
false	false flag (0)	1 cell	0
b	8-bit byte	1 cell	-128 .. 255
c	8-bit character	1 cell	0 .. 255
char	8-bit character	1 cell	0 .. 255

Symbol	Data type	Size	Range
u	16-bit unsigned number	1 cell	0 .. 65535
len	16-bit unsigned number (memory zone length)	1 cell	0 .. 65335
n	16-bit signed number	1 cell	-32768 .. 32767
+n	16-bit non-negative number	1 cell	0 .. 32767
x	16-bit unspecified number	1 cell	-32768 .. 65535
d	32-bit signed double number	2 cells	-2147483648 .. 2147483647
+d	32-bit non-negative double number	2 cells	0 .. 2147483647
ud	32-bit unsigned double number	2 cells	0 .. 4294697295
xd	32-bit unspecified number	2 cells	-2147483648 .. 2147483647
t	48-bit signed triple number	3 cells	-140737488355328 .. 140737488355327
+t	48-bit non-negative triple number	3 cells	0 .. 140737488355327
ut	48-bit unsigned triple number	3 cells	0 .. 281474976710655
q	64-bit signed quadruple number	4 cells	−9223372036854775808 .. 9223372036854775807
+q	64-bit non-negative quadruple number	4 cells	0 .. 9223372036854775807
uq	64-bit unsigned quadruple number	4 cells	0 .. 18446744073709551615
col	8-bit cursor column	1 cell	0 .. 31; 0 .. 41; 0 .. 63
row	8-bit cursor row	1 cell	0 .. 23
gx	8-bit (absolute) or 16-bit (relative) graphic x coordinate	1 cell	0 .. 255; -32768 .. 32767
gy	8-bit (absolute) or 16-bit (relative) graphic y coordinate	1 cell	0 .. 191; -32768 .. 32767; 0 .. 175
xt	execution token (=cfa)	1 cell	
cfa	code field address (=xt)	1 cell	
lfa	link field address	1 cell	
nt	name token (=nfa)	1 cell	
nfa	name field address (=nt)	1 cell	
dfa	data field address	1 cell	
xtp	execution token pointer	1 cell	
wid	word-list identifier	1 cell	

Symbol	Data type	Size	Range
ior	Input/Output result code	1 cell	-32768 .. 0
dosior	Input/Output result code in DOS format	1 cell	-322768 .. 65535
orig	address of an unresolved forward branch	1 cell	0 .. 65535
dest	address of a backward branch target	1 cell	0 .. 65535
cs-id	control structure identifier	1 cell	
do-sys	loop control parameters (=orig)	1 cell	0 .. 65535
loop-sys	loop control parameters	2 cells	
nest-sys	definition call	1 cell	
source-sys	source identifier	n cells	
i*x	any data type	0 or more cells	
j*x	any data type	0 or more cells	
k*x	any data type	0 or more cells	
u*x	u elements of type x (eg. `u*wid`)	0 or more cells	
r	a floating point real number	5 bytes[2]	1E-38 .. 1E38
rf	a floating point real number flag	5 bytes[2]	0 .. 1
op	Z80 8-bit opcode, generally a jump	1 cell	0 .. 255
reg	Z80 8-bit register identifier	1 cell	0 .. 7
regp	Z80 16-bit register pair identifier	1 cell	0; 2; 4
regph	Z80 16-bit HL register pair identifier	1 cell	4
regpi	Z80 16-bit IX/IY register pair identifier	1 cell	4

Table 11. Stack notation symbols for parsed text

Symbol	Description
<char>	the delimiting character marking the end of the string being parsed
<chars>	zero or more consecutive occurrences of the character char
<space>	a delimiting space character
<spaces>	zero or more consecutive occurrences of the character space
<quote>	a delimiting double quote
<paren>	a delimiting right parenthesis
<eol>	an implied delimiter marking the end of a line
ccc	a parsed sequence of arbitrary characters, excluding the delimiter character
name	a token delimited by space, equivalent to `<spaces>ccc<space>` or `<spaces>ccc<eol>`

Z80 flags notation

Table 12. Z80 flags notation

Symbol	Description	bit
Fc	Carry flag	0
Fh	Half Carry flag	4
Fn	Add/Subtract flag	1
Fp	Parity/Overflow flag	2
Fs	Sign flag	7
Fz	Zero flag	6

[1] As Solo Forth runs on the Z80 processor, all addresses are aligned, but the specific symbols for aligned addresses are used in the source, for clarity.

[2] In the floating point stack of the ZX Spectrum operating system.

Z80 instructions

Table 13. Z80 instructions and their equivalent Forth commands

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
ADC A,(HL)	m a adc,	1	8E	7	***V0*	A=A+[HL]+CY
ADC A,(IX+n)	n ix adcx,	3	DD 8E xx	19	***V0*	A=A+[IX+n]+CY
ADC A,(IY+n)	n iy adcx,	3	FD 8E xx	19	***V0*	A=A+[IY+n]+CY
ADC A,n	n adc#,	2	CE xx	7	***V0*	A=A+n+CY
ADC A,r	r adc,	1	88+r	4	***V0*	A=A+r+CY
ADC HL,BC	b adcp,	2	ED 4A	15	***V0*	HL=HL+BC+CY
ADC HL,DE	d adcp,	2	ED 5A	15	***V0*	HL=HL+DE+CY
ADC HL,HL	h adcp,	2	ED 6A	15	***V0*	HL=HL+HL+CY
ADC HL,SP	sp adcp,	2	ED 7A	15	***V0*	HL=HL+SP+CY
ADD A,(HL)	m a add,	1	86	7	***V0*	A=A+[HL]
ADD A,(IX+n)	n ix addx,	3	DD 86 xx	19	***V0*	A=A+[IX+n]
ADD A,(IY+n)	n iy addx,	3	FD 86 xx	19	***V0*	A=A+[IY+n]
ADD A,n	n add#,	2	C6 xx	7	***V0*	A=A+n
ADD A,r	r add,	1	80+r	4	***V0*	A=A+r
ADD HL,BC	b addp,	1	09	11	--*-0*	HL=HL+BC
ADD HL,DE	d addp,	1	19	11	--*-0*	HL=HL+DE
ADD HL,HL	h addp,	1	29	11	--*-0*	HL=HL+HL
ADD HL,SP	sp addp,	1	39	11	--*-0*	HL=HL+SP
ADD IX,BC	b addix,	2	DD 09	15	--*-0*	IX=IX+BC
ADD IX,DE	d addix,	2	DD 19	15	--*-0*	IX=IX+DE
ADD IX,IX	n/a	2	DD 29	15	--*-0*	IX=IX+IX
ADD IX,SP	sp addix,	2	DD 39	15	--*-0*	IX=IX+SP
ADD IY,BC	b addiy,	2	FD 09	15	--*-0*	IY=IY+BC
ADD IY,DE	d addiy,	2	FD 19	15	--*-0*	IY=IY+DE
ADD IY,IY	n/a	2	FD 29	15	--*-0*	IY=IY+IY
ADD IY,SP	sp addiy,	2	FD 39	15	--*-0*	IY=IY+SP
AND (HL)	m and,	1	A6	7	***P00	A=A&[HL]
AND (IX+n)	n ix andx,	3	DD A6 xx	19	***P00	A=A&[IX+n]
AND (IY+n)	n iy andx,	3	FD A6 xx	19	***P00	A=A&[IY+n]
AND n	n and#,	2	E6 xx	7	***P00	A=A&n

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
AND r	r and,	1	A0+r	4	***P00	A=A&r
BIT b,(HL)	m b bit,	2	CB 46+8*b	12	**1*0-	[HL]&{2^b}
BIT b,(IX+n)	n ix b bitx,	4	DD CB xx 46+8*b	20	**1*0-	[IX+n]&{2^b}
BIT b,(IY+n)	n iy b bitx,	4	FD CB xx 46+8*b	20	**1*0-	[IY+n]&{2^b}
BIT b,r	r b bit,	2	CB 40+8*b+r	8	**1*0-	r&{2^b}
CALL C,nn	nn c? ?call,	3	DC xx xx	17/10	------	If CY then [SP-=2]=PC,PC=nn
CALL M,nn	nn m? ?call,	3	FC xx xx	17/10	------	If S then [SP-=2]=PC,PC=nn
CALL NC,nn	nn nc? ?call,	3	D4 xx xx	17/10	------	If !CY then [SP-=2]=PC,PC=nn
CALL nn	nn call,	3	CD xx xx	17	------	SP-=2,[SP+1,SP]=PC,PC=nn
CALL NZ,nn	nn nz? ?call,	3	C4 xx xx	17/10	------	If !Z then [SP-=2]=PC,PC=nn
CALL P,nn	nn p? ?call,	3	F4 xx xx	17/10	------	If !S then [SP-=2]=PC,PC=nn
CALL PE,nn	nn pe? ?call,	3	EC xx xx	17/10	------	If P then [SP-=2]=PC,PC=nn
CALL PO,nn	nn po? ?call,	3	E4 xx xx	17/10	------	If !P then [SP-=2]=PC,PC=nn
CALL Z,nn	nn z? ?call,	3	CC xx xx	17/10	------	If Z then [SP-=2]=PC,PC=nn
CCF	ccf,	1	3F	4	--*-0*	CY=~CY
CP (HL)	m cp,	1	BE	7	***V1*	A-[HL]
CP (IX+n)	n ix cpx,	3	DD BE xx	19	***V1*	A-[IX+n]
CP (IY+n)	n iy cpx,	3	FD BE xx	19	***V1*	A-[IY+n]
CP n	n cp#,	2	FE xx	7	***V1*	A-n
CP r	r cp,	1	B8+r	4	***V1*	A-r
CPD	n/a	2	ED A9	16	****1-	A-[HL],HL=HL-1,BC=BC-1
CPDR	n/a	2	ED B9	21/16	****1-	CPD until A=[HL] or BC=0
CPI	n/a	2	ED A1	16	****1-	A-[HL],HL=HL+1,BC=BC-1
CPIR	cpir,	2	ED B1	21/16	****1-	CPI until A=[HL] or BC=0
CPL	cpl,	1	2F	4	--1-1-	A=~A
DAA	daa,	1	27	4	***P-*	A=adjust result to BCD-format
DEC (HL)	m dec,	1	35	11	***V1-	[HL]=[HL]-1
DEC (IX+n)	n ix decx,	3	DD 35 xx	23	***V1-	[IX+n]=[IX+n]-1
DEC (IY+n)	n iy decx,	3	FD 35 xx	23	***V1-	[IY+n]=[IY+n]-1
DEC A	a dec,	1	3D	4	***V1-	A=A-1

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
DEC B	b dec,	1	05	4	***V1-	B=B-1
DEC BC	b decp,	1	0B	6	------	BC=BC-1
DEC C	c dec,	1	0D	4	***V1-	C=C-1
DEC D	d dec,	1	15	4	***V1-	D=D-1
DEC DE	d decp,	1	1B	6	------	DE=DE-1
DEC E	e dec,	1	1D	4	***V1-	E=E-1
DEC H	h dec,	1	25	4	***V1-	H=H-1
DEC HL	h decp,	1	2B	6	------	HL=HL-1
DEC IX	ix decp,	2	DD 2B	10	------	IX=IX-1
DEC IY	iy decp,	2	FD 2B	10	------	IY=IY-1
DEC L	l dec,	2	2D	4	***V1-	L=L-1
DEC SP	sp decp,	1	3B	6	------	SP=SP-1
DI	di,	1	F3	4	------	disable interrupts
DJNZ n	nn djnz,	2	10 xx	13/8	------	B=B-1, if B != 0 then PC+=n
EI	ei,	1	FB	4	------	enable interrupts
EX (SP),HL	exsp,	1	E3	19	------	[SP]<->HL
EX (SP),IX	n/a	2	DD E3	23	------	[SP]<->IX
EX (SP),IY	n/a	2	FD E3	23	------	[SP]<->IY
EX AF,AF'	exaf,	1	08	4	******	AF<->AF'
EX DE,HL	exde,	1	EB	4	------	DE<->HL
EXX	exx,	1	D9	4	------	BC<->BC',DE<->DE',HL<->HL'
HALT	halt,	1	76	4	------	repeat NOP until interrupt
IM 0	n/a	2	ED 46	8	------	set interrupt 0
IM 1	im1,	2	ED 56	8	------	set interrupt 1
IM 2	im2,	2	ED 5E	8	------	set interrupt 2
IN A,(C)	a inbc,	2	ED 78	12	***P0-	A=[C]
IN A,(n)	n in,	2	DB xx	11	------	A=[n]
IN B,(C)	b inbc,	2	ED 40	12	***P0-	B=[C]
IN C,(C)	c inbc,	2	ED 48	12	***P0-	C=[C]
IN D,(C)	d inbc,	2	ED 50	12	***P0-	D=[C]
IN E,(C)	e inbc,	2	ED 58	12	***P0-	E=[C]
IN H,(C)	h inbc,	2	ED 60	12	***P0-	H=[C]

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
`IN L,(C)`	`l inbc,`	2	`ED 68`	12	`***P0-`	`L=[C]`
`INC (HL)`	`h incp,`	1	`34`	11	`***V0-`	`[HL]=[HL]+1`
`INC (IX+n)`	`n ix incx,`	3	`DD 34 xx`	23	`***V0-`	`[IY+n]=[IX+n]+1`
`INC (IY+n)`	`n iy incx,`	3	`FD 34 xx`	23	`***V0-`	`[IY+n]=[IY+n]+1`
`INC A`	`a inc,`	1	`3C`	4	`***V0-`	`A=A+1`
`INC B`	`b inc,`	1	`04`	4	`***V0-`	`B=B+1`
`INC BC`	`b incp,`	1	`03`	6	`------`	`BC=BC+1`
`INC C`	`c inc,`	1	`0C`	4	`***V0-`	`C=C+1`
`INC D`	`d inc,`	1	`14`	4	`***V0-`	`D=D+1`
`INC DE`	`d incp,`	1	`13`	6	`------`	`DE=DE+1`
`INC E`	`e inc,`	1	`1C`	4	`***V0-`	`E=E+1`
`INC H`	`h inc,`	1	`24`	4	`***V0-`	`H=H+1`
`INC HL`	`h incp,`	1	`23`	6	`------`	`HL=HL+1`
`INC IX`	`ix incp,`	2	`DD 23`	10	`------`	`IX=IX+1`
`INC IY`	`iy incp,`	2	`FD 23`	10	`------`	`IY=IY+1`
`INC L`	`l inc,`	1	`2C`	4	`***V0-`	`L=L+1`
`INC SP`	`sp incp,`	1	`33`	6	`------`	`SP=SP+1`
`IND`	n/a	2	`ED AA`	16	`***?1-`	`[HL]=[C],HL=HL-1,B=B-1`
`INDR`	n/a	2	`ED BA`	21/16	`01*?1-`	`IND until B=0`
`INI`	n/a	2	`ED A2`	16	`***?1-`	`[HL]=[C],HL=HL+1,B=B-1`
`INIR`	n/a	2	`ED B2`	21/16	`01*?1-`	`INI until B=0`
`JP (HL)`	`jphl,`	1	`E9`	4	`------`	`PC=HL`
`JP (IX)`	`jpix,`	2	`DD E9`	8	`------`	`PC=IX`
`JP (IY)`	n/a	2	`FD E9`	8	`------`	`PC=IY`
`JP C,nn`	`nn c? ?jp,`	3	`DA xx xx`	10/10	`------`	`If CY then PC=nn`
`JP M,nn`	`nn m? ?jp,`	3	`FA xx xx`	10/10	`------`	`If S then PC=nn`
`JP NC,nn`	`nn nc? ?jp,`	3	`D2 xx xx`	10/10	`------`	`If !CY then PC=nn`
`JP nn`	`nn jp,`	3	`C3 xx xx`	10	`------`	`PC=nn`
`JP NZ,nn`	`nn nz? ?jp,`	3	`C2 xx xx`	10/10	`------`	`If !Z then PC=nn`
`JP P,nn`	`nn p? ?jp,`	3	`F2 xx xx`	10/10	`------`	`If !S then PC=nn`
`JP PE,nn`	`nn pe? ?jp,`	3	`EA xx xx`	10/10	`------`	`If P then PC=nn`
`JP PO,nn`	`nn po? ?jp,`	3	`E2 xx xx`	10/10	`------`	`If !P then PC=nn`
`JP Z,nn`	`nn z? ?jp,`	3	`CA xx xx`	10/10	`------`	`If Z then PC=nn`

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
JR C,n	nn c? ?jr,	2	38 xx	12/7	------	If CY then PC=PC+n
JR NC,n	nn nc? ?jr,	2	30 xx	12/7	------	If !CY then PC=PC+n
JR NZ,n	nn z? ?jr,	2	20 xx	12/7	------	If !Z then PC=PC+n
JR Z,n	nn z? ?jr,	2	28 xx	12/7	------	If Z then PC=PC+n
JR n	nn jr,	2	18 xx	12	------	PC=PC+n
LD (BC),A	b stap,	1	02	7	------	[BC]=A
LD (DE),A	d stap,	1	12	7	------	[DE]=A
LD (HL),n	n m ld#,	2	36 xx	10	------	[HL]=n
LD (HL),r	r m ld,	1	70+r	7	------	[HL]=r
LD (IX+n1),n2	n2 n1 ix st#x,	4	DD 36 xx xx	19	------	[IX+n]=n
LD (IX+n),r	r n ix stx,	3	DD 70+r xx	19	------	[IX+n]=r
LD (IY+n1),n2	n2 n1 iy st#x,	4	FD 36 xx xx	19	------	[IY+n]=n
LD (IY+n),r	r n iy stx,	3	FD 70+r xx	19	------	[IY+n]=r
LD (nn),A	nn sta,	3	32 xx xx	13	------	[nn]=A
LD (nn),BC	nn b stp,	4	ED 43 xx xx	20	------	[nn]=C, (nn+1)=B
LD (nn),DE	nn d stp,	4	ED 53 xx xx	20	------	[nn]=E, (nn+1)=D
LD (nn),HL	nn h sthl,	3	22 xx xx	16	------	[nn]=L, (nn+1)=H
LD (nn),HL	nn h stp,	3	ED 63 xx xx	20	------	[nn]=L, (nn+1)=H
LD (nn),IX	nn ix stp,	4	DD 22 xx xx	20	------	[nn,nn+1]=IX
LD (nn),IY	nn iy stp,	4	FD 22 xx xx	20	------	[nn,nn+1]=IY
LD (nn),SP	nn sp stp,	4	ED 73 xx xx	20	------	[nn,nn+1]=SP
LD A,(BC)	b ftap,	1	0A	7	------	A=[BC]
LD A,(DE)	d ftap,	1	1A	7	------	A=[DE]
LD A,(HL)	m a ld,	1	7E	7	------	A=[HL]
LD A,(IX+n)	n ix a ftx,	3	DD 7E xx	19	------	A=[IX+n]
LD A,(IY+n)	n iy a ftx,	3	FD 7E xx	19	------	A=[IY+n]
LD A,(nn)	nn fta,	3	3A xx xx	13	------	A=[nn]
LD A,I	ldai,	2	ED 57	9	**0*0-	A=I
LD A,n	n a ld#,	2	3E xx	7	------	A=n
LD A,R	ldar,	2	ED 5F	9	**0*0-	A=R
LD A,r	r a ld,	1	78+r	4	------	A=r
LD B,(HL)	m b ld,	1	46	7	------	B=[HL]
LD B,(IX+n)	n ix b ftx,	3	DD 46 xx	19	------	B=[IX+n]

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
LD B,(IY+n)	n iy b ftx,	3	FD 46 xx	19	------	B=[IY+n]
LD B,n	n b ld#,	2	06 xx	7	------	B=n
LD B,r	r b ld,	1	40+r	4	------	B=r
LD BC,(nn)	nn b ftp,	4	ED 4B xx xx	20	------	C=[nn],B=[nn+1]
LD BC,nn	nn b ldp#,	3	01 xx xx	10	------	BC=nn
LD C,(HL)	m c ld,	1	4E	7	------	C=[HL]
LD C,(IX+n)	n ix c ftx,	3	DD 4E xx	19	------	C=[IX+n]
LD C,(IY+n)	n iy c ftx,	3	FD 4E xx	19	------	C=[IY+n]
LD C,n	n c ld#,	2	0E xx	7	------	C=n
LD C,r	r c ld,	1	48+r	4	------	C=r
LD D,(HL)	m d ld,	1	56	7	------	D=[HL]
LD D,(IX+n)	n ix d ftx,	3	DD 56 xx	19	------	D=[IX+n]
LD D,(IY+n)	n iy d ftx,	3	FD 56 xx	19	------	D=[IY+n]
LD D,n	n d ld#,	2	16 xx	7	------	D=n
LD D,r	r d ld,	1	50+r	4	------	D=r
LD DE,(nn)	nn d ftp,	4	ED 5B xx xx	20	------	E=[nn],D=[nn+1]
LD DE,nn	nn d ldp#,	3	11 xx xx	10	------	DE=nn
LD E,(HL)	m e ld,	1	5E	7	------	E=[HL]
LD E,(IX+n)	n ix e ftx,	3	DD 5E xx	19	------	E=[IX+n]
LD E,(IY+n)	n iy e ftx,	3	FD 5E xx	19	------	E=[IY+n]
LD E,n	n e ld#,	2	1E xx	7	------	E=n
LD E,r	r e ld,	1	58+r	4	------	E=r
LD H,(HL)	m h ld,	1	66	7	------	H=[HL]
LD H,(IX+n)	n ix h ftx,	3	DD 66 xx	19	------	H=[IX+n]
LD H,(IY+n)	n iy h ftx,	3	FD 66 xx	19	------	H=[IY+n]
LD H,n	n h ld#,	2	26 xx	7	------	H=n
LD H,r	r h ld,	1	60+r	4	------	H=r
LD HL,(nn)	nn fthl,	3	2A xx xx	16	------	L=[nn],H=[nn+1]
LD HL,(nn)	nn h ftp,	4	ED 6B xx xx	20	------	L=[nn],H=[nn+1]
LD HL,nn	nn h ldp#,	3	21 xx xx	10	------	HL=nn
LD I,A	ldia,	2	ED 47	9	------	I=A
LD IX,(nn)	nn ix ftp,	4	DD 2A xx xx	20	------	IX=[nn,nn+1]
LD IX,nn	nn ix ldp#,	4	DD 21 xx xx	14	------	IX=nn

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
LD IY,(nn)	nn iy ftp,	4	FD 2A xx xx	20	------	IY=[nn,nn+1]
LD IY,nn	nn iy ldp#,	4	FD 21 xx xx	14	------	IY=nn
LD L,(HL)	m l ld,	1	6E	7	------	L=[HL]
LD L,(IX+n)	n ix l ftx,	3	DD 6E xx	19	------	L=[IX+n]
LD L,(IY+n)	n iy l ftx,	3	FD 6E xx	19	------	L=[IY+n]
LD L,n	n l ld#,	2	2E xx	7	------	L=n
LD L,r	r l ld,	1	68+r	4	------	L=r
LD R,A	ldra,	2	ED 4F	9	------	R=A
LD SP,(nn)	nn sp ftp,	4	ED 7B xx xx	20	------	SP=[nn,nn+1]
LD SP,HL	ldsp,	1	F9	6	------	SP=HL
LD SP,IX	n/a	2	DD F9	10	------	SP=IX
LD SP,IY	n/a	2	FD F9	10	------	SP=IY
LD SP,nn	nn sp ldp#,	3	31 xx xx	10	------	SP=nn
LDD	ldd,	2	ED A8	16	--0*0-	[DE]=[HL],HL-=1,DE-=1,BC-=1
LDDR	lddr,	2	ED B8	21/16	--000-	LDD until BC=0
LDI	ldi,	2	ED A0	16	--0*0-	[DE]=[HL],HL+=1,DE+=1,BC=-1
LDIR	ldir,	2	ED B0	21/16	--000-	LDI until BC=0
NEG	neg,	2	ED 44	8	***V1*	A=-A
NOP	nop,	1	00	4	------	
OR (HL)	m or,	1	B6	7	***P00	A=Av[HL]
OR (IX+n)	n ix orx,	3	DD B6 xx	19	***P00	A=Av[IX+n]
OR (IY+n)	n iy orx,	3	FD B6 xx	19	***P00	A=Av[IY+n]
OR n	n or#,	2	F6 xx	7	***P00	A=AvN
OR r	r or,	1	B0+r	4	***P00	A=Avr
OTDR	n/a	2	ED BB	21/16	01*?1-	OUTD until B=0
OTIR	n/a	2	ED B3	21/16	01*?1-	OUTI until B=0
OUT (C),A	a outbc,	2	ED 79	12	------	[C]=A
OUT (C),B	b outbc,	2	ED 41	12	------	[C]=B
OUT (C),C	c outbc,	2	ED 49	12	------	[C]=C
OUT (C),D	d outbc,	2	ED 51	12	------	[C]=D
OUT (C),E	e outbc,	2	ED 59	12	------	[C]=E
OUT (C),H	h outbc,	2	ED 61	12	------	[C]=H

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
OUT (C),L	l outbc,	2	ED 69	12	------	[C]=L
OUT (n),A	n out,	2	D3 xx	11	------	[n]=A
OUTD	n/a	2	ED AB	16	***?1-	[C]=[HL],HL=HL-1,B=B-1
OUTI	n/a	2	ED A3	16	***?1-	[C]=[HL],HL=HL+1,B=B-1
POP AF	a pop,	1	F1	10	******	F=[SP],SP+,A=[SP],SP+
POP BC	b pop,	1	C1	10	------	C=[SP],SP+,B=[SP],SP+
POP DE	d pop,	1	D1	10	------	E=[SP],SP+,D=[SP],SP+
POP HL	h pop,	1	E1	10	------	L=[SP],SP+,H=[SP],SP+
POP IX	ix pop,	2	DD E1	14	------	IX=[SP,SP+1],SP+,SP+
POP IY	iy pop,	2	FD E1	14	------	IY=[SP,SP+1],SP+,SP+
PUSH AF	a push,	1	F5	11	------	-SP,[SP]=A,-SP,[SP]=F
PUSH BC	b push,	1	C5	11	------	-SP,[SP]=B,-SP,[SP]=C
PUSH DE	d push,	1	D5	11	------	-SP,[SP]=D,-SP,[SP]=E
PUSH HL	h push,	1	E5	11	------	-SP,[SP]=H,-SP,[SP]=L
PUSH IX	ix push,	2	DD E5	15	------	-SP,-SP,[SP,SP+1]=IX
PUSH IY	iy push,	2	FD E5	15	------	-SP,-SP,[SP,SP+1]=IY
RES b,(HL)	m b res,	2	CB 86+8*b	15	------	[HL]=[HL]&{~2^b}
RES b,(IX+n)	n ix b resx,	4	DD CB xx 86+8*b	23	------	[IX+n]=[IX+n]&{~2^b}
RES b,(IY+n)	n iy b resx,	4	FD CB xx 86+8*b	23	------	[IY+n]=[IY+n]&{~2^b}
RES b,r	r b res,	2	CB 80+8*b+r	8	------	r=r&{~2^b}
RET	ret,	1	C9	10	------	PC=[SP,SP+1],SP+,SP+
RET C	c? ?ret,	1	D8	11/5	------	If CY then PC=[SP,SP+1],SP+=2
RET M	m? ?ret,	1	F8	11/5	------	If S then PC=[SP,SP+1],SP+=2
RET NC	nc? ?ret,	1	D0	11/5	------	If !CY then PC=[SP,SP+1],SP+=2
RET NZ	nz? ?ret,	1	C0	11/5	------	If !Z then PC=[SP,SP+1],SP+=2
RET P	p? ?ret,	1	F0	11/5	------	If !S then PC=[SP,SP+1],SP+=2
RET PE	pe? ?ret,	1	E8	11/5	------	If P then PC=[SP,SP+1],SP+=2
RET PO	po? ?ret,	1	E0	11/5	------	If !P then PC=[SP,SP+1],SP+=2
RET Z	z? ?ret,	1	C8	11/5	------	If Z then PC=[SP,SP+1],SP+=2

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
RETI	n/a	2	ED 4D	14	------	PC=[SP,SP+1],SP+,SP+
RETN	n/a	2	ED 45	14	------	PC=[SP,SP+1],SP+,SP+
RL (HL)	m rl,	2	CB 16	15	**0P0*	[HL]={CY,[HL]}<<CY
RL (IX+n)	n ix rlx,	4	DD CB xx 16	23	**0P0*	[IX+n]={CY,[IX+n]}<<CY
RL (IY+n)	n iy rlx,	4	FD CB xx 16	23	**0P0*	[IY+n]={CY,[IY+n]}<<CY
RL r	r rl,	2	CB 10+r	8	**0P0*	r={CY,r}<<CY
RLA	rla,	1	17	4	--0-0*	A={CY,A}<<CY
RLC (HL)	m rlc,	2	CB 06	15	**0P0*	[HL]={[HL]}<<
RLC (IX+n)	n ix rlcx,	4	DD CB xx 06	23	**0P0*	[IX+n]={[IX+n]}<<
RLC (IY+n)	n iy rlcx,	4	FD CB xx 06	23	**0P0*	[IY+n]={[IY+n]}<<
RLC r	r rlc,	2	CB 00+r	8	**0P0*	r={r}<<
RLCA	rlca,	1	07	4	--0-0*	A=*<<
RLD	rld,	2	ED 6F	18	**0P0-	{A,[HL]}={A,[HL]}←4
RR (HL)	m rr,	2	CB 1E	15	**0P0*	[HL]=CY>>{CY,[HL]}
RR (IX+n)	n ix rrx,	4	DD CB xx 1E	23	**0P0*	[IX+n]=CY>>{CY,[IX+n]}
RR (IY+n)	n iy rrx,	4	FD CB xx 1E	23	**0P0*	[IT+n]=CY>>{CY,[IY+n]}
RR r	r rr,	2	CB 18+r	8	**0P0*	r=CY>>{CY,r}
RRA	rra,	1	1F	4	--0-0*	A=CY>>{CY,A}
RRC (HL)	m rrc,	2	CB 0E	15	**0P0*	[HL]⇒>{[HL]}
RRC (IX+n)	n ix rrcx,	4	DD CB xx 0E	23	**0P0*	[IX+n]⇒>{[IX+n]}
RRC (IY+n)	n iy rrcx,	4	FD CB xx 0E	23	**0P0*	[IY+n]⇒>{[IY+n]}
RRC r	r rrc,	2	CB 08+r	8	**0P0*	r⇒>{r}
RRCA	rrca,	1	0F	4	--0-0*	A⇒>*
RRD	n/a	2	ED 67	18	**0P0-	{A,[HL]}=4→{A,[HL]}
RST 0	0 rst,	1	C7	11	------	-SP,-SP,[SP+1,SP]=PC,PC=00
RST 8H	$8 rst,	1	CF	11	------	-SP,-SP,[SP+1,SP]=PC,PC=08
RST 10H	$10 rst,	1	D7	11	------	-SP,-SP,[SP+1,SP]=PC,PC=10
RST 18H	$18 rst,	1	DF	11	------	-SP,-SP,[SP+1,SP]=PC,PC=18
RST 20H	$20 rst,	1	E7	11	------	-SP,-SP,[SP+1,SP]=PC,PC=20
RST 28H	$28 rst,	1	EF	11	------	-SP,-SP,[SP+1,SP]=PC,PC=28
RST 30H	$30 rst,	1	F7	11	------	-SP,-SP,[SP+1,SP]=PC,PC=30
RST 38H	$38 rst,	1	FF	11	------	-SP,-SP,[SP+1,SP]=PC,PC=38
SBC (HL)	m sbc,	1	9E	7	***V1*	A=A-[HL]-CY

Z80 instruction	Forth command	Size	Object code	Clock	SZHPNC	Effect
SBC A,(IX+n)	n ix sbcx,	3	DD 9E xx	19	***V1*	A=A-[IX+n]-CY
SBC A,(IY+n)	n iy sbcx,	3	FD 9E xx	19	***V1*	A=A-[IY+n]-CY
SBC A,n	n sbc#,	2	DE xx	7	***V1*	A=A-n-CY
SBC HL,BC	b sbcp,	2	ED 42	15	***V1*	HL=HL-BC-CY
SBC HL,DE	d sbcp,	2	ED 52	15	***V1*	HL=HL-DE-CY
SBC HL,HL	h sbcp,	2	ED 62	15	***V1*	HL=HL-HL-CY
SBC HL,SP	sp sbcp,	2	ED 72	15	***V1*	HL=HL-SP-CY
SBC r	r sbc,	1	98+r	4	***V1*	A=A-r-CY
SCF	scf,	1	37	4	--0-01	CY=1
SET b,(HL)	m b set,	2	CB C6+8*b	15	------	[HL]=[HL]v{2^b}
SET b,(IX+n)	n ix b setx,	4	DD CB xx C6+8*b	23	------	[IX+n]=[IX+n]v{2^b}
SET b,(IY+n)	n iy b setx,	4	FD CB xx C6+8*b	23	------	[IY+n]=[IY+n]v{2^b}
SET b,r	r b set,	2	CB C0+8*b+r	8	------	r=rv{2^b}
SLA (HL)	m sla,	2	CB 26	15	**0P0*	[HL]=[HL]*2
SLA (IX+n)	n ix sla,	4	DD CB xx 26	23	**0P0*	[IX+n]=[IX+n]*2
SLA (IY+n)	n iy sla,	4	FD CB xx 26	23	**0P0*	[IY+n]=[IY+n]*2
SLA r	r sla,	2	CB 20+r	8	**0P0*	r=r*2
SLL (HL)	m sll,	2	CB 36	15	**0P0*	[HL]=[HL]*2+1
SLL (IX+n)	n ix sllx,	4	DD CB xx 36	23	**0P0*	[IX+n]=[IX+n]*2+1
SLL (IY+n)	n iy sllx,	4	FD CB xx 36	23	**0P0*	[IY+n]=[IY+n]*2+1
SLL r	r sll,	2	CB 30+r	8	**0P0*	r=r*2+1
SRA (HL)	m sra,	2	CB 2E	15	**0P0*	[HL]=(signed)[HL]/2
SRA (IX+n)	n ix srax,	4	DD CB xx 2E	23	**0P0*	[IX+n]=(signed)[IX+n]/2
SRA (IY+n)	n iy srax,	4	FD CB xx 2E	23	**0P0*	[IY+n]=(signed)[IY+n]/2
SRA r	r sra,	2	CB 28+r	8	**0P0*	r=(signed)r/2
SRL (HL)	m sra,	2	CB 3E	15	**0P0*	[HL]=(unsigned)[HL]/2
SRL (IX+n)	n ix srlx,	4	DD CB xx 3E	23	**0P0*	[IX+n]=(unsigned)[IX+n]/2
SRL (IY+n)	n iy srlx,	4	FD CB xx 3E	23	**0P0*	[IY+n]=(unsigned)[IY+n]/2
SRL r	r srl,	2	CB 38+r	8	**0P0*	r=(unsigned)r/2
SUB (HL)	m sub,	1	96	7	***V1*	A=A-[HL]
SUB (IX+n)	n ix subx,	3	DD 96 xx	19	***V1*	A=A-[IX+n]
SUB (IY+n)	n iy subx,	3	FD 96 xx	19	***V1*	A=A-[IY+n]
SUB n	n sub#,	2	D6 xx	7	***V1*	A=A-n

Z80 instruction	Forth command	Size	Object code	Clock	`SZHPNC`	Effect
`SUB r`	`r sub,`	1	`90+r`	4	`***V1*`	`A=A-r`
`XOR (HL)`	`m xor,`	1	`AE`	7	`***P00`	`A=Ax[HL]`
`XOR (IX+n)`	`n ix xorx,`	3	`DD AE xx`	19	`***P00`	`A=Ax[IX+n]`
`XOR (IY+n)`	`n ix xorx,`	3	`FD AE xx`	19	`***P00`	`A=Ax[IY+n]`
`XOR n`	`n xor#,`	2	`EE xx`	7	`***P00`	`A=AxN`
`XOR r`	`r xor,`	1	`A8+r`	4	`***P00`	`A=Axr`

Legend

Clock

The time it takes to execute the instruction in CPU cycles. If there are two numbers given for Clock, then the highest is when the jump is taken, the lowest is when it skips the jump.

Size

How many bytes the instruction takes up in a program.

SZHPNC

How the different Z80 flags (bits of the "F" register) are affected (**S**=Sign, **Z**=Zero, **H**=Half Carry, **P**=Parity/Overflow, **N**=Add/Subtract, **C**=Carry):

Table 14. Flag effect symbols

Symbol	Meaning
`-`	Flag unaffected
`*`	Flag affected
`0`	Flag reset
`1`	Flag set
`?`	Unknown
`P`	Parity/Overflow flag used as parity
`V`	Parity/Overflow flag used as overflow

Object code

The equivalent machine code instruction in hexadecimal, with "xx" instead of the parameters (e.g. addresses or bytes), and some calculations based on certain parameters (e.g. registers or bit numbers).

`b`

Bit. Can be 0-7.

`r`

Register. Can be "B", "C", "D", "E", "H", "L" or "A".

Table 15. Register values in opcodes

Register	Value of `r` in the object code
B	0
C	1
D	2
E	3
H	4
L	5
A	7

NOTE The Solo Forth’s Z80 `assembler` treats `(HL)` as a register named `m`, with value 6.

Glossary

!

!

```
! ( x a -- ) "store"
```

Store *x* at *a*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `@`, `+!`, `2!`, `c!`.

Source file: <src/kernel.z80s>.

!>

```
!>
  Interpretation: ( x "name" -- )
  Compilation:    ( "name" -- )
  Run-time:       ( x -- )
"store-to"
```

A simpler and faster alternative to standard `to` and `value`.

`!>` is an `immediate` word.

Interpretation:

Parse *name*, which is the name of a word created by `constant` or `const`, and make *x* its value.

Compilation:

Parse *name*, which is a word created by `constant` or `const`, and append the run-time semantics given below to the current definition.

Run-time:

Make *x* the current value of constant *name*.

Origin: IsForth.

See also: `c!>`, `2!>`.

Source file: <src/lib/data.store-to.fs>.

!a

```
!a ( x -- ) "store-a"
```

Store *x* at the address register.

See also: `a`, `@a`.

Source file: <src/lib/memory.address_register.fs>.

!a+

```
!a+ ( x -- ) "store-a-plus"
```

Store *x* at the address register and increment the address register by one cell.

See also: `a`, `@a+`.

Source file: <src/lib/memory.address_register.fs>.

!bank

```
!bank ( x a n -- ) "store-bank"
```

Store cell *x* into address *a* ($C000..$FFFF) of `bank` *n*.

`!bank` is written in Z80. Its equivalent definition in Forth is the following:

```
: !bank ( x a n -- ) bank ! default-bank ;
```

See also: `@bank`, `c!bank`.

Source file: <src/lib/memory.far.fs>.

!bit

```
!bit ( f b ca -- ) "store-bit"
```

Store flag *f* in an element of a bit-array, represented by address *ca* and bitmask *b*.

See also: `@bit`, `bit-array`.

Source file: <src/lib/data.array.bit.fs>.

!csp

```
!csp ( -- ) "store-c-s-p"
```

Save the current data stack position, `sp@`, into `csp`, to be checked later by `?csp`. `!csp` is used by `:`, `:noname` and `asm` for error checking.

Definition:

```
: !csp ( -- ) sp@ csp ! ;
```

Origin: fig-Forth.

Source file: <src/kernel.z80s>.

!dos

```
!dos ( x a -- ) "store-dos"
```

Store *x* at the Plus D memory address *a*.

See also: `@dos`, `c!dos`.

Source file: <src/lib/dos.gplusdos.fs>.

!dosvar

```
!dosvar ( x n -- ) "store-dos-var"
```

Store *x* into the G+DOS variable *n*.

See also: `@dosvar`, `c!dosvar`, `!dos`.

Source file: <src/lib/dos.gplusdos.fs>.

!exchange

```
!exchange ( x1 a -- x2 ) "store-exchange"
```

Store *x1* into *a* and return its previous contents *x2*.

See also: `c!exchange`, `exchange`.

Source file: <src/lib/memory.MISC.fs>.

!p

```
!p ( b a -- ) "store-p"
```

Output byte *b* to port *a*.

See also: `@p`, `!`, `c!`.

Source file: <src/lib/memory.ports.fs>.

!sound

```
!sound ( b1 b2 -- ) "store-sound"
```

Set sound register *b2* (0…13) to value *b1*.

See also: `@sound`, `sound`, `play`, `sound-register-port`, `sound-write-port`.

Source file: <src/lib/sound.128.fs>.

!volume

```
!volume ( b1 b2 -- ) "store-volume"
```

Store *b1* at volume register of channel *b2* (0..2, equivalent to notation 'A'..'C').

> Registers 8..10 (Channels A..C Volume)
>
> | | |
> |---|---|
> | **Bits 0-4** | Channel volume level. |
> | **Bit 5** | 1=Use envelope defined by register 13 and ignore the volume setting. |
> | **Bits 6-7** | Not used. |

See also: `@volume`, `!sound`.

Source file: <src/lib/sound.128.fs>.

"

"n"

```
"n" ( -- c ) "quote-n-quote"
```

A character constant containing the (lowercase) character used by `y/n`, `y/n?` and `no?` to represent a negative answer. By default it's "n". For localization, the value can be changed with `c!>`.

See also: `"y"`.

Source file: <src/lib/keyboard.yes-question.fs>.

"y"

```
"y" ( -- c ) "quote-y-quote"
```

A character constant containing the (lowercase) character used by `y/n`, `y/n?` and `yes?`, to represent an affirmative answer. By default it's "y". For localization, the value can be changed with `c!>`.

See also: `"n"`.

Source file: <src/lib/keyboard.yes-question.fs>.

#

#

```
# ( ud1 -- ud2 ) "number-sign"
```

Divide *ud1* by the number in `base`, giving the quotient *ud2* and the remainder *n*. (*n* is the least significant digit of *ud1*.) Convert *n* to external form and add the resulting character to the beginning of the pictured numeric output string that was started by `<#`.

`#` is tipically used between `<#` and `#>`.

Definition:

```
: # ( ud1 -- ud2 ) base @ ud/mod rot >digit hold ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `hold`, `ud/mod`, `>digit`.

Source file: <src/kernel.z80s>.

#>

```
#> ( xd -- ca len ) "number-sign-greater"
```

End the pictured numeric output conversion that was started by `<#`: Drop *xd* and make the pictured numeric output string available as the string *ca len*.

Definition:

```
: #> ( xd -- ca len ) 2drop hld @ pad over - ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `#`, `#s`, `hold`, `hld`, `sign`, `pad`.

Source file: <src/kernel.z80s>.

#block-drives

```
#block-drives ( -- ca ) "number-sign-block-drives"
```

A `cvariable`. *ca* is the address of a byte containing the number of block drives defined in `block-drives`, i.e. the number of drives that are used for blocks. `#block-drives` could be modified manually, provided also `block-drives` is configured accordingly, but `set-block-drives` is provided for that.

The default value of `#block-drives` is 1, which is restored by `cold`.

See also: `max-blocks`.

Source file: <src/kernel.z80s>.

#chars

```
#chars ( ca len c -- +n ) "dash-chars"
```

Return the count *+n* of chars *c* in a string *ca len*.

See also: `#spaces`, `char-in-string?`, `char-position?`.

Source file: <src/lib/strings.MISC.fs>.

#do

```
#do Compilation: ( -- do-sys ) "dash-do"
```

Execute `0 ?do` and leave *do-sys* to be consumed by `loop` or `+loop`.

`#do` is an `immediate` and `compile-only` word.

Usage example:

```
: times ( n -- ) #do i . loop ;

0 times \ prints nothing
4 times \ prints 0 1 2 3
```

See also: `?do`, `do`, `-do`.

Origin: Comus.

Source file: <src/lib/flow.do.fs>.

#esc-order

```
#esc-order ( -- a ) "number-sign-esc-order"
```

A `variable`. *a* is the address of a cell containing the number of word lists in the escaped strings search order.

See also: `esc-context`, `max-esc-order`, `get-esc-order`, `set-esc-order`, `>esc-order`.

Source file: <src/lib/strings.escaped.fs>.

#indented

```
#indented ( -- a ) "number-sign-indented"
```

A `variable`. *a* is the address of a cell containing the numbers of characters indented on the current line.

See also: `#ltyped`, `indented+`.

Source file: <src/lib/display.ltype.fs>.

#kk

```
#kk ( -- n ) "dash-k-k"
```

A `cconstant`. *n* is the number of keyboard keys, i.e. the number of physical rubber keys on the keyboard of the original ZX Spectrum: 40.

See also: `kk-ports`, `kk-chars`, `kk-0#`, `kk-0`, `kk-1#`, `kk-1`, `kk-2#`, `kk-2`, `kk-3#`, `kk-3`, `kk-4#`, `kk-4`, `kk-5#`, `kk-5`, `kk-6#`, `kk-6`, `kk-7#`, `kk-7`, `kk-8#`, `kk-8`, `kk-9#`, `kk-9`, `kk-a#`, `kk-a`, `kk-b#`, `kk-b`, `kk-c#`, `kk-c`, `kk-cs#`, `kk-cs`, `kk-d#`, `kk-d`, `kk-e#`, `kk-e`, `kk-en#`, `kk-en`, `kk-f#`, `kk-f`, `kk-g#`, `kk-g`, `kk-h#`, `kk-h`, `kk-i#`, `kk-i`, `kk-j#`, `kk-j`, `kk-k#`, `kk-k`, `kk-l#`, `kk-l`, `kk-m#`, `kk-m`, `kk-n#`, `kk-n`, `kk-o#`, `kk-o`, `kk-p#`, `kk-p`, `kk-q#`, `kk-q`, `kk-r#`, `kk-r`, `kk-s#`, `kk-s`, `kk-sp#`, `kk-sp`, `kk-ss#`, `kk-ss`, `kk-t#`, `kk-t`, `kk-u#`, `kk-u`, `kk-v#`, `kk-v`, `kk-w#`, `kk-w`, `kk-x#`, `kk-x`, `kk-y#`, `kk-y`, `kk-z`.

Source file: <src/lib/keyboard.MISC.fs>.

#lag

```
#lag ( -- ca n ) "number-sign-lag"
```

Part of `specforth-editor`: Return cursor address *ca* and count *n* after cursor till end of line.

See also: `#lead`.

Source file: <src/lib/prog.editor.specforth.fs>.

#lead

```
#lead ( -- a n ) "number-sign-lead"
```

Part of `specforth-editor`: From the cursor pointer `r#` compute the line address *a* in the block buffer and the offset from *a* to the cursor location *n*.

See also: `#locate`, `#lag`.

Source file: <src/lib/prog.editor.specforth.fs>.

#locate

```
#locate ( -- n1 n2 ) "number-sign-locate"
```

Part of `specforth-editor`: From the cursor pointer `r#` compute the line number *n2* and the character offset *n1* in line number *n2*.

See also: `#lead`, `c/l`.

Source file: <src/lib/prog.editor.specforth.fs>.

#ltyped

```
#ltyped ( -- a ) "l-typed-number-sign"
```

A `variable` . *a* is the address of a cell containing the number of characters displayed by `ltype` on the current row.

See also: `ltyped`, `#indented`.

Source file: <src/lib/display.ltype.fs>.

#order

```
#order ( -- a ) "number-sign-order"
```

A `user` variable. *a* is the address of a cell containing the number of word lists in the search `order`.

See also: `context`, `max-order`, `get-order`, `set-order`, `>order`, `wordlist`.

Source file: <src/kernel.z80s>.

#s

```
#s ( ud1 -- ud2 ) "number-sign-s"
```

Convert one digit of *ud1* according to the rule for `#`. Continue conversion until the quotient is zero. *ud2* is zero. Used between `<#` and `#>`.

Definition:

```
#s ( ud1 -- ud2 ) begin # 2dup or 0until ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

Source file: <src/kernel.z80s>.

#spaces

```
#spaces ( ca len -- +n ) "dash-spaces"
```

Count number *+n* of spaces in a string *ca len*.

See also: `#chars`, `spaces`.

Source file: <src/lib/strings.MISC.fs>.

#tib

```
#tib ( -- a ) "number-sign-t-i-b"
```

A `variable`. *a* is the address of a cell containing the number of characters in 'tib', the terminal input buffer.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE EXT, obsolescent).

See also: `/tib`.

Source file: <src/kernel.z80s>.

#words

```
#words ( -- n ) "number-sign-words"
```

n is the number of words currently defined in the system, which is updated by `header,`.

See also: `fyi`, `greeting`, `cold`.

Source file: <src/kernel.z80s>.

%

%

```
% ( n1 n2 -- n3 ) "per-cent"
```

n1 is percentage *n3* of *n2*.

See also: `u%`, `*/`.

Source file: <src/lib/math.operators.1-cell.fs>.

'

'

```
' ( "name" -- xt ) "tick"
```

If *name* is found in the current search order, return its execution token *xt*, else `throw` an exception.

Definition:

```
: ' ( "name" -- xt ) defined dup ?defined name> ;
```

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `[']`, `''`, `defined`, `?defined`, `>`.

Source file: <src/kernel.z80s>.

''

```
'' ( "name" -- xtp ) "tick-tick"
```

If *name* is found in the current search order, return its execution-token pointer *xtp*, else `throw` an exception.

Since aliases share the execution token of their original word, it’s not possible to get the name of an alias from its execution token. But `''` can do it:

```
' drop alias discard
' discard >name .name       \ this prints "drop"
'' discard >>name .name     \ this prints "discard"
```

See also: `['']`, `'`.

Source file: <src/lib/compilation.fs>.

'bs'

```
'bs' ( -- c ) "tick-b-s-tick"
```

A character constant that returns the caracter code used as backspace (8).

See also: `'cr'`, `'tab'`.

Source file: <src/lib/display.control.fs>.

'cr'

```
'cr' ( -- c ) "tick-c-r-tick"
```

A character constant that returns the caracter code used as carriage return (13).

See also: `cr`, `crs`, `newline`, `'lf'`.

Source file: <src/lib/display.control.fs>.

'lf'

```
'lf' ( -- c ) "tick-l-f-tick"
```

A character constant that returns the caracter code used as line feed (10).

NOTE In the ZX Spectrum’s character set, control character code 10 is not called "line feed" but "cursor down", which is analogous.

See also: `cr`, `newline`.

Source file: <src/lib/display.control.fs>.

'line

```
'line ( -- ca len )
```

Part of the `gforth-editor`: Return the rest of the current line, from the current position.

See also: `'rest`, `c/l`, `'par`.

Source file: <src/lib/prog.editor.gforth.fs>.

'par

```
'par ( buf "ccc<eol>" -- ca len )
```

Part of the `gforth-editor`: Parse *ccc*. If the result string is empty, discard it and return the counted string at *buf*; else return the parsed string and also store it at *buf* as a counted string.

See also: `'rest`, `'line`.

Source file: <src/lib/prog.editor.gforth.fs>.

'rest

```
'rest ( -- ca len )
```

Part of the `gforth-editor`: Return the rest of the current screen, from the current position.

See also: `'line`, `'par`, `scr`, `block`, `b/buf`.

Source file: <src/lib/prog.editor.gforth.fs>.

'tab'

```
'tab' ( -- c ) "tick-tab-tick"
```

A character constant that returns the caracter code used as tabulator (6).

See also: `tab`, `'cr'`, `'bs'`.

Source file: <src/lib/display.control.fs>.

(

(

```
( ( "ccc<paren>" -- ) "paren"
```

Parse *ccc* delimited by a right parenthesis. The number of characters in *ccc* may be zero to the number of characters in the parse area.

`(` is an `immediate` word.

Definition:

```
: ( ( "ccc<paren>" -- ) ')' parse 2drop ; immediate
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `\`, `parse`.

Source file: <src/kernel.z80s>.

((cat

```
((cat ( a -- ior ) "paren-paren-cat"
```

Show a disk catalogue of the current drive, calling the corresponding G+DOS hook command, which uses the data in UFIA at *a*, which was already copied into `ufia1`. Return I/O result code *ior*.

`((cat` is a low-level factor of `(cat`.

Source file: <src/lib/dos.gplusdos.fs>.

(+ato

```
(+ato ( n1 n2 xt -- ) "paren-plus-a-to"
```

Add *n1* to element *n2* of 1-dimension single-cell values array *xt*.

See also: `avalue`, `+ato`.

Source file: <src/lib/data.array.value.fs>.

(+cato

```
(+cato ( c n xt -- ) "paren-plus-c-a-to"
```

Add *c* to element *n* of 1-dimension character values array *xt*.

See also: `cavalue`, `+cato`.

Source file: <src/lib/data.array.value.fs>.

(+loop

```
(+loop ( n -- ) ( R: loop-sys1 -- loop-sys2 ) "paren-plus-loop"
```

Add *n* to the loop index. If the loop index did not cross the boundary between the loop limit minus one and the loop limit, continue execution at the beginning of the loop. Otherwise, discard the loop parameters and continue execution immediately following the loop.

`(+loop` is compiled by `+loop`.

See also: `(loop`.

Source file: <src/kernel.z80s>.

(-do

```
(-do ( n1|u1 n2|u2 -- ) ( R: -- loop-sys | ) "paren-minus-do"
```

If *n1*|*u1* is not less than *n2*|*u2*, discard both parameters and continue execution at the location given by the consumer of the *do-sys* left by `-do` at compilation time. Otherwise set up loop control parameters *loop-sys* with index *n2*|*u2* and limit *n1*|*u1* and continue executing immediately following `-do`. Anything already on the return stack becomes unavailable until the loop control parameters *loop_sys* are discarded.

`(-do` is compiled by `-do`.

Source file: <src/lib/flow.do.fs>.

(."

```
(." ( -- ) "paren-dot-quote"
```

Type the compiled string that follows. `(."` is the run-time procedure compiled by `."`.

Definition:

```
: (." ( -- ) r@ count dup char+ r> + >r type ;
```

See also: `,"`, `type`, `count`.

Source file: <src/kernel.z80s>.

(.word

```
(.word ( nt -- ) "paren-dot-word"
```

Default action of `.word`: display the name of the definition *nt* and execute `tab`.

Source file: <src/lib/tool.list.words.fs>.

(.xs

```
(.xs ( -- ) "paren-dot-x-s"
```

Display a list of the items in the current `xstack`; TOS is the right-most item.

`(.xs` is a factor of `.xs`.

Source file: <src/lib/data.xstack.fs>.

(0-1-8-color.

```
(0-1-8-color. ( n c -- ) "paren-zero-one-eight-color-dot"
```

`emit` control character *c*. Then convert *n* to the set 0, 1 and 8 and `emit` it. The conversion of *n* is done as follows:

- 0, 1 and 8 are not changed.
- 2, 4 and 6 are converted to 0.
- 3, 5 and 7 are converted to 1.

- Values greater than 8 or less than 0 are converted to 8.

This word is a factor of `flash.` and `bright.`.

Source file: <src/lib/display.attributes.fs>.

(0-9-color.

```
(0-9-color. ( -- a ) "paren-zero-nine-color-dot"
```

Return the address *a* of a routine used by `paper.` and `ink.`. This routine prints a color attribute in the range 0..9.

Input: - A = attribute control char ($10 for ink, $11 for paper) - TOS = attribute value (0..9)

NOTE If TOS is greater than 9, 9 is used instead.

Source file: <src/lib/display.attributes.fs>.

(2ato

```
(2ato ( xd n xt -- ) "paren-two-a-to"
```

Store *xd* into element *n* of 1-dimension double-cell values array *xt*.

See also: `2ato`.

Source file: <src/lib/data.array.value.fs>.

(;code

```
(;code ( -- ) ( R: a -- ) "paren-semicolon-code"
```

Rewrite the code field of the most recently defined high-level word (it cannot be a `code` word) to point to the following machine code sequence, which is at *a*.

`(;code` is the run-time procedure compiled by `;code` and `does>`.

Definition:

```
: (;code ( -- ) ( R: a -- ) r> latestxt 1+ ! ;
```

See also: `latestxt`.

Source file: <src/kernel.z80s>.

(>drive-block

```
(>drive-block ( u1 -- u2 ) "paren-to-drive-block"
```

Convert block *u1* to its equivalent *u2* in its corresponding disk drive, which is set the current drive.

`(>drive-block` becomes the action of `>drive-block` after `block-drives` has been loaded.

See also: `?drive#`, `?block-drive`, `set-drive`, `set-block-drives`.

Source file: <src/lib/dos.COMMON.fs>.

(>file

```
(>file ( -- ior ) "paren-to-file"
```

Save a file to disk using the data hold in `ufia` and return the I/O result code *ior*.

`(>file` is a factor of `>file`.

Source file: <src/lib/dos.gplusdos.fs>.

(>tape-file

```
(>tape-file ( -- ) "paren-to-tape-file"
```

Write a tape file using the data stored at `tape-header`.

`(>tape-file` is a factor of `>tape-file`.

Source file: <src/lib/tape.fs>.

(?ccase

```
(?ccase ( c ca len -- ) "paren-question-c-case"
```

Run-time procedure compiled by `?ccase`. If *c* is in the string *ca len*, execute the n-th word compiled after `?ccase`, where *n* is the position of the first *c* in the string (0..len-1). If *c* is not in *ca len*, do nothing.

Source file: <src/lib/flow.ccase.fs>.

(?do

```
(?do ( n1|u1 n2|u2 -- ) ( R: -- loop-sys | ) "paren-question-do"
```

If *n1|u1* is equal to *n2|u2*, continue execution at the location given by the consumer of the *do-sys* left by `?do` at compilation time. Otherwise set up loop control parameters *loop-sys* with index *n2|u2* and limit *n1|u1* and continue executing immediately following `?do`. Anything already on the return stack becomes unavailable until the loop control parameters *loop_sys* are discarded.

`(?do` is compiled by `?do`.

See also: `(do`, `(-do`.

Source file: <src/kernel.z80s>.

(abort

```
(abort ( -- ) "paren-abort"
```

Restart the system by emptying the stack and performing `quit`.

Definition:

```
: (abort ( -- ) empty-stack boot quit ;
```

See also: `error`, `abort`, `boot`, `empty-stack`.

Source file: <src/kernel.z80s>.

(abort"

```
(abort"  ( x -- ) "paren-abort-quote"
```

If *x* is not zero, perform the function of `-2 throw`, displaying the string that was compiled inline by `abort"`.

`(abort"` is the run-time procedure compiled by `abort"`.

See also: `throw`.

Source file: <src/lib/exception.fs>.

(aif

```
(aif ( op -- orig cs-id ) "paren-a-if"
```

Compile the Z80 `assembler` absolute-jump instruction *op* and put the location of a new unresolved forward reference *orig* and the `assembler` control-structure identifier *cs_id* onto the stack, to be consumed by `aelse` or `athen`.

op was left by any of the following `assembler` conditions: `nz?`, `z?`, `nc?`, `c?`, `po?`, `pe?`, `p?`, `m?`.

`(aif` is a factor of `aif` and `aelse`.

See also: `>mark`.

Source file: <src/lib/assembler.fs>.

(any-of

```
(any-of ( x#0 x#1 ... x#n n -- x#0 x#0 | x#0 0 ) "paren-any-of"
```

The run-time factor of `any-of`. If *x#0* equals any of *x#1 … x#n*, return *x#0 x#0*; else return *x#0 0*.

Source file: <src/lib/flow.case.fs>.

(at-xy

```
(at-xy  ( col row -- ) "paren-at-x-y"
```

Set the cursor coordinates to column *col* and row *row*, by displaying control character 22 followed by *col* and *row*, as needed by some display modes, e.g. `mode-64ao` and `mode-42pw`. The upper left corner is column zero, row zero.

`(at-xy` is a possible action of `at-xy`, which is a deferred word (see `defer`) configured by the current display mode.

WARNING	The default `mode-32` expects *row* right after control character 22, and then *col*, i.e in the order used by Sinclair BASIC. This will be fixed/unified in a future version of Solo Forth.

Source file: <src/lib/display.mode.COMMON.fs>.

(ato

```
(ato ( x n xt -- ) "paren-a-to"
```

Store *x* into element *n* of 1-dimension single-cell values array *xt*.

See also: `ato`.

Source file: <src/lib/data.array.value.fs>.

(auntil

```
(auntil ( dest cs-id op ) "paren-a-until"
```

Compile a Z80 `assembler` conditional absolute-jump opcode *op*.

`(auntil` is a factor of `auntil` and `aagain`.

Source file: <src/lib/assembler.fs>.

(baden-sqrt

```
(baden-sqrt ( n1 -- n2 n3 ) "paren-baden-square-root"
```

Integer square root *n3* of radicand *n1* with remainder *n2*. `(baden-sqrt` is a factor of `baden-sqrt`.

Source file: <src/lib/math.operators.1-cell.fs>.

(between-of

```
(between-of ( x1 x2 x3 -- x1 x1 | x1 x4 ) "paren-between-of"
```

The run-time factor of `between-of`. If *x1* is in range *x2 x3*, as calculated by `between`, return *x1 x1*; otherwise return *x1 x4*, being *x4* not equal to *x1*.

Source file: <src/lib/flow.case.fs>.

(bye

```
(bye ( -- ) "paren-bye"
```

Restore the two lower lines of the screen, as expected by BASIC, set interrupt mode 1, restore the OS stack pointer, restore the alternate HL Z80 register, and finally force a "STOP" BASIC error in order to return control to the host OS.

`(bye` is the final low-level procedure of `bye`.

Source file: <src/kernel.z80s>.

(c

```
(c ( ca len -- ) "paren-c"
```

Copy the string *ca len* to the cursor line at the cursor position. `(c` is a factor of `c`.

See also: `#lag`, `r#`, `#lead`, `cmove`, `update`.

Source file: <src/lib/prog.editor.specforth.fs>.

(cat

```
(cat ( b -- ) "paren-cat"
```

Show a disk catalogue of the current drive, using the data in `ufia`, being *b* the type of catalogue:

- $02 = abbreviated
- $04 = detailed
- $12 = abbreviated with wild-card
- $14 = detailed with wild-card

`(cat` is the common factor of all words that show disk catalogues: `cat`, `acat`, `wcat`, `wacat`.

See also: `((cat`, `set-drive`.

Source file: <src/lib/dos.gplusdos.fs>.

(cato

```
(cato ( c n xt -- ) "paren-c-a-to"
```

Store *c* into element *n* of 1-dimension character values array *xt*.

See also: `cato`.

Source file: <src/lib/data.array.value.fs>.

(ccase

```
(ccase ( c ca len -- ) "paren-c-case"
```

Run-time procedure compiled by `ccase`. If *c* is in the string *ca len*, execute the n-th word compiled after `ccase`, where *n* is the position of the first *c* in the string (0..len-1). If *c* is not in *ca len*, execute the word compiled right before `endccase`.

Source file: <src/lib/flow.ccase.fs>.

(ccase0

```
(ccase0 ( c ca len -- ) "paren-c-case-zero"
```

Run-time procedure compiled by `ccase0`. If *c* is in the string *ca len*, execute the n-th word compiled after `ccase0`, where *n* is the position of the first *c* in the string (0..len-1) plus 1. If *c* is not in *ca len*, execute the word compiled right after `ccase0`.

Source file: <src/lib/flow.ccase.fs>.

(comp'

```
(comp' ( nt -- xt ) "paren-comp-tick"
```

A factor of `name>compile`. If *nt* is an `immediate` word, return the *xt* of `execute`, else return the *xt* of `compile,`.

See also: `immediate?`.

Source file: <src/lib/compilation.fs>.

(cr

```
(cr ( -- ) "paren-c-r"
```

Transmit a carriage return to the selected output device. `(cr` is the default action of the deferred word `cr` (see `defer`).

Source file: <src/kernel.z80s>.

(d.

```
(d. ( d n -- ca len ) "paren-d-dot"
```

Convert *d* to an unsigned number in the current `base`, with *n* digits, as string *ca len*.

See also: `(dbin.`, `(dhex.`.

Source file: <src/lib/display.numbers.fs>.

(dbin.

```
(dbin. ( d n -- ) "paren-d-bin-dot"
```

Display *d* as an unsigned binary number with *n* digits.

See also: `(dhex.`, `32bin.`, `16bin.`, `8bin.`, `bin.`.

Source file: <src/lib/display.numbers.fs>.

(defer

```
(defer ( -- ) "paren-defer"
```

`throw` `error` #-261 ("deferred word is uninitialized". `(defer` is the default action of the uninitialized deferred words (see `defer`).

Definition:

```
: (defer ( -- ) #-261 error ;
```

Source file: <src/kernel.z80s>.

(delete-file

```
(delete-file ( -- ior ) "paren-delete-file"
```

Delete a disk file using the data hold in `ufia`. Return the I/O result code *ior*.

`(delete-file` is a factor of `delete-file`.

Source file: <src/lib/dos.gplusdos.fs>.

(dhex.

```
(dhex. ( d n -- ) "paren-d-hex-dot"
```

Display *d* as an unsigned hexadecimal number with *n* digits.

See also: `(dbin.`, `32hex.`, `16hex.`, `8hex.`, `hex.`.

Source file: <src/lib/display.numbers.fs>.

(do

```
(do ( n1|u1 n2|u2 -- ) ( R: -- loop-sys ) "paren-do"
```

Set up loop control parameters *loop-sys* with index *n2*|*u2* and limit *n1*|*u1* and continue executing immediately following `do`. Anything already on the return stack becomes unavailable until the loop control parameters *loop_sys* are discarded.

`(do` is compiled by `do`.

See also: `(?do`, `(-do`.

Source file: <src/kernel.z80s>.

(dstep

```
(dstep ( R: x ud -- x ud' | x ) "paren-d-step"
```

The run-time procedure compiled by `dstep`.

If the loop index *ud* is zero, discard it and continue execution after the loop. Otherwise decrement the loop index and continue execution at the beginning of the loop.

Source file: <src/lib/flow.dfor.fs>.

(file-status

```
(file-status ( -- a ior ) "paren-file-status"
```

Return the status of the file whose name is hold in `ufia`. If the file exists, *ior* is zero and the file header is read into *a*, which is the address returned by `ufia`, Otherwise *ior* is the I/O result code. and *a* is undefined.

NOTE Only the 9-byte header in `ufia` is updated, i.e. `hd00`, `hd0b`, `hd0d`, `hd0f` and `hd11`.

`(file-status` is a low-level factor of `file-status`.

Source file: <src/lib/dos.gplusdos.fs>.

(file>

```
(file> ( ca len -- ior ) "paren-file-from"
```

Read a file from disk, using the data hold in `ufia` and the alternative destination zone *ca len*, following the following two rules:

1. If *len* is not zero, use it as the count of bytes that must be read from the file defined in `ufia` and use *ca* as destination address.
2. If *len* is zero, use the file length stored in the file header instead, and then check also *ca*: If *ca* is not zero, use it as destination address, else use the file address stored in the file header instead.

Return the I/O result code *ior*.

`(file>` is a factor of `file>`.

See also: `file-length`.

Source file: <src/lib/dos.gplusdos.fs>.

(fp@

```
(fp@ ( -- fa ) "paren-f-p-fetch"
```

Return the address *fa* above the top of the floating-point stack. `(fp@` is a factor of `fp@`.

See also: `fp`.

Source file: <src/lib/math.floating_point.rom.fs>.

(g-emit

```
(g-emit ( c -- ) "paren-g-emit"
```

Display character *c* (32..127) at the current graphic coordinates.

The character is printed with overprinting (equivalent to `1 overprint`).

See also: `g-emit`, `g-emit_`.

Source file: <src/lib/display.g-emit.fs>.

(gigatype

```
(gigatype ( ca len a1 a2 -- ) "paren-gigatype"
```

If *len* is greater than zero, display text string *ca len* at screen address *a1* using the current fonts, doubled pixels (16x16 pixels per character) and modifying the characters on the fly after style data table *a2*.

`(gigatype` is written in Z80 and it's the low-level procedure of `gigatype`.

Source file: <src/lib/display.gigatype.fs>.

(greater-of

```
(greater-of ( n1 n2 -- n1 n1 | n1 n3 ) "paren-greater-of"
```

The run-time factor of `greater-of`.

If *n1* is greater than *n2*, leave *n1 n1*; otherwise leave *n1 n3*, being *n3* not equal to *n1*.

See also: `(less-of`.

Source file: <src/lib/flow.case.fs>.

(heap-in

```
(heap-in  ( -- ) "paren-heap-in"
```

If the current `heap` was created by `bank-heap`, page in its bank, which is stored at `heap-bank`; else do nothing.

`(heap-in` is the action of `heap-in`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

(home

```
(home ( -- ) "paren-home"
```

Default action of `home`: Set the cursor position at the top left position (column 0, row 0).

Source file: <src/kernel.z80s>.

(index-block

```
(index-block ( u -- ) "paren-index-block"
```

Index block *u*, evaluating its header line. The only word list in the search order must be `index-wordlist`.

`(index-block` is a common factor of `index-block` and `(make-thru-index`.

Source file: <src/lib/blocks.indexer.COMMON.fs>.

(jr,

```
(jr, ( a op -- ) "paren-j-r-comma"
```

Compile a Z80 `assembler` relative-jump intruction *op* to the absolute address *a*.

`(jr,` is a factor of `jr,`.

Source file: <src/lib/assembler.fs>.

(lcr

```
(lcr ( -- ) "paren-l-c-r"
```

A deferred word (see `defer`) whose default action is `cr`. `(lcr` is the actual carriage return done by `lcr`, before updating the data of the left-justified displaying system. (`(lcr` is a hook for the application, for special cases.

See also: `ltype`.

Source file: <src/lib/display.ltype.fs>.

(less-of

```
(less-of ( n1 n2 -- n1 n1 | n1 n3 ) "paren-less-of"
```

The run-time factor of `less-of`.

If *n1* is less than *n2*, leave *n1 n1*; otherwise leave *n1 n3*, being *n3* not equal to *n1*.

See also: `(greater-of`.

Source file: <src/lib/flow.case.fs>.

(load

```
(load ( u -- ) "paren-load"
```

Make block *u* the current input source and interpret it.

`(load` is a common factor of `load` and `continued`.

Definition:

```
: (load ( u -- ) dup lastblk ! block>source interpret ;
```

See also: `block>source`, `interpret`.

Source file: <src/kernel.z80s>.

(load-program

```
(load-program ( u -- ) "paren-load-program"
```

Load a program from block *u*, i.e. a set of blocks that are loaded as a whole. The blocks of a program don’t have block headers. Therefore programs cannot have internal requisites, i.e. they use `need` only to load from the library, which must be before the blocks of the program on the disk or disks.

Programs don’t need `-->` or any similar word to control the loading of blocks. The loading starts from block *u* and continues until the last block of the disk or until `end-program` is executed.

`(load-program` is a factor of `load-program`. `(load-program` can be used to resume `load-program` after an error, provided the code of block where the error happened (`lastblk`) is not the continuation of the previous block.

See also: `loading-program`.

Source file: <src/lib/blocks.fs>.

(located

```
(located ( ca len -- block | 0 ) "paren-located"
```

Locate the first block whose header contains the string *ca len* (surrounded by spaces), and return its number. If not found, return zero. The search is case-sensitive.

Only the blocks delimited by `first-locatable` and `last-locatable` are searched.

`(located` is a deferred word (see `defer`). Its default action is `multiline-(located`, which is under development; its alternative old action is `1-line-(located`.

`(located` is the default action of `located`, which is changed by `use-fly-index`.

See also: `default-first-locatable`.

Source file: <src/lib/002.need.fs>.

(loop

```
(loop ( R: loop-sys1 -- loop-sys2 ) "paren-loop"
```

Increment the loop index by one. If the loop index did not cross the boundary between the loop limit minus one and the loop limit, continue execution at the beginning of the loop. Otherwise, discard the loop parameters and continue execution immediately following the loop.

`(loop` is compiled by `loop`.

See also: `(+loop`.

Source file: <src/kernel.z80s>.

(make-thru-index

```
(make-thru-index ( -- ) "paren-make-thru-index"
```

Create the blocks index, from `first-locatable` to `last-locatable`.

`(make-thru-index` is a factor of `make-thru-index`.

See also: `use-thru-index`.

Source file: <src/lib/blocks.indexer.thru.fs>.

(mode-64ao-output_

```
(mode-64ao-output_  ( -- a ) "paren-mode-64-a-o-output"
```

a is the address of a Z80 routine, the low-level `mode-64ao` driver, which displays the character in the A register. The Forth IP is not preserved.

`mode-64ao-output_` is called by `mode-64ao-output_` and `mode-64ao-emit`.

Source file: <src/lib/display.mode.64ao.fs>.

(options

```
(options ( i*x x -- j*x ) "paren-options"
```

Run-time procedure compiled by `options[`.

x = option to search for

Source file: <src/lib/flow.options-bracket.fs>.

(or-of

```
(or-of ( x1 x2 x3 -- x1 x1 | x1 x4 ) "paren-or-of"
```

The run-time factor of `less-of`.

Source file: <src/lib/flow.case.fs>.

(parse-esc-string

```
(parse-esc-string ( ca len "ccc<quote>"  -- ca' len' ) "paren-parse-esc-string"
```

Parse a text string delimited by a double quote, translating some configurable characters that are escaped with a backslash. Add the translated string to *ca len*, returning a new string *ca' len'* in the `stringer`.

`(parse-esc-string` is a factor of `parse-esc-string`.

See also: `set-esc-order`.

Source file: <src/lib/strings.escaped.fs>.

(pixel-pan-right

```
(pixel-pan-right ( -- a ) "paren-pixel-pan-right"
```

Return the address *a* of a Z80 routine that pans the whole screen one pixel to the right.

WARNING	The BC register (the Forth IP) is not preserved. This is intended, in order to save time when this routine is called in a loop. Therefore the calling code must save the BC register.

See also: `pixel-pan-right`, `pixel-scroll-up`.

Source file: <src/lib/graphics.scroll.fs>.

(pixel-scroll-up

```
(pixel-scroll-up ( -- a ) "paren-pixel-scroll-up"
```

Return the address *a* of a Z80 routine that scrolls the whole screen one pixel up.

WARNING	The BC register (the Forth IP) is not preserved. This is intended, in order to save time when this routine is called in a loop. Therefore the calling code must save the BC register.

See also: `pixel-scroll-up`.

Source file: <src/lib/graphics.scroll.fs>.

(rename-file

```
(rename-file ( -- ior ) "paren-rename-file"
```

Rename the file named by the filename stored in `ufia1` to the filename stored in `ufia2`. and return I/O result code *ior*.

`(rename-file` is a factor of `rename-file`.

Source file: <src/lib/dos.gplusdos.fs>.

(resolve-ref

```
(resolve-ref ( orig b -- ) "paren-resolve-ref"
```

Resolve reference at *orig* to `assembler` label *b*.

See also: `resolve-rl#`, `resolve-al#`.

Source file: <src/lib/assembler.labels.fs>.

(rif

```
(rif ( op -- orig cs-id ) "paren-r-if"
```

Compile the Z80 `assembler` conditional relative-jump instruction *op*. Leave address *orig* to be resolved by `relse` or `rthen` and the identifier *cs-id* of the control-flow structure `rif` .. `relse` .. `rthen`.

`(rif` is a factor of `rif` and `relse`.

Source file: <src/lib/assembler.fs>.

(runtil

```
(runtil ( dest cs-id op -- ) "paren-r-until"
```

Compile a Z80 `assembler` conditional relative-jump instruction *op* to address *dest*, as part of a control-flow structure identified by *cs-id*.

`(runtil` is a factor of `runtil`, `ragain` and `rstep`.

Source file: <src/lib/assembler.fs>.

(source-id

```
(source-id ( -- a ) "paren-source-i-d"
```

A `constant`. *a* is the address of a cell containinig the value returned by `source-id`.

Source file: <src/kernel.z80s>.

(step

```
(step ( R: u -- u' ) "paren-step"
```

The run-time procedure compiled by `step`.

If the loop index is zero, discard the loop parameters and continue execution after the loop. Otherwise decrement the loop index and continue execution at the beginning of the loop.

Source file: <src/lib/flow.for.fs>.

(substitution

```
(substitution ( ca1 len1 -- ca2 ) "paren-substitution"
```

Given a string *ca1 len1* create its definition in `substitute-wordlist` its substitution and return the address of its storage space in data space, not allocated.

`(substitution` is a common factor of `substitution` and `xt-substitution`.

See also: `substitution`, `xt-substitution`, `replaces`.

Source file: <src/lib/strings.replaces.fs>.

(tape-file>

```
(tape-file> ( -- ) "paren-tape-file-from"
```

Read a tape file using the data stored at `tape-header`.

`(tape-file>` is a factor of `tape-file>`.

Source file: <src/lib/tape.fs>.

(udg-block

```
(udg-block ( width height a "name..." -- ) "paren-u-d-g-block"
```

Parse a UDG block, and store it from address *a*. *width* and *height* are in characters. The maximum *width* is 7 (imposed by the size of Forth source blocks). *height* has no maximum, as the UDG block can ocuppy more than one Forth block (provided the Forth block has no index line, i.e. `load-program` is used to load the source).

The scans can be formed by binary digits, by the characters hold in `udg-blank` and `udg-dot`, or any combination of both notations.

`(udg-block` is a common factor of `udg-block` and `,udg-block`, whose documentation include usage examples.

See also: `csprite`, `udg-group`.

Source file: <src/lib/graphics.udg.fs>.

(user

```
(user ( +n "name" -- ) "paren-user"
```

Create a user variable *name*. *+n* is the offset within the user area where the value for *name* is stored. Execution of *name* leaves its absolute user area storage address. No user space is allocated.

`(user` is a factor of `ucreate`.

See also: `user`, `2user`, `uallot`.

Source file: <src/kernel.z80s>.

(warning"

```
(warning" ( f -- ) "paren-warning-quote"
```

If *f* is not zero, display the in-line string; else do nothing.

`(warning"` is the inner procedure compiled by `warning"`.

Source file: <src/lib/exception.fs>.

(wat-xy

```
(wat-xy ( col row -- ) "paren-w-at-x-y"
```

Set the cursor coordinates to `current-window` cursor coordinates *col row*. The upper left corner of the `window` is column zero, row zero.

See also: `wat-xy`.

Source file: <src/lib/display.window.fs>.

(within-of

```
(within-of ( x1 x2 x3 -- x1 x1 | x1 x4 ) "paren-within-of"
```

The run-time factor of `within-of`. If *x1* is in range *x2 x3*, as calculated by `within`, return *x1 x1*; otherwise return *x1 x4*, being *x4* not equal to *x1*.

Source file: <src/lib/flow.case.fs>.

(~~

```
(~~ ( nt n u -- ) "paren-tilde-tilde"
```

The runtime action compiled by `~~` during the definition of word *nt* in line *n* of block *u*:

If the content of `~~?` is not zero, execute the following words in the given order: `~~before-info`,

`~~info`, `~~control` and `~~after-info`.

See also: `~~y`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

(~~info

```
(~~info ( -- ) "paren-tilde-tilde-info"
```

Default action of `~~info`: Show the debugging info compiled by `~~` and the current contents of the data stack. At least to lines are used, depending on the contents of the stack. The first line shows the block, line and definition name where `~~` was compiled; the second line shows the contents of the stack. The printing position can be configured with `~~y`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

)

)

```
) ( f -- ) "close-paren"
```

End an assertion.

`)` is an `immediate` word.

Origin: Gforth.

See also: `assert(`.

Source file: <src/lib/tool.debug.assert.fs>.

*

*

```
* ( n1|u1 n2|u2 -- n3|u3 ) "star"
```

Multiply *n1*|*u1* by *n2*|*u2* giving the product *n3*|*u3*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `m*`, `um*`, `d*`.

Source file: <src/kernel.z80s>.

*!

```
*! ( n|u a -- ) "star-store"
```

Multiply *n|u* by the single-cell number stored at *a* and store the product in *a*

See also: `2*!` `/!`, `+!`, `-!`.

Source file: <src/lib/memory.MISC.fs>.

*/

```
*/ ( n1 n2 n3 -- n4 ) "star-slash"
```

Multiply *n1* by *n2* producing the intermediate *d*. Divide *d* by *n3* giving the quotient *n4*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `*/mod`, `m*/`, `*`, `/`, `*/_`, `*/-`.

Source file: <src/lib/math.operators.1-cell.fs>.

*/-

```
*/- ( n1 n2 n3 -- n4 ) "star-slash-dash"
```

Multiply *n1* by *n2* producing the intermediate result *d*. Divide *d* by *n3* (doing a symmetric division), giving the symmetric quotient *n4*.

See also: `*/-rem`, `*/`, `*/_`, `sm/rem`.

Source file: <src/lib/math.operators.1-cell.fs>.

*/-rem

```
*/-rem ( n1 n2 n3 -- n4 n5 ) "star-slash-dash-rem"
```

Multiply *n1* by *n2* producing the intermediate result *d*. Divide *d* by *n3* (doing a symmetric division), giving the remainder *n4* and the symmetric quotient *n5*.

See also: `*/mod`, `*/_mod`, `sm/rem`.

Source file: <src/lib/math.operators.1-cell.fs>.

*/_

```
*/_ ( n1 n2 n3 -- n4 ) "star-slash-underscore"
```

Multiply *n1* by *n2* producing the intermediate result *d*. Divide *d* by *n3* (doing a floored division), giving the floored quotient *n4*.

See also: `*/_mod`, `*/`, `*/-`, `fm/mod`.

Source file: <src/lib/math.operators.1-cell.fs>.

*/_mod

```
*/_mod ( n1 n2 n3 -- n4 n5 ) "star-slash-underscore-mod"
```

Multiply *n1* by *n2* producing the intermediate result *d*. Divide *d* by *n3* (doing a floored division), giving the remainder *n4* and the floored quotient *n5*.

See also: `*/mod`, `*/_`, `*/-rem`, `fm/mod`.

Source file: <src/lib/math.operators.1-cell.fs>.

*/mod

```
*/mod ( n1 n2 n3 -- n4 n5 ) "star-slash-mod"
```

Multiply *n1* by *n2* producing the intermediate result *d*. Divide *d* by *n3* producing the remainder *n4* and the quotient *n5*.

Definition:

```
: */mod ( n1 n2 n3 -- n4 n5 ) >r m* r> m/ ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `/mod`, `*/`, `*/_mod`, `*/-rem`, `m*`, `m/`.

Source file: <src/kernel.z80s>.

+

+

```
+ ( n1|u1 n2|u2 -- n3|u3 ) "plus"
```

Add *n1|u1* to *n2|u2*, giving the sum *n3|u3*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `m+`, `d+`, `2+`, `1+`, `-`.

Source file: <src/kernel.z80s>.

+!

```
+! ( n|u a -- ) "plus-store"
```

Add *n|u* to the single-cell number at *a*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `c+!`, `@`, `+`, `!`.

Source file: <src/kernel.z80s>.

+3dos

```
+3dos ( -- ) "plus-three-dos"
```

An alias of `noop` that is defined only in the +3DOS version of Solo Forth. Its goal is to check the DOS a program is running on, using `defined` or `[defined]`.

`+3dos` is an `immediate` word.

See also: `dos`, `tr-dos`, `g+dos`.

Source file: <src/kernel.z80s>.

+ato

```
+ato ( n1 n2 "name" -- ) "plus-a-to"
```

Add *n1* to element *n2* of 1-dimension single-cell values array *name*.

`+ato` is an `immediate` word.

See also: `avalue`, `(+ato`.

Source file: <src/lib/data.array.value.fs>.

+beep>note

```
+beep>note ( +n1 -- +n2 +n3 ) "plus-beet-to-note"
```

Convert a positive pitch *+n1* of `beep` to its corresponding note *+n3* (0..11) in octave *+n2*, being zero the middle octave.

See also: `beep>note`, `-beep>note`, `/octave`, `beep>dhz`, `beep>bleep`.

Source file: <src/lib/sound.48.fs>.

+branch

```
+branch ( n -- ) "plus-branch"
```

A run-time procedure to branch conditionally. If *n* is positive, the following in-line address is copied to IP to branch forward or backward.

`+branch` is compiled by `-if` and `-until`.

See also: `branch`, `?branch`, `0branch`, `-branch`.

Source file: <src/lib/flow.branch.fs>.

+cato

```
+cato ( c n "name" -- ) "plus-c-a-to"
```

Add *c* to element *n* of 1-dimension character values array *name*.

`+cato` is an `immediate` word.

See also: `cavalue`, `(+cato`.

Source file: <src/lib/data.array.value.fs>.

+exit

```
+exit ( n -- ) ( R: nest-sys | -- nest-sys | ) "plus-exit"
```

If *n* is positive, return control to the calling definition, specified by *nest-sys*.

WARNING `+exit` is not intended to be used within a `loop`. Use `0>= if unloop exit then` instead.

`+exit` can be used in interpretation mode to stop the interpretation of a block.

See also: `exit`, `?exit`, `0exit`, `-exit`, `+if`, `+while`, `+until`.

Source file: <src/lib/flow.conditionals.positive.fs>.

+field

```
+field ( n1 n2 "name" -- n3 ) "plus-field"
```

Create a definition for *name* with the execution semantics defined below. Return *n3* = *n1* + *n2* where *n1* is the offset in the data structure before `+field` executes, and *n2* is the size of the data to be added to the data structure. *n1* and *n2* are in bytes.

name execution: `( a1 -- a2 )`

Add *n1* to *a1* giving *a2*.

In Solo Forth, `+field` is an unitialized deferred word (see `defer`), for which three implementations are provided: `+field-unopt`, `+field-opt-0` and `+field-opt-0124`.

Origin: Forth-2012 (FACILITY EXT).

See also: `begin-structure`.

Source file: <src/lib/data.begin-structure.fs>.

+field-opt-0

```
+field-opt-0 ( n1 n2 "name" -- n3 ) "plus-field-opt-zero"
```

Optimized implementation of `+field`. This implementation is more efficient than `+field-unopt` (but less than `+field-opt-0124`) because the field 0 does not calculate the field offset.

`+field-opt-0` uses 31 bytes of data space.

NOTE Loading `+field-opt-0` makes it the action of `+field`.

Source file: <src/lib/data.begin-structure.fs>.

+field-opt-0124

```
+field-opt-0124 ( n1 n2 "name" -- n3 ) "plus-field-opt-zero-one-two-four"
```

Optimized implementation of `+field` that optimizes the calculation of field offsets 0, 1, 2 and 4. Therefore it is more efficient than `+field-unopt` and `+field-opt-0`, but it uses 106 bytes of data space and needs `case`.

NOTE Loading `+field-opt-0124` makes it the action of `+field`.

Source file: <src/lib/data.begin-structure.fs>.

+field-unopt

```
+field-unopt ( n1 n2 "name" -- n3 ) "plus-field-unopt"
```

Unoptimized implementation of `+field`. This implementation is less efficient than `+field-opt-0` and `+field-opt-0124` because the field offset is calculated also when it is 0.

The advantage of this implementation is it uses only 22 bytes of data space, so it could be useful in some cases.

NOTE Loading `+field-unopt` makes it the action of `+field`.

Source file: <src/lib/data.begin-structure.fs>.

+if

```
+if "plus-if"
  Compilation: ( C: -- orig )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0>= if`.

`+if` is an `immediate` and `compile-only` word.

See also: `if`, `0if`, `-if`, `-branch` ,`+while`, `+until`, `+exit`.

Source file: <src/lib/flow.conditionals.positive.fs>.

+load

```
+load ( n -- ) "plus-load"
```

Load the block that is *n* blocks from the current one.

See also: `load`, `blk`, `+thru`.

Source file: <src/lib/blocks.fs>.

+loop

```
+loop "plus-loop"
  Compilation: ( do-sys -- )
```

Compilation: Compile `(+loop` and resolve the *do-sys* address left by `do`, `?do` or `-do`.

`+loop` is an `immediate` and `compile-only` word.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `loop`.

Source file: <src/lib/flow.do.fs>.

+order

```
+order ( wid -- ) "plus-order"
```

Remove all instances of the word list identified by *wid* from the search order, then add it to the top.

See also: `-order`, `>order`, `set-order`, `order`.

Source file: <src/lib/word_lists.fs>.

+origin

```
+origin ( n -- a ) "plus-origin"
```

Leave the memory address *a* relative by *n* bytes to the origin parameter area. `+origin` is used to access or modify the boot-up parameters at the origin area.

See the details in the source of the kernel.

Origin: fig-Forth.

Source file: <src/kernel.z80s>.

+perform

```
+perform ( a n -- ) "plus-perform"
```

Execute the execution token pointed by an offset of *n* cells from base address *a*, i.e., execute the contents of element *n* of the cell table that starts at *a*.

If the execution token is zero, do nothing.

See also: `perform`, `execute`, `array>`.

Source file: <src/lib/flow.MISC.fs>.

+place

```
+place ( ca1 len1 ca2 -- ) "plus-place"
```

Add the string *ca1 len1* to the end of the counted string *ca2*.

See also: `place`, `s+`, `smove`, `count`.

Source file: <src/lib/strings.MISC.fs>.

+seclusion

```
+seclusion ( wid1 wid2 -- wid1 wid2 ) "plus-seclusion"
```

Start more private definitions of a `seclusion` module.

See also: `-seclusion`, `end-seclusion`.

Source file: <src/lib/modules.MISC.fs>.

+stringer

```
+stringer ( -- a ) "plus-stringer"
```

A `variable`. *a* is the address of a cell containing the pointer of the `stringer`, i.e. an offset to its first free address. The offset equals the number of free characters in the `stringer`.

See also: `empty-stringer`.

Source file: <src/kernel.z80s>.

+thru

```
+thru ( u1 u2 -- ) "plus-thru"
```

Load consecutively the blocks that are *u1* blocks through *u2* blocks from the current one.

See also: `+load`, `blk`, `load`.

Source file: <src/lib/blocks.fs>.

+toarg

```
+toarg ( -- ) "plus-to-arg"
```

Set the add action for the next local variable. Used with locals created by `arguments`.

Loading `+toarg` makes `@` the default action of `arguments` locals, which is hold in `arg-default-action`.

See also: `toarg`.

Source file: <src/lib/locals.arguments.fs>.

+under

```
+under ( n1|u1 n2|u2 x -- n3|u3 x ) "plus-under"
```

Add *n2|u2* to *n1|u2*, giving the sum *n3|u3*.

`+under` is written in Z80. Its definition in Forth is the following:

```
: +under ( n1|u1 n2|u2 x -- n3|u3 x ) >r + r> ;
```

Origin: Comus.

See also: `under+`, `+`.

Source file: <src/lib/math.operators.1-cell.fs>.

+until

```
+until "plus-until"
  Compilation: ( C: dest -- )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0>= until`.

`+until` is an `immediate` and `compile-only` word.

See also: `until`, `0until`, `-until`, `-branch`, `+if`, `+while`, `+exit`.

Source file: <src/lib/flow.conditionals.positive.fs>.

+while

```
+while ( n -- ) "plus-while"
  Compilation: ( C: dest -- orig dest )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0>= while`.

`+while` is an `immediate` and `compile-only` word.

See also: `while`, `0while`, `-while`, `+if`, `+until`, `+exit`.

Source file: <src/lib/flow.conditionals.positive.fs>.

,

,

```
, ( x -- ) "comma"
```

Reserve one cell of data space and store *x* in the cell.

Definition:

```
: , ( x -- ) here ! cell allot ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `2,`, `c,`, `here`, `!`, `cell`, `allot`.

Source file: <src/kernel.z80s>.

,"

```
," ( "ccc<quote>" -- ) "comma-quote"
```

Parse "ccc" delimited by a double-quote and compile the string.

Definition:

```
: ," ( -- ) '"' parse s, ;
```

See also: `parse`, `s,`, `far,"`.

Source file: <src/kernel.z80s>.

,np

```
,np ( x -- ) "comma-n-p"
```

Store *x* into the cell address pointed by `np`, the name-space pointer, increasing it by one `cell`.

See also: `far!`.

Source file: <src/kernel.z80s>.

,udg-block

```
,udg-block ( width height "name..." -- ) "comma-u-d-g-block"
```

Parse a UDG block, and compile it in data space. *width* and *height* are in characters. The maximum *width* is 7 (imposed by the size of Forth source blocks). *height* has no maximum, as the UDG block can ocuppy more than one Forth block (provided the Forth block has no index line, i.e. `load-program` is used to load the source).

The scans can be formed by binary digits, by the characters hold in `udg-blank` and `udg-dot`, or any combination of both notations.

Usage example:

```
here 3 1 ,udg-block
..........X..X..........
...XXXXXX.X..X.XXXXXXX..
..XXXXXXXXXXXXXXXXXXXXX.
.XXXXXXXXXXXXXXXXXXXXXXX
.XX.X.X.X.X.X.X.X.X.X.XX
..XX..XX..XX..XX..XX.XX.
...X.XXX.XXX.XXX.XXX.X..
....X.X.X.X.X.X.X.X.X... constant tank

: .tank ( -- )
  tank dup emit-udga /udg+ dup emit-udga /udg+ emit-udga ;

cr .tank cr
```

See also: `udg-block`, `csprite`, `udg-group`, `emit-udga`.

Source file: <src/lib/graphics.udg.fs>.

-

-

```
- ( n1|u1 n2|u2 -- n3|u3 ) "minus"
```

Substract *n2|u2* from *n1|u1*, giving the difference *n3|u3*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `d-`, `2-`, `1-`, `+`.

Source file: <src/kernel.z80s>.

-!

```
-! ( n|u a -- ) "minus-store"
```

Subtract $n|u$ from the single-cell number stored at a.

See also: `+!`, `1-!`, `c-!`.

Source file: <src/lib/memory.MISC.fs>.

-->

```
--> ( -- ) "next-block"
```

Continue interpretation with the next block.

`-->` is an `immediate` word.

Definition:

```
: --> ( -- )
  ?loading refill 0= #-35 ?throw ; immediate
```

Origin: fig-Forth, Forth-79 (Reference Word Set), Forth-83 (Controlled Reference Words).

See also: `?-->`, `load`, `continued`, `?loading`, `refill`.

Source file: <src/kernel.z80s>.

-1

```
-1 ( -- -1 ) "minus-one"
```

Return *-1*. `-1` is not a `constant`, but a `code` word, which is faster.

See also: `0`, `1`, `2`, `true`.

Source file: <src/kernel.z80s>.

-1..1

```
-1..1 ( -- -1|0|1 ) "minus-one-dot-dot-one"
```

Return a random number: -1, 0 or 1.

See also: `-1|1`, `rnd`, `fast-random`.

Source file: <src/lib/random.fs>.

-1|1

```
-1|1 ( -- -1|1 ) "minus-one-bar-one"
```

Return a random number: -1 or 1.

See also: `-1..1`, `rnd`, `fast-random`.

Source file: <src/lib/random.fs>.

->

```
-> ( i*x -- )
```

Part of the `hayes-test`: Record depth and content of stack.

See also: `{`, `}`.

Source file: <src/lib/meta.tester.hayes.fs>.

->

```
-> ( i*x -- )
```

Part of `ttester`: Record depth and contents of stack.

See also: `t{`, `}t`.

Source file: <src/lib/meta.tester.ttester.fs>.

->in/l

```
->in/l ( -- n ) "minus-to-in-slash-l"
```

Return number *n* of characters not interpreted yet in the current line of the block being interpreted. No check is done whether any block is actually being interpreted.

`->in/l` is a factor of `\`.

Definition:

```
: ->in/l ( -- n ) c/l >in/l - ;
```

See also: `blk-line`, `>in/l`, `>in`, `c/l`.

Source file: <src/kernel.z80s>.

-beep>note

```
-beep>note ( -n1 -- -n2 +n3 ) "minus-beep-to-note"
```

Convert a negative pitch *-n1* of `beep` to its corresponding note *+n3* (0..11) in octave *-n2*, being zero the middle octave.

See also: `beep>note`, `+beep>note`, `/octave`, `beep>dhz`, `beep>dhz`.

Source file: <src/lib/sound.48.fs>.

-block-drives

```
-block-drives ( -- ) "minus-block-drives"
```

Fill `block-drives` with `not-block-drive`, making no disk drive be used as block drive.

See also: `set-block-drives`, `get-block-drives`.

Source file: <src/lib/dos.COMMON.fs>.

-branch

```
-branch ( n -- ) "minus-branch"
```

A run-time procedure to branch conditionally. If *n* is negative, the following in-line address is copied to IP to branch forward or backward.

`-branch` is compiled by `+if` and `+until`.

See also: `branch`, `?branch`, `0branch`, `+branch`.

Source file: <src/lib/flow.branch.fs>.

-do

```
-do
  Compilation: ( -- do-sys )
"minus-do"
```

Compile `(-do` and leave *do-sys* to be consumed by `loop` or `+loop`. `-do` is an alternative to `do` and `?do`, to create count-down loops with `+loop`.

`-do` is an `immediate` and `compile-only` word.

Usage example:

```
: -count-down ( limit start -- )
  -do i . -1 +loop ;

0 0 -count-down \ prints nothing
4 0 -count-down \ prints nothing
0 4 -count-down \ prints 4 3 2 1

\ Compare to:

: ?count-down ( limit start -- )
  ?do i . -1 +loop ;

0 0 ?count-down \ prints nothing
4 0 ?count-down \ prints 0 -1..-32768 32767..4
0 4 ?count-down \ prints 4 3 2 1 0

: count-down ( limit start -- )
  do i . -1 +loop ;

0 0 count-down \ prints 0
4 0 count-down \ prints 0 -1..-32768 32767..4
0 4 count-down \ prints 4 3 2 1 0
```

Origin: Gforth.

Source file: <src/lib/flow.do.fs>.

-dup

```
-dup ( x -- x x | x ) "minus-dup"
```

Duplicate *x* if it’s negative.

See also: `dup`, `0dup`.

Source file: <src/lib/data_stack.fs>.

-exit

```
-exit ( n -- ) ( R: nest-sys | -- nest-sys | ) "minus-exit"
```

If *n* is negative, return control to the calling definition, specified by *nest-sys*.

WARNING	`-exit` is not intended to be used within a `loop`. Use `0< if unloop exit then` instead.

`-exit` can be used in interpretation mode to stop the interpretation of a block.

See also: `exit`, `?exit`, `0exit`, `+exit`, `-if`, `-while`, `-until`.

Source file: <src/lib/flow.conditionals.negative.fs>.

-filename

```
-filename ( -- ) "minus-filename"
```

Blank the filename in `ufia`, i.e. replace it with spaces.

See also: `/filename`, `set-filename`.

Source file: <src/lib/dos.gplusdos.fs>.

-if

```
-if "minus-if"
  Compilation: ( C: -- orig )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0< if`.

`-if` is an `immediate` and `compile-only` word.

See also: `if`, `0if`, `+if`, `+branch`, `-while`, `-until`, `-exit`.

Source file: <src/lib/flow.conditionals.negative.fs>.

-jiffy

```
-jiffy ( -- ) "minus-jiffy"
```

Deactivate the so called "jiffy call", the Z80 routine that is called by G+DOS after the OS interrupts routine (every 50th of a second), by setting its default value (a noop routine in the RAM of the Plus D interface).

See also: `jiffy!`, `jiffy@`.

Source file: <src/lib/multitask.gplusdos.fs>.

-keys

```
-keys ( -- ) "minus-keys"
```

Remove all keys from the keyboard buffer.

See also: `key?`, `new-key`, `new-key-`, `key`, `xkey`.

Source file: <src/lib/keyboard.MISC.fs>.

-leading

```
-leading ( ca1 len1 -- ca2 len2 ) "minus-leading"
```

Adjust the start and length of a string *ca1 len1* to suppress the leading blanks, returning the result *ca2 len2*.

Definition:

```
: -leading ( ca len -- ca' len' ) bl skip ;
```

See also: `-trailing`, `trim`, `bl`, `skip`.

Source file: <src/kernel.z80s>.

-mixer

```
-mixer ( -- ) "minus-mixer"
```

Disable the noise and tone mixers for the three channels of the AY-3-8912 sound generator.

See also: `set-mixer`, `get-mixer`, `silence`.

Source file: <src/lib/sound.128.fs>.

-move

```
-move ( ca n -- ) "minus-move"
```

Part of `specforth-editor`: Move a line of text from *ca* to line *n* of current block.

See also: `m`, `c/l`, `cmove`, `update`.

Source file: <src/lib/prog.editor.specforth.fs>.

-order

```
-order ( wid -- ) "minus-order"
```

Remove all instances of word list identified by *wid* from the search order.

See also: `+order`, `>order`, `set-order`, `order`.

Source file: <src/lib/word_lists.fs>.

-prefix

```
-prefix ( ca1 len1 ca2 len2 -- ca1 len1 | ca3 len3 ) "minus-prefix"
```

Remove prefix *ca2 len2* from string *ca1 len1*.

See also: `-suffix`, `/string`, `1/string`, `-leading`.

Source file: <src/lib/strings.MISC.fs>.

-rem

```
-rem ( n1 n2 -- n3 ) "dash-rem"
```

Divide *n1* by *n2* (doing a symmetric division), giving the remainder *n3*.

See also: `/-rem`, `/`, `/_mod`.

Source file: <src/lib/math.operators.1-cell.fs>.

-rot

```
-rot ( x1 x2 x3 -- x3 x1 x2 ) "minus-rot"
```

Rotate the top three stack entries in reverse order.

See also: `rot`, `over`, `tuck`, `swap`, `roll`, `pick`, `unpick`.

Source file: <src/kernel.z80s>.

-seclusion

```
-seclusion ( wid1 wid2 -- wid1 wid2 ) "minus-seclusion"
```

Start the public definitions of a `seclusion` module.

See also: `+seclusion`, `end-seclusion`.

Source file: <src/lib/modules.MISC.fs>.

-suffix

```
-suffix ( ca1 len1 ca2 len2 -- ca1 len1 | ca3 len3 ) "minus-suffix"
```

Remove suffix *ca2 len2* from string *ca1 len1*.

See also: `-prefix`, `string/`, `chop`, `-trailing`.

Source file: <src/lib/strings.MISC.fs>.

-tape-filename

```
-tape-filename ( -- ) "minus-tape-filename"
```

Blank `tape-filename` in `tape-header`.

Source file: <src/lib/tape.fs>.

-text

```
-text ( ca1 len1 ca2 -- f ) "minus-text"
```

Part of `specforth-editor`: Return a non-zero *f* if string *ca1 len1* exactly match string *ca2 len1*, else return a false flag.

See also: `match`.

Source file: <src/lib/prog.editor.specforth.fs>.

-trailing

```
-trailing ( ca1 len1 -- ca2 len2 ) "minus-trailing"
```

Adjust the length of a string *ca1 len1* to suppress the trailing blanks, returning the result *ca2 len2*.

If *len* is greater than zero, *len2* is equal to *len1* less the number of spaces at the end of the character string specified by *ca1 len1*. If *len1* is zero or the entire string consists of spaces, *len2* is zero.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (STRING), Forth-2012 (STRING).

See also: `-leading`, `trim`.

Source file: <src/kernel.z80s>.

-ufia

```
-ufia ( -- )
```

Erase the contents of `ufia` with zeroes,

See also: `init-ufia`, `/ufia`.

Source file: <src/lib/dos.gplusdos.fs>.

-until

```
-until "minus-until"
  Compilation: ( C: dest -- )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0< until`.

`-until` is an `immediate` and `compile-only` word.

See also: `until`, `0until`, `+until`, `+branch`, `-if`, `-while`, `-exit`.

Source file: <src/lib/flow.conditionals.negative.fs>.

-while

```
-while "minus-while"
  Compilation: ( C: dest -- orig dest )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0< while`.

`-while` is an `immediate` and `compile-only` word.

See also: `while`, `0while`, `+while`, `-if`, `-until`, `-exit`.

Source file: <src/lib/flow.conditionals.negative.fs>.

.

.

```
. ( n -- ) "dot"
```

Display signed integer *n* according to current `base`, followed by one blank.

See also: `?`, `u.`, `d.`, `f.`.

Source file: <src/kernel.z80s>.

."

```
." "dot-quote"
  Compilation: ( "ccc<quote>" -- )
  Run-time: ( -- )
```

Parse "ccc" delimited by a double-quote and compile the corresponding string and the execution procedure `(."`, which will display it at run-time.

`."` is an `immediate` and `compile-only` word.

Definition:

```
: ." ( "ccc<quote> -- ) compile (." ," ; immediate
```

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `s"`, `.(`, `,"`.

Source file: <src/kernel.z80s>.

.(

```
.( ( "ccc<paren>" -- ) "dot-paren"
```

Parse and display *ccc* delimited by a right parenthesis.

`.(` is an `immediate` word.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `."`, `(`.

Source file: <src/kernel.z80s>.

.00

```
.00 ( +n -- ) "dot-zero-zero"
```

Display +*n* with two digits.

See also: `.0000`, `.time`, `.date`.

Source file: <src/lib/display.numbers.fs>.

.0000

```
.0000 ( +n -- ) "dot-zero-zero-zero-zero"
```

Display +*n* with four digits.

See also: `.00`, `.date`.

Source file: <src/lib/display.numbers.fs>.

.\"

```
.\"
  Compilation: ( "ccc<quote>" -- )
  Run-time:    (-- ca len )
"dot-backslash-quote"
```

`.\"` is an `immediate` and `compile-only` word.

NOTE When `.\"` is loaded, `esc-standard-chars-wordlist` is set as the only word list by `set-esc-order`. That is the standard behaviour. Alternative escaped chars can be configured with `esc-block-chars-wordlist` and `esc-udg-chars-wordlist`.

See also: `parse-esc-string`, `set-esc-order`, `s\"`.

Source file: <src/lib/strings.escaped.fs>.

.context

```
.context ( -- ) "dot-context"
```

Display the word lists in the search order in their search order sequence, from first searched to last searched.

See also: `get-order`, `.wordlist`, `order`.

Source file: <src/lib/tool.list.word_lists.fs>.

.current

```
.current ( -- ) "dot-current"
```

Display the compilation word list.

See also: `get-current`, `.wordlist`, `order`.

Source file: <src/lib/tool.list.word_lists.fs>.

.date

```
.date ( day month year -- ) "dot-date"
```

Display the given time in ISO 8601 extended format.

See also: `.time`, `.time&date`, `time&date`, `.0000`, `.00`.

Source file: <src/lib/time.fs>.

.depth

```
.depth ( n -- )
```

Display *n* with the format used by `.s` and `u.s` to display the `depth` of the data stack`.

See also: `.r`, `depth`.

Source file: <src/lib/tool.list.stack.fs>.

.error-word

```
.error-word ( -- ) "dot-error-word"
```

Display the string identified by the cell pair stored in `parsed-name`, followed by a question mark.

Definition:

```
: .error-word ( -- ) parsed-name 2@ cr type ."  ? " ;
```

See also: `error`, `.throw`.

Source file: <src/kernel.z80s>.

.fid

```
.fid ( fid -- ) "dot-f-i-d"
```

Display the contents of the data structure pointed by file identifier *fid*.

See also: `.ufia`, `fid`.

Source file: <src/lib/dos.gplusdos.fs>.

.fs

```
.fs ( F: i*r -- i*r )
```

See also: `dump-fs`, `f.`.

Source file: <src/lib/math.floating_point.rom.fs>.

.gil-heap

```
.gil-heap ( -- ) "dot-gil-heap"
```

Print the map of the current memory `heap`, in the implementation based on code written by Javier Gil, whose words are defined in `gil-heap-wordlist`.

Occupied chunks are marked with a 'x'; free chunks are marked with a '-'.

Source file: <src/lib/memory.allocate.gil.fs>.

.index

```
.index ( u -- ) "dot-index"
```

Display the first line of the block *u*, which conventionally contains a comment with a title.

Source file: <src/lib/tool.list.blocks.fs>.

.l

```
.l ( -- ) "dot-l"
```

Dump the contents of the tables pointed by `labels` and `l-refs`.

`.l` is a debugging tool for `assembler` labels defined by `l:`.

Source file: <src/lib/assembler.labels.fs>.

.line

```
.line ( n1 n2 -- ) "dot-line"
```

Display line *n1* from block *n2*, without trailing spaces.

Origin: fig-Forth.

See also: `.line#`, `blk-line`.

Source file: <src/lib/tool.list.blocks.fs>.

.line#

```
.line# ( n -- ) "dot-line-number-sign"
```

Display line number *n* right-aligned in a field whose width depends on the current radix (decimal, hex or binary).

See also: `/line#`.

Source file: <src/lib/tool.list.blocks.fs>.

.menu

```
.menu  ( -- ) "dot-menu"
```

Display the current menu, which has been set by `set-menu` and can be activated by `menu`.

See also: `new-menu`, `.menu-banner`, `.menu-options`, `.menu-border`.

Source file: <src/lib/menu.sinclair.fs>.

.menu-banner

```
.menu-banner ( -- ) "dot-menu-banner"
```

Display the banner of the current `menu`.

See also: `menu-banner-attr`, `menu-title`, `menu-width`, `.sinclair-stripes`, `.menu`, `.menu-options`, `.menu-border`, `type-left-field`, `menu-xy`.

Source file: <src/lib/menu.sinclair.fs>.

.menu-border

```
.menu-border ( -- ) "dot-menu-border"
```

Draw a 1-pixel border around the current `menu` options, preserving the attributes.

See also: `.menu`, `.menu-options`, `.menu-banner`, `ortholine`, `menu-xy`, `xy>gxy`.

Source file: <src/lib/menu.sinclair.fs>.

.menu-option

```
.menu-option ( n -- ) "dot-menu-option"
```

Display menu option *n* of the current `menu`.

See also: `.menu-options`, `.menu`.

Source file: <src/lib/menu.sinclair.fs>.

.menu-options

```
.menu-options ( -- ) "dot-menu-options"
```

Display the options of the current `menu`.

See also: `.menu`, `.menu-option`, `.menu-border`, `.menu-banner`.

Source file: <src/lib/menu.sinclair.fs>.

.name

```
.name ( nt -- ) "dot-name"
```

Display the name of the word identified by *nt*.

NOTE `.name` is called `.id` or `id.` in other Forth systems.

See also: `name>string`, `type`, `space`.

Source file: <src/lib/compilation.fs>.

.ok

```
.ok ( -- ) "dot-ok"
```

Display "ok". `.ok` is the default action of `ok`.

Source file: <src/kernel.z80s>.

.os-chans

```
.os-chans ( -- ) "dot-o-s-chans"
```

Display the contents of `os-chans`.

See also: `.os-strms`.

Source file: <src/lib/os.fs>.

.os-strms

```
.os-strms ( -- ) "dot-o-s-streams"
```

Display the contents of `os-strms`.

See also: `.os-chans`, `first-stream`, `last-stream`, `stream?`.

Source file: <src/lib/os.fs>.

.r

```
.r ( n1 n2 -- ) "dot-r"
```

Display *n1* right aligned in a field *n2* characters wide. If the number of characters required to display *n1* is greater than *n2*, all digits are displayed with no leading spaces in a field as wide as necessary.

Definition:

```
: .r ( n1 n2 -- ) >r s>d r> d.r ;
```

Origin: Forth-79 (Reference Word Set)[3], Forth-83 (Controlled Reference Word)[4], Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `u.r`, `d.r`, `0.r`, `s>d`.

Source file: <src/kernel.z80s>.

.s

```
.s ( -- )
```

Display, using `.`, the values currently on the data stack.

See also: `u.s`, `depth`, `.depth`.

Source file: <src/lib/tool.list.stack.fs>.

.sinclair-stripes

```
.sinclair-stripes ( -- ) "dot-sinclair-stripes"
```

Display the Sinclair stripes by using `sinclair-stripes` as UDG font and typing `sinclair-stripes$`. The current UDG font is preserved.

See also: `set-udg`, `get-udg`.

Source file: <src/lib/menu.sinclair.fs>.

.throw

```
.throw ( n -- ) "dot-throw"
```

Display a message giving information about the condition associated with the `throw` code *n*.

`.throw` is executed by `error`. It’s a deferred word (see `defer`) whose default action is `.throw#`, which displays only the number. An alternative action is `.throw-message`, which displays also the description.

Source file: <src/kernel.z80s>.

.throw#

```
.throw# ( n -- ) "dot-throw-number-sign"
```

Display the number of `throw` code *n*, as a decimal number, prefixed with a '#' and followed by a space.

`.throw#` is the default action of `.throw`. Its alternative action `.throw-message` displays also the error description.

Source file: <src/kernel.z80s>.

.throw-message

```
.throw-message ( n -- ) "dot-throw-message"
```

Alternative action of the deferred word `.throw` (see `defer`): Display the description of the `throw` exception code *n*. The variable `errors-block` contains the number of the first block where messages are hold. If `errors-block` contains zero, only the error number is displayed.

For convenience, loading `.throw-message` makes it the action of `.throw`.

NOTE The error descriptions are not stored in memory, but read from the library every time. Therefore the library must be accessible.

See also: `.throw#`, `error>line`.

Source file: <src/lib/exception.fs>.

.time

```
.time ( second minute hour -- ) "dot-time"
```

Display the given time in ISO 8601 extended format.

See also: `.date`, `.time&date`, `time&date`, `.00`.

Source file: <src/lib/time.fs>.

.time&date

```
.time&date ( second minute hour day month year -- ) "dot-time-and-date"
```

Display the given time and date in ISO 8601 extended format.

See also: `.date`, `.time`, `time&date`.

Source file: <src/lib/time.fs>.

.ufia

```
.ufia ( fid -- ) "dot-u-f-i-a"
```

Display the contents of the UFIA data structure pointed by *a*.

See also: `.fid`, `ufia`.

Source file: <src/lib/dos.gplusdos.fs>.

.unused

```
.unused ( -- ) "dot-unused"
```

Display the total RAM in the system, and the amount of space remaining in the regions addressed by `here` and `np`, in bytes.

See also: `unused`, `farunused`, `.words`.

Source file: <src/lib/tool.debug.MISC.fs>.

.version

```
.version ( -- ) "dot-version"
```

Display the Solo Forth `version`.

Source file: <src/kernel.z80s>.

.word

```
.word ( nt -- ) "dot-word"
```

A deferred word (see `defer`) whose default action is `(.word`. This word is used by `words`, `words-like` and `wordlist-words`, therefore their output can be changed by the user in special cases, for example when more details are needed for debugging.

Source file: <src/lib/tool.list.words.fs>.

.wordlist

```
.wordlist ( wid -- ) "dot-wordlist"
```

If the `wordlist` identified by *wid* has an associated name, display it; else display *wid*.

See also: `wordlists`, `dump-wordlist`, `wordlist>name`.

Source file: <src/lib/tool.list.word_lists.fs>.

.wordname

```
.wordname ( nt -- ) "dot-wordname"
```

An alternative action for the deferred word `.word` (see `defer`), which is used by `words`, `words-like` and `wordlist-words`. `.wordname` prints *nt* and its correspondent name.

Source file: <src/lib/tool.list.words.fs>.

.words

```
.words ( -- ) "dot-words"
```

Display a message informing about the number of words defined in the system.

See also: `#words`, `greeting`, `.unused`.

Source file: <src/lib/tool.debug.MISC.fs>.

.xs

```
.xs ( -- ) "dot-x-s"
```

Display the number of items on the current `xstack`, followed by a list of the items, if any; TOS is the right-most item.

See also: `xdepth` ,`(.xs`.

Source file: <src/lib/data.xstack.fs>.

/

/

```
/ ( n1 n2 -- n3 ) "slash"
```

Divide *n1* by *n2*, giving the quotient *n3*.

Definition:

```
: / ( n1 n2 -- n3 ) /mod nip ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `m/`, `/mod`, `/_`, `/-`, `gcd`.

Source file: <src/kernel.z80s>.

/!

```
/! ( n a -- ) "slash-store"
```

Divide *n* by the single-cell number stored at *a* and store the quotient in *a*

See also: `2/!`, `*!`, `+!`, `-!`.

Source file: <src/lib/memory.MISC.fs>.

/-

```
/- ( n1 n2 -- n3 ) "slash-dash"
```

Divide *n1* by *n2* (doing a symmetric division), giving the symmetric quotient *n4*.

See also: `/-rem`, `/`, `/_`, `sm/rem`.

Source file: <src/lib/math.operators.1-cell.fs>.

/-rem

```
/-rem ( n1 n2 -- n3 n4 ) "slash-dash-rem"
```

Divide *n1* by *n2* (doing a symmetric division), giving the remainder *n3* and the symmetric quotient *n4*.

See also: `/mod`, `/_mod`, `sm/rem`.

Source file: <src/lib/math.operators.1-cell.fs>.

/_

```
/_ ( n1 n2 -- n3 ) "slash-underscore"
```

Divide *n1* by *n2* (doing a floored division), giving the floored quotient *n4*.

See also: `/_mod`, `/`, `/-`, `fm/mod`.

Source file: <src/lib/math.operators.1-cell.fs>.

/_mod

```
/_mod ( n1 n2 -- n3 n4 ) "slash-underscore-mode"
```

Divide *n1* by *n2* (doing a floored division), giving the remainder *n3* and the floored quotient *n4*.

See also: `/mod`, `/-rem`, `fm/mod`.

Source file: <src/lib/math.operators.1-cell.fs>.

/bank

```
/bank ( -- n ) "slash-bank"
```

n is the size in bytes of a memory bank: $4000.

See also: `bank-start`.

Source file: <src/lib/memory.far.fs>.

/counted-string

```
/counted-string ( -- n ) "slash-counted-string"
```

n is the maximum size of a counted string, in characters.

See also: `max-char`, `environment?`.

Source file: <src/lib/environment-question.fs>.

/fid

```
/fid ( -- n ) "slash-f-i-d"
```

n is the length of a data structure pointed by an address that is used as file identifier describing a file that is open. The structure is identical to `ufia`, except field `~fid-link` is added at the end.

See also: `/ufia`.

Source file: <src/lib/dos.gplusdos.fs>.

/filename

```
/filename ( -- b ) "slash-filename"
```

A `cconstant` that returns the maximum length of a G+DOS filename.

See also: `set-filename`.

Source file: <src/lib/dos.gplusdos.fs>.

/first-name

```
/first-name ( ca1 len1 -- ca2 len2 ca3 len3 ) "slash-first-name"
```

Get the first name *ca3 len3* from string *ca2 len2*, returning also the remaining string *ca3 len3*.

See also: `first-name`, `/name`.

Source file: <src/lib/strings.MISC.fs>.

/heap

```
/heap  ( -- n ) "slash-heap"
```

Size of the current `heap`, in bytes.

See also: `get-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

/hold

```
/hold ( -- len ) "slash-hold"
```

A 'cconstant`. *len* is the length of the pictured output string buffer, which is located right below `pad`.

The default value of `/hold` is 80. It may be changed by `c!>`.

See also: `hld`, `<#`, `/pad`.

Source file: <src/kernel.z80s>.

/kk

```
/kk ( -- n ) "slash-k-k"
```

n is the number of bytes ocuppied by every key stored in `kk-ports`: 3 (smaller and slower table) or 4 (bigger and faster table).

There are two versions of `kk,` and `kk@`. They depend on the value of `/kk`.

The application can define `/kk` before needing `kk-ports`; otherwise it will be defined as a `cconstant` with value 4.

Source file: <src/lib/keyboard.MISC.fs>.

/l-ref

```
/l-ref ( -- n ) "slash-l-ref"
```

n is the size in bytes of each `assembler` label reference stored in the `l-refs` table.

See also: `/l-refs`.

Source file: <src/lib/assembler.labels.fs>.

/l-refs

```
/l-refs ( -- n ) "slash-l-refs"
```

n is the size in bytes of the `l-refs` table.

See also: `max-l-refs`, `/l-ref`, `/labels`.

Source file: <src/lib/assembler.labels.fs>.

/labels

```
/labels ( -- n ) "slash-labels"
```

n is the size in bytes of the `labels` table.

See also: `max-labels`, `/l-refs`.

Source file: <src/lib/assembler.labels.fs>.

/line#

```
/line# ( -- n ) "slash-line-number-sign"
```

Maximum length of a line number in the current radix. It works for decimal, hex and binary.

See also: `.line#`.

Source file: <src/lib/tool.list.blocks.fs>.

/mod

```
/mod ( n1 n2 -- n3 n4 ) "slash-mod"
```

Divide *n1* by *n2*, giving the remainder *n3* and the quotient *n4*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `m/`, `du/mod`, `/`, `mod`, `/-rem`, `/_mod`.

Source file: <src/kernel.z80s>.

/name

```
/name ( ca1 len1 -- ca2 len2 ca3 len3 ) "slash-name"
```

Split string *ca1 len1* into *ca2 len2* (from the start of the first name in *ca1 len1*) and *ca3 len3* (from the char after the first name in _ca1 len1). A name is a substring separated by spaces.

See also: `first-name`, `/string`, `-prefix`, `-suffix`, `string/`.

Source file: <src/lib/strings.MISC.fs>.

/octave

```
/octave ( -- c ) "slash-octave"
```

A `cconstant` that returns the number of notes in one octave: 12.

See also: `middle-octave`.

Source file: <src/lib/sound.48.fs>.

/pad

```
/pad ( -- n ) "slash-pad"
```

n is the size of the scratch area pointed to by `pad`, in characters.

See also: `/hold`, `environment?`.

Source file: <src/lib/environment-question.fs>.

/qx

```
/qx ( -- n ) "slash-q-x"
```

n is the number of header lines shown on a quick index. It depends on the rows and columns of the current screen mode.

See also: `qx`.

Source file: <src/lib/tool.list.blocks.fs>.

/qx-column

```
/qx-column ( -- n ) "slash-q-x-column"
```

n is the width of a column of the quick index. It depends on the columns (32, 42, 64…) of the current screen mode.

See also: `qx`, `qx-columns`.

Source file: <src/lib/tool.list.blocks.fs>.

/sinclair-stripes

```
/sinclair-stripes ( -- len )
```

A `cconstant`. *len* is the size of `sinclair-stripes$` in graphic characters, i.e. the visible length of the string when displayed.

`/sinclair-stripes` is used by `set-menu` and other menu words.

Source file: <src/lib/menu.sinclair.fs>.

/sound

```
/sound ( -- b ) "slash-sound"
```

A character constant that returns 14, the number of sound registers used by ZX Spectrum 128.

See also: `!sound`, `@sound`, `sound`, `play`.

Source file: <src/lib/sound.128.fs>.

/string

```
/string ( ca1 len1 n -- ca2 len2 ) "slash-string"
```

Adjust the character string *ca1 len1* by *n* characters. The resulting character string *ca2 len2* begins at *ca1* plus *n* characters and is *len1* minus *n* characters long.

`/string` is written in Z80. Equivalent definitions in Forth are the following:

```
: /string ( ca1 len1 n -- ca2 len2 ) rot over + -rot - ;
: /string ( ca1 len1 n -- ca2 len2 ) dup >r - swap r> + swap ;
```

Origin: Forth-94 (STRING), Forth-2012 (STRING).

See also: `1/string`, `-prefix`, `string/`.

Source file: <src/kernel.z80s>.

/stringer

```
/stringer ( -- len ) "slash-stringer"
```

A `constant`. *len* is the maximum size of the `stringer`, in characters. See how to configure it in the documentation of `stringer`.

See also: `+stringer`, `empty-stringer`, `default-stringer'.

Source file: <src/kernel.z80s>.

/tabulate

```
/tabulate ( -- ca ) "slash-tabulate"
```

ca is the address of a byte containing the number of spaces that `tabulate` counts for. Its default value is 8.

See `tabulate`.

Source file: <src/lib/display.control.fs>.

/tape-filename

```
/tape-filename ( -- n ) "slash-tape-filename"
```

n is the maximum length of a tape filename, which is 10 characters.

See also: `tape-filename`. `/filename`.

Source file: <src/lib/tape.fs>.

/tape-header

```
/tape-header ( -- n ) "slash-tape-header"
```

n is the length of a `tape-header`: 17 bytes.

Source file: <src/lib/tape.fs>.

/tib

```
/tib ( -- b ) "slash-t-i-b"
```

A `cconstant`. *b* is the maximum size of `tib`, the terminal input buffer,

See also: `#tib`.

Source file: <src/kernel.z80s>.

/udg

```
/udg ( -- b ) "slash-u-d-g"
```

b is the size of a UDG (User Defined Graphic), in bytes.

See also: `udg-width`, `udg!`, `/udg*`, `/udg+`.

Source file: <src/lib/graphics.udg.fs>.

/udg*

```
/udg* ( n1 -- n2 ) "slash-u-d-g-star"
```

Multiply *n1* by `/udg`, resulting *n2*. Used by `udg>`.

`/udg*` is equivalent to `/udg *` but faster: it's an `alias` of `8*`.

See also: `/udg+`.

Source file: <src/lib/graphics.udg.fs>.

/udg+

```
/udg+ ( n1 -- n2 ) "slash-u-d-g-plus"
```

Add `/udg` to *n1*, resulting *n2*.

`/udg+` is useful when UDG are referenced by address, e.g. with `emit-udga` and `,udg-block`.

`/udg+` is equivalent to `/udg +` but faster: it's an `alias` of `8+`.

See also: `/udg*`.

Source file: <src/lib/graphics.udg.fs>.

/ufia

```
/ufia ( -- n ) "slash-u-f-i-a"
```

n is the length of a UFIA (User File Information Area), a 24-byte structure which describes a file.

See also: `ufia`, `ufia0`, `ufia1`, `ufia2`, `/fid`.

Source file: <src/lib/dos.gplusdos.fs>.

/user

```
/user ( -- n ) "slash-user"
```

A `constant`. *n* is the length of the user area.

See also: `up`.

Source file: <src/kernel.z80s>.

/window

```
/window ( -- n ) "slash-window"
```

A `cconstant`. *n* is the size in bytes of a `window` data structure.

See also: `current-window`.

Source file: <src/lib/display.window.fs>.

/wordlist

```
/wordlist ( -- n )
```

A `cconstant`. *n* is the length in bytes of a `wordlist` data structure, created by `wordlist,`.

Source file: <src/lib/word_lists.fs>.

/wtype

```
/wtype ( ca len len1 n -- ca' len' ) "slash-w-type"
```

Display the first *len1* characters of string *ca len* in the `current-window`, then remove the first *n* characters from the string, returning the result string *ca' len'*.

`/wtype` is a factor of `wltype`.

See also: `free/wtype`.

Source file: <src/lib/display.window.fs>.

0

0

```
0 ( -- 0 )
```

Return *0*. `0` is not a `constant`, but a `code` word, which is faster.

See also: `-1`, `1`, `2`, `false`.

Source file: <src/kernel.z80s>.

0.r

```
0.r ( n -- ) "zero-dot-r"
```

Display *n* according to current base, with no leading or trailing spaces. `0.r` is a faster alternative to the idiom `0 .r`.

`0.r` is written in Z80. Its equivalent definition in Forth is the following:

```
: 0.r ( n -- ) 0 .r ;
```

See also: `.r`, `0d.r`.

Source file: <src/kernel.z80s>.

0<

```
0< ( x -- f ) "0-less"
```

f is true if and only if *n* is less than zero.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `0>`, `0<=`, `0=`, `0<>`.

Source file: <src/kernel.z80s>.

0<=

```
0<= ( n -- f ) "zero-less-or-equal"
```

f is `true` if and only if *n* is less than or equal to zero.

See also: `0>=`, `<=`, `u<=`.

Source file: <src/lib/math.operators.1-cell.fs>.

0<>

```
0<> ( x -- f ) "zero-not-equals"
```

f is true if and only if *x* is not equal to zero.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `0=`.

Source file: <src/kernel.z80s>.

0=

```
0= ( x -- f ) "zero-equals"
```

f is true if and only if *x* is equal to zero.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `0<>`, `0<`, `0>`, `negate`, `invert`.

Source file: <src/kernel.z80s>.

0>

```
0> ( n -- f ) "zero-greater"
```

f is true if and only if *n* is greater than zero.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `0<`, `0>=`, `0=`, `0<>`.

Source file: <src/kernel.z80s>.

0>=

```
0>= ( n -- f ) "zero-greater-or_equal"
```

f is `true` if and only if *n* is greater than or equal to zero.

See also: `0<=`, `>=`, `u>=`.

Source file: <src/lib/math.operators.1-cell.fs>.

0branch

```
0branch ( f -- ) "zero-branch"
```

A run-time procedure to branch conditionally. If *f* is false (zero), the following in-line address is copied to IP to branch forward or backward.

Origin: fig-Forth.

See also: `branch`, `?branch`, `-branch`, `+branch`.

Source file: <src/kernel.z80s>.

0d.r

```
0d.r ( d -- ) "zero-d-dot-r"
```

Display *d* according to current base, with no leading or trailing spaces. `d0.r` is a faster alternative to the idiom `0 d.r`.

`0d.r` is written in Z80. Its equivalent definition in Forth is the following:

```
: 0d.r ( d -- ) 0 d.r ;
```

See also: `d.r`, `0.r`.

Source file: <src/kernel.z80s>.

0dup

```
0dup ( x -- x | 0 0 ) "zero-dup"
```

Duplicate *x* if it's zero.

See also: `dup`, `-dup`.

Source file: <src/lib/data_stack.fs>.

0exit

```
0exit ( f -- ) ( R: nest-sys | -- nest-sys | ) "zero-exit"
```

If *f* is zero, return control to the calling definition, specified by *nest-sys*.

WARNING	`0exit` is not intended to be used within a `loop`. Use `0= if unloop exit then` instead.

`0exit` can be used in interpretation mode to stop the interpretation of a block.

See also: `?exit`, `exit`, `-exit` ,`+exit`, `0if`, `0while`, `0until`, `unloop`.

Source file: <src/kernel.z80s>.

0if

```
0if "zero-if"
  Compilation: ( C: -- orig )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0= if`.

`0if` is an `immediate` and `compile-only` word.

See also: `if`, `-if`, `+if`, `0while`, `0until`, `0exit`.

Source file: <src/lib/flow.conditionals.zero.fs>.

0leave

```
0leave ( f -- ) ( R: loop-sys -- | loop-sys ) "question-leave"
```

If *f* is zero, discard the loop-control parameters for the current nesting level and continue execution immediately following the innermost syntactically enclosing `loop` or `+loop`.

See also: `?leave`, `leave`, `unloop`, `do`, `?do`.

Source file: <src/lib/flow.MISC.fs>.

0max

```
0max ( n -- n | 0 ) "zero-max"
```

If *n* is negative, return 0; else return *n*. `0max` is a faster alternative to the idiom `0 max`.

See also: `max`, `min`, `0`.

Source file: <src/lib/math.operators.1-cell.fs>.

0repeat

```
0repeat "zero-repeat"
  Compilation: ( dest -- dest )
  Run-time:    ( f -- )
```

An alternative exit point for `begin` … `until` loops: If *f* is zero, continue execution at `begin`, otherwise continue execution after `until`.

`0repeat` is an `immediate` word.

Usage example:

```
: test ( -- )
    begin
      ...
    flag 0repeat  \ Go back to ``begin`` if flag is zero
      ...
    flag ?repeat  \ Go back to ``begin`` if flag is non-zero
      ...
    flag until    \ Go back to ``begin`` if flag is false
    ...
  ;
```

See also: `?repeat`.

Source file: <src/lib/flow.MISC.fs>.

0until

```
0until "zero-until"
  Compilation: ( C: dest -- )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0= until`.

`0until` is an `immediate` and `compile-only` word.

See also: `until`, `-until`, `+until`, `0if`, `0while`, `0exit`.

Source file: <src/lib/flow.conditionals.zero.fs>.

0while

```
0while "zero-while"
  Compilation: ( C: dest -- orig dest )
  Run-time:    ( f -- )
```

Faster and smaller alternative to the idiom `0= while`.

`0while` is an `immediate` and `compile-only` word.

See also: `while`, `-while`, `+while`, `0if`, `0until`, `0exit`.

Source file: <src/lib/flow.conditionals.zero.fs>.

1

1

```
1 ( -- 1 )
```

Return *1*. `1` is not a `constant`, but a `code` word, which is faster.

See also: `-1`, `0`, `2`.

Source file: <src/kernel.z80s>.

1+

```
1+ ( n1 -- n2 ) "one-plus"
```

Add 1 to *n1*, according to the operation of `+`, giving *n2*.

`1+` is equivalent to `1 +` but faster.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set).

See also: `1-`, `2+`, `8+`, `c@1+`, `1`, `+`.

Source file: <src/kernel.z80s>.

1+!

```
1+! ( a - ) "one-plus-store"
```

Increment the single-cell number stored at *a*.

See also: `c1+!`, `1-!`, `+!`.

Source file: <src/lib/memory.MISC.fs>.

1-

```
1- ( n1 -- n2 ) "one-minus"
```

Subtract 1 from *n1*, according to the operation of `-`, giving *n2*.

`1-` is equivalent to `1 -` but faster.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set).

See also: `1+`, `2-`, `8-`, `c@1-`, `1`, `-`.

Source file: <src/kernel.z80s>.

1-!

```
1-! ( a - ) "one-minus-store"
```

Decrement the single-cell number stored at *a*.

See also: `1+!`, `c1-!`, `-!`.

Source file: <src/lib/memory.MISC.fs>.

1-line-(located

```
1-line-(located ( ca len -- block | 0 ) "one-line-paren-located"
```

Locate the first block whose single-line header contains the string *ca len* (surrounded by spaces), and return its number. If not found, return zero. The search is case-sensitive.

Only the blocks delimited by `first-locatable` and `last-locatable` are searched.

`1-line-(located` is an alternative, deprecated action of `(located`.

Source file: <src/lib/002.need.fs>.

1/string

```
1/string ( ca1 len1 -- ca1+1 len1-1 ) "one-slash-string"
```

Adjust the character string *ca1 len1* by 1 character.

`1/string` is equivalent to the idiom `1 /string` but faster (0.9 the execution time).

See also: `/string`.

Source file: <src/kernel.z80s>.

16bin.

```
16bin. ( n -- ) "16-bin-dot"
```

Display *n* as an unsigned 16-bit binary number.

See also: `16bin.`, `32bin.`, `8bin.`, `bin.`, `binary`.

Source file: <src/lib/display.numbers.fs>.

16hex.

```
16hex. ( d -- ) "16-hex-dot"
```

Display *d* as an unsigned 16-bit hexadecimal number.

See also: `16bin.`, `32hex.`, `8hex.`, `hex.`, `hex`.

Source file: <src/lib/display.numbers.fs>.

1array

```
1array ( n1 n2 "name" -- ) "one-array"
```

Define a 1-dimension array *name* with *n1* items of *n2* bytes each.

See also: `}`, `array>items`, `2array`.

Source file: <src/lib/data.array.noble.fs>.

1line

```
1line ( -- f ) "1-line"
```

Part of `specforth-editor`: Scan the cursor line for a match to `pad` text. Return flag and update the cursor `r#` to the end of matching text, or to the start of the next line if no match is found.

See also: `#lag`, `match`.

Source file: <src/lib/prog.editor.specforth.fs>.

2

2

```
2 ( -- 2 )
```

Return *2*. `2` is not a `constant`, but a `code` word, which is faster.

See also: `-1`, `0`, `1`, `cell`.

Source file: <src/kernel.z80s>.

2!

```
2! ( x1 x2 a -- ) "two-store"
```

Store the cell pair *x1 x2* at *a*, with *x2* at *a* and *x1* at the next consecutive cell. It is equivalent to the sequence `swap over ! cell+ !`.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `2@`, `!`, `c!`.

Source file: <src/kernel.z80s>.

2!>

```
2!>
  Interpretation: ( xd "name" -- )
  Compilation:    ( "name" -- )
  Run-time:       ( xd -- )
"two-store-to"
```

A simpler and faster alternative to standard `to` and `2value`.

`2!>` is an `immediate` word.

Interpretation:

Parse *name*, which is the name of a word created by `2constant` or `2const`, and make *xd* its value.

Compilation:

Parse *name*, which is a word created by `2constant` or `2const`, and append the run-time semantics given below to the current definition.

Run-time:

Make *xd* the current value of double-cell constant *name*.

Origin: IsForth's `!>`.

See also: `!>`, `c!>`.

Source file: <src/lib/data.store-to.fs>.

2*

```
2* ( x1 -- x2 ) "two-star"
```

x2 is the result of shifting *x1* one bit toward the most-significant bit, filling the vacated least-significant bit with zero.

`2*` is equivalent to `1 lshift`, but faster.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `lshift`, `8*`, `3*`, `*`.

Source file: <src/kernel.z80s>.

2*!

```
2*! ( a -- ) "two-star-store"
```

Do a `2*` shift to the single-cell number stored at *a*.

See also: `2/!`, `2*`.

Source file: <src/lib/memory.MISC.fs>.

2+

```
2+ ( n1 -- n2 ) "two-plus"
```

Add 2 to *n1*, according to the operation of `+`, giving *n2*.

`2+` is equivalent to `2 +` but faster.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set).

See also: `2-`, `1+`, `8+`, `c@2+`, `2`, `+`.

Source file: <src/kernel.z80s>.

2,

```
2, ( x1 x2 -- ) "2-comma"
```

Definition:

```
: 2, ( x1 x2 -- ) here 2! [ 2 cells ] literal allot ;
```

Reserve two cells of data space and store *x1 x2* in them. *x2* is stored in the first cell, and *x1* is stored in the second cell.

See also: `,`, `c,`, `here`, `2!`, `cells`, `literal`, `allot`.

Source file: <src/kernel.z80s>.

2-

```
2- ( n1 -- n2 ) "two-minus"
```

Subtract 2 from *n1*, according to the operation of `-`, giving *n2*.

`2-` is equivalent to `2 -` but faster.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set).

See also: `2+`, `1-`, `8-`, `c@2-`, `2`, `-`.

Source file: <src/kernel.z80s>.

2-block-drives

```
2-block-drives ( -- )
```

Set all drives as block drives, in normal order: 1 and 2.

NOTE For convenience, when this word is loaded, it's also executed.

See also: `set-block-drives`.

Source file: <src/lib/dos.gplusdos.fs>.

2/

```
2/ ( x1 -- x2 ) "two-slash"
```

x2 is the result of shifting *x1* one bit toward the least-significant bit, leaving the most-significant bit unchanged.

`2/` is equivalent to `s>d 2 fm/mod swap drop`. `2/` is not the same as `2 /`, nor is it the same as `1 rshift`.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `/`, `rshift`, `s>d`, `fm/mod`, `2`.

Source file: <src/lib/math.operators.1-cell.fs>.

2/!

```
2/! ( a -- ) "two-slash-store"
```

Do a `2/` shift to the single-cell number stored at *a*.

See also: `2*!`, `2/`.

Source file: <src/lib/memory.MISC.fs>.

2>bstring

```
2>bstring ( x1 x2 -- ca len ) "two-to-b-string"
```

Convert *xd* to a 2-cell binary string in the `stringer`. *ca len* contains *x2 x1*, i.e. in the usual order in memory.

See also: `>bstring`, `char>string`, `chars>string`.

Source file: <src/lib/strings.MISC.fs>.

2>false

```
2>false ( x1 x2 -- false ) "two-to-false"
```

Replace *x1 x2* with `false`.

See also: `2>true`, `>false`.

Source file: <src/lib/data_stack.fs>.

2>r

```
2>r ( x1 x2 -- ) ( R: -- x1 x2 ) "two-to-r"
```

Move *x1 x2* from the data stack to the return stack. Semantically equivalent to `swap >r >r`.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `2r>`, `2r@`, `>r`.

Source file: <src/kernel.z80s>.

2>true

```
2>true ( x1 x2 -- true ) "two-to-true"
```

Replace *x1 x2* with `true`.

See also: `2>false`, `>true`.

Source file: <src/lib/data_stack.fs>.

2>x

```
2>x ( x1 x2 -- ) ( X: -- x1 x2 ) "two-to-x"
```

Move the cell pair *x1 x2* from the data stack to the current `xstack`.

See also: `2x>`, `2x@`, `>x`.

Source file: <src/lib/data.xstack.fs>.

2?

```
2? ( ca -- ) "two-question"
```

Display the double-cell signed integer stored at *a*, using the format of `d.`.

See also: `?`, `c?`, `2@`.

Source file: <src/lib/memory.MISC.fs>.

2@

```
2@ ( a -- x1 x2 ) "two-fetch"
```

Fetch the cell pair *x1 x2* stored at *a*. *x2* is stored at *a* and *x1* is stored at the next consecutive cell. It is equivalent to the sequence `dup cell+ @ swap @`.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `2!`, `@`, `c@`.

Source file: <src/kernel.z80s>.

2@+

```
2@+ ( a -- a' xd ) "two-fetch-plus"
```

Fetch *xd* from *a*. Return *a'*, which is *a* incremented by two cells. This is handy for stepping through double-cell arrays.

See also: `2@`, `@+`, `c@+`.

Source file: <src/lib/memory.MISC.fs>.

2array

```
2array ( n1 n2 n3 "name" -- ) "two-array"
```

Define a 2-dimension array *name* with *n1* x *n2* items of *n3* bytes each.

See also: `}}`, `1array`.

Source file: <src/lib/data.array.noble.fs>.

2array<

```
2array< ( a1 n -- a2 ) "two-array-from"
```

Return address *a2* of element *n* of a 1-dimension double-cell array *a1*.

`2array<` is written in Z80. Its equivalent definition in Forth is the following:

```
: 2array< ( a1 n -- a2 ) [ 2 cells ] literal * + ;
```

See also: `2array>`, `array<`.

Source file: <src/lib/data.array.COMMON.fs>.

2array>

```
2array> ( n a1 -- a2 ) "two-array-to"
```

Return address *a2* of element *n* of a 1-dimension double-cell array *a1*. `2array>` is a common factor of `2avalue` and `2avariable`.

`2array>` is written in Z80. Its equivalent definition in Forth is the following:

```
: 2array> ( n a1 -- a2 ) swap [ 2 cells ] literal * + ;
```

See also: `2array<`, `array>`.

Source file: <src/lib/data.array.COMMON.fs>.

2ato

```
2ato ( xd n "name" -- ) "two-a-to"
```

Store *xd* into element *n* of 1-dimension double-cell values array *name*.

`2ato` is an `immediate` word.

See also: `2avalue`, `(2ato`.

Source file: <src/lib/data.array.value.fs>.

2avalue

```
2avalue ( n "name" -- ) "two-a-value"
```

Create a 1-dimension double-cell values array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — xd)

Return contents *xd* of element *n*.

See also: `2ato`.

Source file: <src/lib/data.array.value.fs>.

2avariable

```
2avariable ( n "name" -- ) "two-a-variable"
```

Create a 1-dimension double-cell variables array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — a)

Return address *a* of element *n*.

See also: `avariable`, `cavariable`, `far2avariable`.

Source file: <src/lib/data.array.variable.fs>.

2const

```
2const ( x1 x2 "name" -- ) "two-const"
```

Create a double fast constant *name*, with value *x1 x2*.

A double fast constant works like an ordinary `2constant`, except its value is compiled as a literal.

Origin: IsForth's `const`.

See also: `[2const]`, `const`, `cconst`.

Source file: <src/lib/data.const.fs>.

2constant

```
2constant ( x1 x2 "name" -- ) "two-constant"
```

Parse *name*. `create` a definition for *name* that will place *x1 x2* on the stack. *name* is referred to as a "two-constant".

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `constant`, `cconstant`, `2!>`, `2const`, `[2const]`, `2value`, `2variable`.

Source file: <src/kernel.z80s>.

2drop

```
2drop ( x1 x2 -- ) "two-drop"
```

Remove cell pair *x1 x2* from the stack.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `drop`, `nip`.

Source file: <src/kernel.z80s>.

2dup

```
2dup ( x1 x2 -- x1 x2 x1 x2 ) "two-dup"
```

Duplicate cell pair *x1 x2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `dup`, `2over`, `2drop`, `3dup`, `4dup`.

Source file: <src/kernel.z80s>.

2entry:

```
2entry: ( dx wid "name" -- ) "two-entry-colon"
```

Create a double-cell entry *name* in the `associative-list` *wid*, with value *dx*.

See also: `entry:`, `centry:`, `sentry:`, `create-entry`.

Source file: <src/lib/data.associative-list.fs>.

2field:

```
2field: ( n1 "name" -- n2 ) "two-field-colon"
```

Parse *name*. *offset* is the first double-cell aligned value greater than or equal to *n1*. *n2* = *offset* + 2 cells.

Create a definition for *name* with the execution semantics defined below.

name execution: `( a1 -- a2 )`

Add the *offset* calculated during the compile-time action to *a1* giving the address *a2*.

See also: `begin-structure`, `+field`.

Source file: <src/lib/data.begin-structure.fs>.

2lit

```
2lit ( -- x1 x2 ) "two-lit"
```

Return *x1 x2*, which was compiled by `2literal` after `2lit`.

`2lit` is a `compile-only` word.

See also: `lit`, `clit`.

Source file: <src/kernel.z80s>.

2literal

```
2literal ( x1 x2 -- ) "two-literal"
```

Compile *x1 x2* in the current definition.

`2literal` is an `immediate` and `compile-only` word.

Definition:

```
: 2literal ( x1 x2 -- ) postpone 2lit 2, ; immediate compile-only
```

See also: `2lit`, `literal`, `cliteral`, `xliteral`, `]2l`.

Source file: <src/kernel.z80s>.

2local

```
2local ( a -- )
```

Save the value of double-cell variable *a*, which will be restored at the end of the current definition.

`2local` is a `compile-only` word.

Usage example:

```
2variable v
1. v 2!  v 2@ d. \ default value

: test ( -- )
  v 2local
  v 2@ u.  1887. v 2!  v 2@ d. ;

v 2@ d. \ default value
```

See also: `local`, `clocal`, `arguments`, `anon`.

Source file: <src/lib/locals.local.fs>.

2ndrop

```
2ndrop ( dx1...dxn n -- ) "two-n-drop"
```

Drop *n* double cell items from the stack.

See also: `ndrop`, `drop`, `2drop`.

Source file: <src/lib/data_stack.fs>.

2nip

```
2nip ( x1 x2 x3 x4 -- x3 x4 ) "two-nip"
```

See also: `nip`.

Source file: <src/lib/data_stack.fs>.

2over

```
2over ( x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2 ) "two-over"
```

Copy cell pair *x1 x2* on top of the stack.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `over`, `2swap`.

Source file: <src/kernel.z80s>.

2r>

```
2r> ( -- x1 x2 ) ( R: x1 x2 -- ) "two-r-from"
```

Move *x1 x2* from the return stack to the data stack. Semantically equivalent to `r> r> swap`.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `2>r`, `2r@`, `r>`.

Source file: <src/kernel.z80s>.

2r@

```
2r@ ( -- x1 x2 ) ( R: x1 x2 -- x1 x2 ) "two-r-fetch"
```

Copy *x1 x2* from the return stack to the data stack. Semantically equivalent to `r> r> 2dup >r >r swap`.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `2>r`, `2r>`, `r@`.

Source file: <src/kernel.z80s>.

2rdrop

```
2rdrop ( R: x1 x2 -- ) "two-r-drop"
```

Remove *x1 x2* from the return stack.

See also: `rdrop`, `2drop`.

Source file: <src/lib/return_stack.fs>.

2rot

```
2rot ( x1 x2 x3 x4 x5 x6 -- x3 x4 x5 x6 x1 x2 ) "two-rot"
```

Source file: <src/lib/data_stack.fs>.

2storer

```
2storer ( xd a "name" -- ) `two-storer"
```

Define a word *name* which, when executed, will cause that *xd* be stored at *a*.

Origin: variant of the word `set` found in Forth-79 (Reference Word Set) and Forth-83 (Appendix B. Uncontrolled Reference Words).

Source file: <src/lib/data.storer.fs>.

2swap

```
2swap ( x1 x2 x3 x4 -- x3 x4 x1 x2 ) "two-swap"
```

Exchange the top two cell pairs.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `swap`, `2over`.

Source file: <src/kernel.z80s>.

2switch

```
2switch ( xd switch -- ) "two-switch"
```

Execute the switch *switch* for the key *xd*.

See also: `switch:`, `:2clause`.

Source file: <src/lib/flow.switch-colon.fs>.

2toval

```
2toval ( -- ) "two-to-val"
```

Change the default behaviour of words created by `2val`: make them store a new value instead of returning its actual one.

`2toval` and `2val` are a non-parsing alternative to the standard `to` and `2value`.

See also: `toval`, `ctoval`.

Source file: <src/lib/data.val.fs>.

2user

```
2user ( "name" -- ) "two-user"
```

Parse *name*. Create a user double-cell variable *name* in the first available offset within the user area. When *name* is later executed, its absolute user area storage address is placed on the stack.

See also: `user`, `ucreate`, `uallot`, `?user`.

Source file: <src/lib/data.user.fs>.

2val

```
2val ( x1 x2 "name" -- ) "two-val"
```

Create a definition for *name* that will place *x1 x2* on the stack (unless `2toval` is used first) and then will execute `init-2val`.

`2val` is an alternative to the standard `2value`.

See also: `val`, `cval`, `2variable`, `2constant`.

Source file: <src/lib/data.val.fs>.

2value

```
2value ( x1 x2 "name" -- ) "two-value"
```

Create a definition *name* with initial value *x1 x2*. When *name* is later executed, *x1 x2* will be placed on the stack. `to` can be used to assign a new value to *name*.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `cvalue`, `value`, `2constant`, `2variable`, `2val`.

Source file: <src/lib/data.value.fs>.

2variable

```
2variable ( "name" -- ) "two-variable"
```

Parse *name*. `create` a definition for *name*, which is referred to as a "two-variable". `allot` two cells of data space, the data field of *name*, to hold the contents of the two-variable. When *name* is later executed, the address of its data field is placed on the stack.

The program is responsible for initializing the contents of the two-variable.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `cells`, `literal`, `variable`, `2variable`, `2constant`.

Source file: <src/lib/data.MISC.fs>.

2x>

```
2x> ( -- x1 x2 ) ( X: x1 x2 -- ) "two-x-from"
```

Move the cell pair *x1 x2* from the current `xstack` to the data stack.

See also: `2>x`, `2x@`, `x>`.

Source file: <src/lib/data.xstack.fs>.

2x@

```
2x@ ( -- x1 x2 ) ( X: x1 x2 -- x1 x2 ) "two-x-fetch"
```

Copy the cell pair *x1 x2* from the current `xstack` to the data stack.

Source file: <src/lib/data.xstack.fs>.

2xdrop

```
2xdrop ( X: x1 x2 -- ) "two-x-drop"
```

Remove the cell pair *x1 x2* from the current `xstack`.

See also: `xdrop`.

Source file: <src/lib/data.xstack.fs>.

2xdup

```
2xdup ( X: x1 x2 -- x1 x2 x1 x2 ) "two-x-dup"
```

Duplicate the cell pair *x1 x2* in the current `xstack`.

See also: `xdup`.

Source file: <src/lib/data.xstack.fs>.

3

3*

```
3* ( n1 -- n2 ) "three-plus"
```

Multiply *n1* by 3 giving *n2*.

`3*` is equivalent to `3 *` or `dup dup + +`, but faster.

See also: `2*`, `8*`, `*`, `+`.

Source file: <src/lib/math.operators.1-cell.fs>.

32bin.

```
32bin. ( d -- ) "32-bin-dot"
```

Display *d* as an unsigned 32-bit binary number.

See also: `32hex.`, `16bin.`, `8bin.`, `bin.`, `binary`.

Source file: <src/lib/display.numbers.fs>.

32hex.

```
32hex. ( d -- ) "32-hex-dot"
```

Display *d* as an unsigned 32-bit hexadecimal number.

See also: `32bin.`, `16hex.`, `8hex.`, `hex.`, `hex`.

Source file: <src/lib/display.numbers.fs>.

3drop

```
3drop ( x1 x2 x3 -- ) "three-drop"
```

See also: `3dup`, `drop`, `2drop`, `4drop`.

Source file: <src/lib/data_stack.fs>.

3dup

```
3dup ( x1 x2 x3 -- x1 x2 x3 x1 x2 x3 ) "three-dup"
```

`3dup` is written is Z80. An equivalent definition in Forth is the following:

```
: 3dup ( x1 x2 x3 -- x1 x2 x3 x1 x2 x3 ) dup 2over rot ;
```

See also: `3drop`, `dup`, `2dup`, `4dup`.

Source file: <src/lib/data_stack.fs>.

4

4drop

```
4drop ( x1 x2 x3 x4 -- ) "four-drop"
```

See also: `4dup`, `drop`, `2drop`, `3drop`.

Source file: <src/lib/data_stack.fs>.

4dup

```
4dup ( x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2 x3 x4 ) "four-dup"
```

See also: `4drop`, `dup`, `2dup`, `3dup`.

Source file: <src/lib/data_stack.fs>.

8

8*

```
8* ( x1 -- x2 ) "eight-star"
```

x2 is the result of shifting *x1* three bits toward the most-significant bit, filling the vacated least-significant bit with zero.

`8*` is equivalent to `3 lshift` or `2* 2* 2*`, but faster.

See also: `2*`, `3*`, `lshift`, `8+`, `8-`, `*`.

Source file: <src/lib/math.operators.1-cell.fs>.

8+

```
8+ ( n1 -- n2 ) "eight-plus"
```

Add 8 to *n1*, according to the operation of `+`, giving *n2*.

`8+` is equivalent to `8 +` but faster.

See also: `8-`, `1+`, `2+`, `8*`, `+`.

Source file: <src/lib/math.operators.1-cell.fs>.

8-

```
8- ( n1 -- n2 ) "eight-minus"
```

Subtract 8 from *n1*, according to the operation of `-`, giving *n2*.

`8-` is equivalent to `8 -` but faster.

See also: `8+`, `1-`, `2-`, `8*`, `-`.

Source file: <src/lib/math.operators.1-cell.fs>.

8bin.

```
8bin. ( n -- ) "8-bin-dot"
```

Display *n* as an unsigned 8-bit binary number.

See also: `8hex.`, `32bin.`, `16bin.`, `bin.`, `binary`.

Source file: <src/lib/display.numbers.fs>.

8hex.

```
8hex. ( d -- ) "8-hex-dot"
```

Display *d* as an unsigned 8-bit hexadecimal number.

See also: `8bin.`, `16hex.`, `hex.`, `hex`.

Source file: <src/lib/display.numbers.fs>.

:

:

```
: ( "name" -- ) "colon"
```

Parse *name*. Create a definition for *name*, called a "colon definition". Enter compilation `state` and start the current definition. Append the initiation semantics given below to the current definition.

Initiation: `( i*x -- i*x ) ( R: -- nest-sys )`

Save implementation-dependent information *nest-sys* about the calling definition. The stack effects *i*x* represent arguments to *name*.

name execution: `( i*x -- j*x )`

Execute the definition name. The stack effects *i*x* and *j*x* represent arguments to and results from *name*, respectively.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `;`, `does>`, `header`.

Source file: <src/kernel.z80s>.

:2clause

```
:2clause ( xd switch -- ) "colon-two-clause"
```

Start the definition of a switch clause *xd* for switch *switch*.

See also: `switch:`, `2switch`.

Source file: <src/lib/flow.switch-colon.fs>.

::

```
:: ( class "name" -- ) "colon-colon"
```

Compile the method for the selector *name* of the class *class* (not immediate!).

Source file: <src/lib/objects.mini-oof.fs>.

:cclause

```
:cclause (  switch -- ) "colon-c-clause"
```

Start the definition of a switch clause *c* for switch *switch*.

See also: `switch:`, `cswitch`.

Source file: <src/lib/flow.switch-colon.fs>.

:clause

```
:clause ( x switch -- ) "colon-clause"
```

Start the definition of a switch clause *x* for switch *switch*.

See also: `switch:`, `switch`.

Source file: <src/lib/flow.switch-colon.fs>.

:noname

```
:noname ( -- xt ) "colon-no-name"
```

Create an execution token *xt*. Enter compilation `state` and start the current definition, which can be executed later by using *xt*.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `nextname`.

Source file: <src/lib/define.MISC.fs>.

:switch

```
:switch ( xt "name" -- a ) "colon-switch"
```

Create a code switch *name* whose default action is given by *xt*. Leave the address *a* of the head of its list on the stack.

The head *a* of the switch structure is the address of a 2-cell structure, with the following contents:

1. Link to the last clause of the switch
2. Execution token of the default action

Usage example:

```
: one   ( -- )   ." unu " ;
: two   ( -- )   ." du "  ;
: three ( -- )   ." tri " ;
  \ clauses of the switch

: many  ( n -- ) . ." is too much! " ;
  \ default action of the switch

' many :switch .number

  ' one   1 <switch
  ' two   2 <switch
  ' three 3 <switch drop

cr 1 .number 2 .number 3 .number 4 .number

' .number >body  :noname  ." kvar " ; 4 <switch drop
  \ add a new nameless clause for number 4

cr 1 .number 2 .number 3 .number 4 .number
```

NOTE `[switch` is the syntactic-sugar variant of `:switch`.

Origin: SwiftForth.

See also: `<switch`, `[switch`, `switcher`.

Source file: <src/lib/flow.bracket-switch.fs>.

;

;

```
; "semicolon"
  Compilation: ( -- )
  Run-time:    ( -- ) ( R: nest-sys -- )
```

Compilation: Append the run-time semantics below to the current definition. End the current definition, allow it to be found in the dictionary and enter interpretation state.

Run-time: Return to the calling definition specified by *nest-sys*.

`;` is an `immediate` and `compile-only` word.

Definition:

```
: ; \ Compilation: ( -- )
    \ Run-time:    ( -- ) ( R: nest-sys -- )
  postpone exit finish-code ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `exit`, `:`, `finish-code`, `;code`.

Source file: <src/kernel.z80s>.

;]

```
;] "semicolon-bracket"
  Compilation: ( orig xt1 -- )
  Run-time:    ( -- xt2 )
```

End a quotation started by `[:`.

`;]` is an `immediate` and `compile-only` word.

Compilation:

End the current nested definition, and resume compilation to the previous (containing) current definition, identified by *xt1*. Resolve the branch from *orig* left by `[:`. Append the following run-time to the (containing) current definition:

Run-time:

xt2 is the execution token of the nested definition.

Source file: <src/lib/flow.bracket-colon.fs>.

;and

```
;and ( -- ) "colon-and"
```

Allow continuation of a definition where `make` is used.

```
doer flashes
cls \ does nothing
: activate ( -- ) make cls page ;and ." cls is ready" ;
activate \ reconfigure ``cls`` and display "cls is ready"
cls \ do ``page``
```

`;and` is an `immediate` word.

See also: `undo`.

Source file: <src/lib/flow.doer.fs>.

;code

```
;code "semicolon-code"
  Compilation: ( -- )
  Run-time:    ( -- ) ( R: nest-sys -- )
```

Define the execution-time action of a word created by a low-level defining word. Used in the form:

```
: namex ... create ... ;code ... end-code

namex name
```

where `create` could be also any user defined word which executes `create`.

`;code` marks the termination of the defining part of the defining word *namex* and then begins the definition of the execution-time action for words that will later be defined by *namex*. When *name* is called, its parameter field address is in register HL and the `assembler` code compiled between `;code` and `end-code` is executed.

Detailed description:

Compilation:

Append the run-time semantics below to the current definition. End the current definition, allow it to be found in the dictionary, and enter interpretation `state`.

Enter `assembler` mode by executing `asm`, until `end-code` is executed.

Run-time:

Replace the execution semantics of the most recent definition, which should be defined with `create` or a user-defined word that calls `create`, with the name execution semantics given below. Return control to the calling definition specified by *nest-sys*.

Initiation: `( i*x -- i*x dfa ) ( R: -- nest-sys2 )`

Save information *nest-sys2* about the calling definition. Place *name*'s data field address *dfa* on the stack. The stack effects *i*x* represent arguments to name.

name execution:

Perform the machine code sequence that was generated following `;code` and finished by `end-code`.

`;code` is an `immediate` and `compile-only` word.

Usage example:

```
: border-changer ( n -- )
  create c, ;code ( -- ) m a ld, FE out, jpnext, end-code

0 border-changer black-border
1 border-changer blue-border
2 border-changer red-border
```

Which is equivalent to:

```
: border-changer ( n -- )
  create c, does> ( -- ) ( dfa ) c@ border ;

0 border-changer black-border
1 border-changer blue-border
2 border-changer red-border
```

Origin: fig-Forth, Forth-79 (Assembler Word Set), Forth-83 (Assembler Extension Word Set), Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

See also: `(;code`, `does>`, `asm`, `create`.

Source file: <src/lib/assembler.MISC.fs>.

<

<

```
< ( n1 n2 -- f ) "less-than"
```

f is true if and only if *n1* is less than *n2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `>`, `u<`, `0<`, `min`.

Source file: <src/kernel.z80s>.

<#

```
<# ( -- ) "less-number-sign"
```

Initialize the pictured numeric output process: Set `hld` to its initial value, right below `pad`.

Definition:

```
: <# ( -- ) pad hld ! ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `/hold`, `#>`, `#`, `#s`, `hold`, `holds`, `sign`.

Source file: <src/kernel.z80s>.

<<

```
<< ( -- ca +n ) "less-than-less-than"
```

Mark the start of a code zone to be dumped by `>>`. *ca* is the current data-pointer and *+n* is the current `depth`. Both of them are used by `>>`. See `>>` for a usage example.

Origin: Pygmy Forth.

Source file: <src/lib/assembler.MISC.fs>.

<=

```
<= ( n1 n2 -- f ) "less-or-equal"
```

f is `true` if and only if *n1* is less than or equal to *n2*.

See also: `>=`, `u<=`, `0<=`.

Source file: <src/lib/math.operators.1-cell.fs>.

<=>

```
<=> ( n1 n2 -- -1|0|1 ) "less-or-equal-or-greater"
```

If *n1* equals *n2*, return zero. If *n1* is less than *n2*, return negative one. If *n1* is greater than *n2*, return positive one.

See also: `polarity`, `<`, `=`, `>`.

Source file: <src/lib/math.operators.1-cell.fs>.

<>

```
<> ( x1 x2 -- f ) "not-equals"
```

f is true only and only if *x1* is not bit-for-bit the same as *x1*.

Origin: Forth-79 (Reference Word Set), Forth-83 (Uncontrolled Reference Words), Forth-94 (CORE), Forth-2012 (CORE).

See also: `=`, `>`, `<`.

Source file: <src/kernel.z80s>.

<bin

```
<bin ( -- ) "start-bin"
```

Start a code zone where binary radix is the default, by saving the current value of `base` to `base'` and executing `binary`. The zone is finished by `bin>`.

See also: `<hex`.

Source file: <src/lib/display.numbers.fs>.

<hex

```
<hex ( -- ) "start-hex"
```

Start a code zone where hexadecimal radix is the default, by save the current value of `base` to `base'` and executing `hex`. The zone is finished by `hex>`.

Origin: lina.

See also: `<bin`.

Source file: <src/lib/display.numbers.fs>.

<is>

```
<is> ( xt "name" -- ) "less-is"
```

Set *name*, which was defined by `defer`, to execute *xt*.

`<is>` is a factor of `is`.

Origin: Gforth.

See also: `[is]`.

Source file: <src/lib/define.deferred.fs>.

<mark

```
<mark ( C: -- dest ) "backward-mark"
```

dest is the current data-space pointer, to be used as the destination of a backward branch. *dest* is typically only used by `<resolve` to compile a branch address.

`<mark` is an `alias` of `here`.

Origin: Forth-83 (System Extension Word Set).

See also: `>mark`, `begin`.

Source file: <src/kernel.z80s>.

<resolve

```
<resolve ( C: dest -- ) "backward-resolve"
```

Resolve a backward branch. Compile a branch address using *dest*, the address left by `<mark`, as the destination address. Used at the source of a backward branch after either `branch` or `?branch` or `0branch`.

`<resolve` is an `alias` of `,`.

Origin: Forth-83 (System Extension Word Set).

Source file: <src/kernel.z80s>.

<rresolve

```
<rresolve ( dest -- ) "less-than-r-resolve"
```

Resolve a Z80 `assembler` backward relative branch reference *dest*.

See also: `>rresolve`, `rresolve`.

Source file: <src/lib/assembler.fs>.

<switch

```
<switch ( a xt n -- a ) "start-switch"
```

Define a new clause of a `:switch` structure whose head is *a* to execute *xt* when the key *n* is matched.

The switch clauses are 3-cell structures:

1. Link to the previous clause of the switch
2. Key
3. Execution token

Origin: SwiftForth.

Source file: <src/lib/flow.bracket-switch.fs>.

=

=

```
= ( x1 x2 -- f ) "equals"
```

f is true only and only if *x1* is bit-for-bit the same as *x1*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `<>`, `>`, `<`.

Source file: <src/kernel.z80s>.

>

>

```
> ( n1 n2 -- f ) "greater-than"
```

f is true if and only if *n1* is greater than *n2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `<`, `u>`, `0>`, `max`.

Source file: <src/kernel.z80s>.

>=

```
>= ( n1 n2 -- f ) "greater-or-equal"
```

f is `true` if and only if *n1* is greater than or equal to *n2*.

See also: `<=`, `u>=`, `0>=`.

Source file: <src/lib/math.operators.1-cell.fs>.

>>

```
>> ( ca +n -- ) "greater-than-greater-than"
```

Display starting address *ca* as a 16-bit hexadecimal number. Then dump the code compiled in data space from *ca* to the current data-space pointer, in hexadecimal. *+n* is used for error checking. *ca* and *+n* were left by <<.

`<<` and `>>` allow you to dump short (or long) snippets of assembly code to the screen for your inspection. If you want to see how a piece of assembly code gets assembled, just put it between the brackets.

Usage example:

```
create useless-code-routine ( -- a )
  asm  << C9 c, >> end-asm

need assembler

code useless-code-word ( n1 -- n1 )
  << h pop, h incp, h decp, h push, jpnext, >>
end-code
```

Origin: Pygmy Forth.

See also: `dump`, `wdump`, `assembler`.

Source file: <src/lib/assembler.MISC.fs>.

>>link

```
>>link ( xtp -- lfa ) "to-to-link"
```

Convert *xtp* into its corresponding *lfa*.

See also: `>>name`, `name>link`.

Source file: <src/lib/compilation.fs>.

>>name

```
>>name ( xtp -- nt ) "to-to-name"
```

Convert *xtp* into its corresponding *nt*.

See also: `name>>`, `>>link`, `>name`.

Source file: <src/lib/compilation.fs>.

>action

```
>action ( xt -- a ) "to-action"
```

Return the address *a* that contains the execution token currently associated to the deferred word *xt*.

See also: `defer`, `action-of`, `defer!`, `defer@`.

Source file: <src/kernel.z80s>.

>amark

```
>amark ( -- a ) "greater-than-a-mark"
```

Leave the address of a Z80 `assembler` absolute forward reference.

Source file: <src/lib/assembler.fs>.

>aresolve

```
>aresolve ( orig -- ) "greater-than-a-resolve"
```

Resolve a Z80 `assembler` forward absolute branch reference *orig*.

See also: `>amark`.

Source file: <src/lib/assembler.fs>.

>body

```
>body  ( xt -- dfa ) "to-body"
```

Convert *xt* into its corresponding *dfa*.

If *xt* is for a word defined by `create`, *dfa* is the address that `here` would have returned had it been executed immediately after the execution of the `create` that defined *xt*.

If *xt* is for a word defined by `variable`, `2variable`, `cvariable`, `constant`, `2constant` and `cconstant`, *dfa* is the address containing their value.

If *xt* is for a word defined by `:`, *dfa* is the address of its compiled definition.

If *xt* is for a word defined by `code`, *dfa* makes no sense.

dfa is always in data space.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `body>`, `name>body`, `>name`.

Source file: <src/lib/compilation.fs>.

>bstring

```
>bstring ( x -- ca len ) "to-b-string"
```

Convert *x* to a 1-cell binary string *ca len* in the `stringer`. *ca len* contains *x* "as is", as stored in memory.

See also: `2>bstring`, `chars>string`, `char>string`.

Source file: <src/lib/strings.MISC.fs>.

>digit

```
>digit ( n -- c ) "to-digit"
```

Convert a number to its character digit: 0 .. 9A .. Z.

`>digit` is written in Z80. Its equivalent definition if Forth is the following:

```
: >digit ( n -- c )
  dup 9 > [ 'A' '0' - 1+ ] literal and + '0' + ;
```

Source file: <src/kernel.z80s>.

>drive-block

```
>drive-block ( u1 -- u2 ) "to-drive-block"
```

Convert block *u1* to its equivalent *u2* in its corresponding disk drive, which is set the current drive.

`>drive-block` is a deferred word (see `defer`) whose default action is `noop`. Its action is set to `(>drive-block` when `block-drives` is loaded.

Source file: <src/kernel.z80s>.

>e

```
>e ( x a -- ) "to-e"
```

Move *x* to the extra stack *a* defined with `estack`.

See also: `e>`, `e@`.

Source file: <src/lib/data.estack.fs>.

>esc-order

```
>esc-order ( wid -- ) "to-esc-order"
```

Push *wid* on the escaped strings search order.

See also: `set-esc-order`, `get-esc-order`, `esc-standard-chars-wordlist`, `esc-block-chars-wordlist`, `esc-udg-chars-wordlist`.

Source file: <src/lib/strings.escaped.fs>.

>false

```
>false ( x -- false ) "to-false"
```

Replace *x* with `false`.

See also: `>true`, `2>false`.

Source file: <src/lib/data_stack.fs>.

>file

```
>file ( ca1 len1 ca2 len2 -- ior ) "to-file"
```

Save memory region *ca1 len1* to a file named by the string *ca2 len2*, and return the I/O result code *ior*.

See also: `file>`, `(>file`.

Source file: <src/lib/dos.gplusdos.fs>.

>form

```
>form ( cols rows -- ) "to-form"
```

Adapt the cursor position of the current display mode to a display mode whose `form` is *cols rows*.

`>form` is used by the display modes, e.g. `mode-32` and `mode-64ao`.

NOTE When `>form` is executed, the action of `at-xy` must be that of the new mode, but `xy`, `rows` and `columns` must still return the values of the current (old) mode.

Source file: <src/lib/display.mode.COMMON.fs>.

>graphic-ascii-char

```
>graphic-ascii-char ( c1 -- c1 | c2 )
```

If character *c1* is a printable ASCII character, return it, else return the character returned by `default-graphic-ascii-char`.

See also: `graphic-ascii-char?`.

Source file: <src/lib/chars.fs>.

>in

```
>in ( -- a ) "to-in"
```

A `user` variable. *a* is the address of a cell containing the offset in characters from the start of the input buffer to the start of the parse area.

Source file: <src/kernel.z80s>.

>in/l

```
>in/l ( -- n ) "to-in-slash-l"
```

Return number *n* of characters already interpreted in the current line of the block being interpreted. No check is done whether any block is actually being interpreted.

Definition:

```
: >in/l ( -- n ) >in @ c/l mod ;
```

See also: `blk-line`, `->in/l`, `>in`, `c/l`.

Source file: <src/kernel.z80s>.

>l

```
>l ( b -- a ) "to-l"
```

a is the address of label *b* in the `labels` table.

Source file: <src/lib/assembler.labels.fs>.

>mark

```
>mark ( C: -- orig ) "forward-mark"
```

Compile space in the dictionary for a branch address which will later be resolved by `>resolve`.

Used at the source of a forward branch. Typically used after either `branch`, `0branch` or `?branch`.

Definition:

```
: >mark ( C: -- orig ) here 0 , ;
```

Origin: Forth-83 (System Extension Word Set).

See also: `<mark`.

Source file: <src/kernel.z80s>.

>name

```
>name ( xt -- nt | 0 ) "to-name"
```

Try to find the name token *nt* of the word represented by execution token *xt*. Return 0 if it fails.

NOTE `>name` searches all word lists, from newest to oldest; and the searching of every word list is done also from the newest to the oldest definition. The first header whose execution token pointer contains *xt* is a match. Therefore, when a word has additional headers created by `alias` or `synonym`, the *nt* of its latest alias or synonym is found first.

Origin: Gforth.

See also: `>name/order`, `>oldest-name`, `>oldest-name/order`, `>oldest-name/fast`, `name>`, `>body`, `name>body`, `name>name`, `>>name`.

Source file: <src/lib/compilation.fs>.

>name/order

```
>name/order ( xt -- nt | 0 ) "to-name-slash-order"
```

Try to find the name token *nt* of the word represented by execution token *xt*, in the current search `order`. Return 0 if it fails.

NOTE `>name/order` searches all word lists in the current search `order`, and the searching of every word list is done from the newest to the oldest definition. The first header whose execution token pointer contains *xt* is a match. Therefore, when a word has additional headers created by `alias` or `synonym`, the *nt* of its latest alias or synonym in the current search order is found first.

See also: `>name`, `>oldest-name/order`, `>oldest-name`, `>oldest-name/fast`, `name>`, `>body`, `name>body`, `name>name`, `name>>`.

Source file: <src/lib/compilation.fs>.

>number

```
>number ( ud1 ca1 len1 -- ud2 ca2 len2 ) "to-number"
```

ud2 is the unsigned result of converting the characters within the string specified by *ca1 len1* into digits, using the number in `base`, and adding each into *ud1* after multiplying *ud1* by the number in `base`. Conversion continues left-to-right until a character that is not convertible, including any "+" or "-", is encountered or the string is entirely converted. *ca2* is the location of the first unconverted character or the first character past the end of the string if the string was entirely converted. *len2* is the number of unconverted characters in the string.

Definition:

```
: >number ( d1 ca1 len1 -- d2 ca2 len2 )
  begin  dup  while
    over c@ base @ digit? while
       >r 2swap r> swap base @ um* drop rot base @
       um* d+ 2swap 1 /string
  repeat then ;
```

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `number?`, `number`.

Source file: <src/kernel.z80s>.

>oldest-name

```
>oldest-name ( xt -- nt | 0 ) "to-oldest-name"
```

Try to find the oldest name token *nt* of the word represented by execution token *xt*, in the current search `order`. Return 0 if it fails.

NOTE `>oldest-name` searches all word lists, from newest to oldest; and the searching of every word list is done also from the newest to the oldest definition. The oldest header whose execution token pointer contains *xt* is a match. Therefore, when a word has additional headers created by `alias` or `synonym`, the *nt* of the original word is returned.

See also: `>oldest-name/order`, `>oldest-name/fast`, `>name`, `>name/order`, `name>`, `>body`, `name>body`, `name>name`, `name>>`.

Source file: <src/lib/compilation.fs>.

>oldest-name/fast

```
>oldest-name/fast ( xt -- nt | 0 ) "to-oldest-name-slash-fast"
```

Try to find the name token *nt* of the word represented by execution token *xt*. Return 0 if it fails.

`>oldest-name/fast` searches the whole dictionary, from the oldest definition to the newest one, for the first definition whose execution token pointer contains *xt*. This way, when a word has additional headers created by `alias` or `synonym`, its original name is found first.

WARNING `>oldest-name/fast` is not absolutely reliable, because it uses `name>name` to calculate the address of the next header. If something other than definition headers was compiled in name space or the name-space pointer `np` was altered between two definitions, the linking will fail and the algorithm probably will enter and endless loop.

Origin: Gforth.

See also: `>oldest-name`, `>oldest-name/order`, `>name`, `>name/order`, `name>`, `>body`, `name>body`, `name>name`, `>>name`.

Source file: <src/lib/compilation.fs>.

>oldest-name/order

```
>oldest-name/order ( xt -- nt | 0 ) "to-oldest-name-slash-order"
```

Try to find the oldest name token *nt* of the word represented by execution token *xt*, in the current search `order`. Return 0 if it fails.

> **NOTE** `>oldest-name/order` searches all word lists in the current search `order`, and the searching of every word list is done from the newest to the oldest definition. The oldest header whose execution token pointer contains *xt* is a match. Therefore, when a word has additional headers created by `alias` or `synonym`, the *nt* of the original word is returned.

See also: `>oldest-name`, `>oldest-name/fast`, `>name`, `>name/order`, `name>`, `>body`, `name>body`, `name>name`, `name>>`.

Source file: <src/lib/compilation.fs>.

>order

```
>order ( wid -- ) "to-order"
```

Push word list identifier *wid* on the search order.

Definition:

```
: >order ( wid -- ) also context ! ;
```

Origin: Gforth.

See also: `previous`, `also`, `set-order`, `context`.

Source file: <src/kernel.z80s>.

>r

```
>r ( x -- ) ( R: -- x ) "to-r"
```

Move *x* from the data stack to the return stack.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `r>`, `r@`, 2>r`, `dup>r`.

Source file: <src/kernel.z80s>.

>resolve

```
>resolve ( C: orig -- ) "forward-resolve"
```

Resolve a forward branch by placing the address of the current data-space pointer into the space compiled by `>mark`.

Definition:

```
: >resolve ( C: orig -- ) here swap ! ;
```

Origin: Forth-83 (System Extension Word Set).

See also: `here`, `<resolve`.

Source file: <src/kernel.z80s>.

>rmark

```
>rmark ( -- orig ) "greater-than-r-mark"
```

Leave the origin address of a Z80 `assembler` forward relative branch just compiled, to be resolved by `>rresolve`.

Source file: <src/lib/assembler.fs>.

>rresolve

```
>rresolve ( orig -- ) "greater-than-r-resolve"
```

Resolve a Z80 `assembler` forward relative branch reference *orig*.

See also: `<rresolve`, `rresolve`.

Source file: <src/lib/assembler.fs>.

>stringer

```
>stringer ( ca1 len1 -- ca2 len1 ) "to-stringer"
```

Copy string *ca1 len1* to the `stringer` and return it as *ca2 len1*.

Definition:

```
: >stringer ( ca1 len1 -- ca2 len1 )
  dup allocate-stringer swap 2dup 2>r move 2r> ;
```

See also: `allocate-stringer`, `far>stringer`.

Source file: <src/kernel.z80s>.

>tape-file

```
>tape-file ( ca1 len1 ca2 len2 -- ) "to-tape-file"
```

Write a memory region *ca1 len1* into a tape file *ca2 len2*.

See also: `tape-file>`, `(>tape-file` , `>file`.

Source file: <src/lib/tape.fs>.

>true

```
>true ( x -- true ) "to-true"
```

Replace *x* with `true`.

See also: `>false`, `2>true`.

Source file: <src/lib/data_stack.fs>.

>ufia1

```
>ufia1 ( a -- ) "to-u-f-i-a-1"
```

Move a UFIA (User File Information Area) from *a* to `ufia1`.

See also: `>ufia2`, `>ufiax`, `ufia`, `/ufia`.

Source file: <src/lib/dos.gplusdos.fs>.

>ufia2

```
>ufia2 ( a -- ) "to-u-f-i-a-2"
```

Move a UFIA (User File Information Area) from *a* to `ufia2`.

See also: `>ufia1`, `>ufiax`, `ufia`, `/ufia`.

Source file: <src/lib/dos.gplusdos.fs>.

>ufiax

```
>ufiax ( a1 a2 -- ) "to-u-f-i-a-x"
```

Move a UFIA (User File Information Area) from *a1* to *a2*, with the Plus D Memory paged in.

`>ufiax` is a common factor of `>ufia1` and `>ufia2`.

See also: `ufia`, `/ufia`, `ufia1`, `ufia2`.

Source file: <src/lib/dos.gplusdos.fs>.

>x

```
>x ( x -- ) ( X: -- x ) "to-x"
```

Move *x* from the data stack to the `xstack`.

See also: `x>`, `x@`.

Source file: <src/lib/data.xstack.fs>.

?

?

```
? ( a -- ) "question"
```

Display the 1-cell signed integer stored at *a*, using the format of `.`.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-94 (TOOLS), Forth-2012 (TOOLS).

See also: `c?`, `2?`, `@`.

Source file: <src/lib/memory.MISC.fs>.

?(

```
?( ( f "ccc<space><question><paren><space>" -- ) "question-paren"
```

If *f* is not zero, parse and discard until "?)" is found or until the end of the parse area is reached. `?(` cannot be used across blocks.

`?(` is used for conditional compilation, as a simpler but more compact alternative to the standard `[if]`.

`?(` is an `immediate` word.

Definition:

```
: ?( ( f "ccc<space><question><paren><space>" -- )
  0exit begin  parse-name dup
        while  s" ?)" str= ?exit repeat 2drop ; immediate
```

See also: `?\`, `?-->`, `(`.

Source file: <src/kernel.z80s>.

?)

```
?) ( -- ) "question-right-paren"
```

Do nothing. `?(` parses until `?)` is found.

`?)` is an `immediate` word.

Source file: <src/kernel.z80s>.

?-->

```
?--> ( f -- ) "question-next-block"
```

If *f* is not false, continue interpretation on the next sequential block. parse area. `?-->` is used for conditional compilation.

`?-->` is an `immediate` word.

See also: `-->`, `?(`, `?\`.

Source file: <src/lib/blocks.fs>.

??

```
?? "question-question"
  Compilation: ( "name" -- )
  Run-time:    ( f -- )
```

`??` is an `immediate` and `compile-only` word.

Compilation:

Parse *name* and search the current search order for it. If not found, `throw` an exception #-13. If found and it’s an `immediate` word, execute it, else compile it.

Run-time:

If *f* is not zero, execute *name*, which was compiled.

Source file: <src/lib/flow.MISC.fs>.

?\

```
?\ ( f "ccc<eol>" -- ) "question-backslash"
```

If *f* is not zero, execute `\`, else do nothing.

`?\` is an `immediate` word.

`?\` is a conditional version of `\`, used for conditional compilation, as a simpler but more compact alternative to the standard `[if]`.

Definition:

```
: ?\ ( "ccc<eol>" -- ) 0exit postpone \ ; immediate
```

See also: `?(`, `?-->`, `\`.

Source file: <src/kernel.z80s>.

?block-drive

```
?block-drive ( u -- ) "question-block-drive"
```

If *u* is `not-block-drive`, `throw` an exception #-35 ("invalid block number").

See also: `(>drive-block`, `block-drives`, `?drive#`, `?drives`.

Source file: <src/lib/dos.COMMON.fs>.

?branch

```
?branch ( f -- ) "question-branch"
```

A run-time procedure to branch conditionally. If *f* is not not zero, the following in-line address is copied to IP to branch forward or backward.

NOTE	This `?branch` is not Forth-83's `?branch`, which is equivalent to fig-Forth's `0branch`, which is a more logical name for the "branch if zero" action. Solo Forth borrows `0branch` from fig-Forth, and completes the branches after a logical naming convention shared with optional control flow words, e.g. `?exit`, `0exit`, `-exit`…

See also: `branch`, `-branch`, `+branch`.

Source file: <src/kernel.z80s>.

?c1-!

```
?c1-! ( ca - ) "question-c-one-minus-store"
```

If the character stored at *ca* is not zero, decrement it.

See also: `c1-!`, `c1+!`, `c-!`, `1-!`.

Source file: <src/lib/memory.MISC.fs>.

?call,

```
?call, ( a op -- ) "question-call-comma"
```

Compile a Z80 `assembler` conditional absolute-call instruction to address *a*, being *op* the identifier of the condition, which was put on the stack by `z?`, `nz?`, `c?`, `nc?`, `po?`, `pe?`, `p?`, or `m?`.

See also: `call,`, `?ret,`, `?jp,`.

Source file: <src/lib/assembler.fs>.

?ccase

```
?ccase "question-c-case"
  Compilation: ( C: -- orig )
  Run-time: ( c ca len --)
```

Start a `?ccase`..`end?ccase` structure. If *c* is in the string *ca len*, execute the n-th word compiled after `?ccase`, where *n* is the position of the first *c* in the string (0..len-1), then continue after `end?ccase`. If *c* is not in *ca len*, just continue after `end?ccase`.

`?ccase` is an `immediate` and `compile-only` word.

Usage example:

```
: .a   ( -- ) ." Letter A" ;
: .b   ( -- ) ." Letter B" ;
: .c   ( -- ) ." Letter C" ;

: letter ( c -- )
  s" abc" ?ccase  .a .b .c  end?ccase  ."  The End" cr ;
```

See also: `ccase`, `ccase0`.

Source file: <src/lib/flow.ccase.fs>.

?compiling

```
?compiling ( -- ) "question-compiling"
```

If not compiling, `throw` exception #-14 ("interpreting a compile-only word").

See also: `compile-only`, `?executing`.

Source file: <src/lib/exception.fs>.

?csp

```
?csp ( -- ) "question-c-s-p"
```

If the current data stack position does not match the value saved by `!csp`, `throw` an exception #-264 ("definition not finished").

Origin: fig-Forth.

Source file: <src/kernel.z80s>.

?defined

```
?defined ( f -- ) "question-defined"
```

If *f* is false, `throw` exception code #-13 (not found).

Source file: <src/kernel.z80s>.

?depth

```
?depth ( -- ) "question-depth"
```

If `depth` is not zero, set `base` to `decimal`, display the stack on a new line with `.s` and finally `throw` exception #-258 (stack imbalance).

See also: `?csp`.

Source file: <src/lib/tool.debug.MISC.fs>.

?dnegate

```
?dnegate ( d1 n -- d1|d2 ) "question-d-negate"
```

If *n* is negative, negate *d1*, giving its arithmetic inverse *d2*. Otherwise return *d1*.

`?dnegate` is written in Z80. Its equivalent definition in Forth is the following:

```
: ?dnegate ( d1 n -- d1|d2 ) 0< if dnegate then ;
```

Origin: fig-Forth's `d+-`.

See also: `dnegate`, `?negate`.

Source file: <src/kernel.z80s>.

?do

```
?do "question-do"
  Compilation: ( -- do-sys )
```

Compile `(?do` and leave *do-sys* to be consumed by `loop` or `+loop`.

`?do` is an `immediate` and `compile-only` word.

Definition:

```
: ?do ( -- do-sys )
  postpone (?do >mark ; immediate compile-only
```

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `do`, `-do`, `times`, `executions`.

Source file: <src/kernel.z80s>.

?drive#

```
?drive# ( u -- ) "question-drive-number-sign"
```

If *u* is greater than the maximum number of disk drives, `throw` an exception #-35 ("invalid block number").

See also: `(>drive-block`, `block-drives`, `?block-drive`, `?drives`.

Source file: <src/lib/dos.COMMON.fs>.

?drives

```
?drives ( n -- ) "question-drives"
```

If *n* is greater than the maximum number of disk drives, `throw` an exception #-287 ("wrong number of drives").

See also: `set-block-drives`. `?block-drive`, `?drive#`.

Source file: <src/lib/dos.COMMON.fs>.

?dup

```
?dup ( x -- 0 | x x ) "question-dup"
```

Duplicate *x* if it is non-zero.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `dup`, `0dup`, `-dup`.

Source file: <src/kernel.z80s>.

?esc-order

```
?esc-order ( n -- ) "question-esc-order"
```

Check if *n* is a valid size for the escaped strings search order, else `throw` an exception #-281 ("escaped strings search-order overflow").

See also: `#esc-order`, `esc-context`, `>esc-order`, `set-esc-order`, `get-esc-order`.

Source file: <src/lib/strings.escaped.fs>.

?executing

```
?executing ( -- ) "question-executing"
```

If not executing, `throw` exception #-263 ("execution only").

See also: `?compiling`.

Source file: <src/lib/exception.fs>.

?exit

```
?exit ( f -- ) ( R: nest-sys | -- nest-sys | ) "question-exit"
```

If *f* is zero, do nothing. Otherwise return control to the calling definition, specified by *nest-sys*.

`?exit` is the conditional version of `exit`.

`?exit` cannot be used within a `loop`. Use `if unloop exit then` instead.

`?exit` can be used in interpretation mode to stop the interpretation of a block.

See also: `exit`, `0exit`, `-exit`, `+exit`.

Source file: <src/kernel.z80s>.

?index-block

```
?index-block ( block -- ) "question-index-block"
```

Index block *block*, if not done before.

See also: `use-fly-index`.

Source file: <src/lib/blocks.indexer.fly.fs>.

?jp,

```
?jp, ( a op -- ) "question-j-p-comma"
```

Compile a Z80 `assembler` conditional absolute-jump instruction to the address *a*, being *op* the identifier of the condition, which was put on the stack by `z?`, `nz?`, `c?`, `nc?`, `po?`, `pe?`, `p?`, or `m?`.

See also: `jp,`, `?jr,`, `?ret,`, `?call,`.

Source file: <src/lib/assembler.fs>.

?jr,

```
?jr, ( a op -- ) "question-j-r-comma"
```

Compile a Z80 `assembler` conditional relative-jump instruction to address *a*, being *op* the identifier of the condition, which was put on the stack by `z?`, `nz?`, `c?`, or `nc?`.

See also: `jr,`, `?jp,`, `djnz,`, `jp>jr`, `(jr,`.

Source file: <src/lib/assembler.fs>.

?l#

```
?l# ( n -- ) "question-l-number-sign"
```

If `assembler` label *n* is out of range, `throw` exception #-283.

See also: `max-labels`.

Source file: <src/lib/assembler.labels.fs>.

?leave

```
?leave ( f -- ) ( R: loop-sys -- | loop-sys ) "question-leave"
```

If *f* is non-zero, discard the loop-control parameters for the current nesting level and continue execution immediately following the innermost syntactically enclosing `loop` or `+loop`.

See also: `0leave`, `leave`, `unloop`, `do`, `?do`.

Source file: <src/lib/flow.MISC.fs>.

?load

```
?load ( u f -- ) "question-load"
```

Load block *u* if flag *f* is true, else do nothing.

Origin: Pygmy Forth.

See also: `load`.

Source file: <src/lib/blocks.fs>.

?loading

```
?loading ( -- ) "question-loading"
```

If a `block` is not being loaded, i.e., if the content of `blk` is zero, `throw` exception code #-265 ("loading only").

See also: `loading?`, `load`.

Source file: <src/kernel.z80s>.

?located

```
?located ( n -- ) "question-located"
```

If *n* is zero, store `needed-word` into `parsed-name` (in order to make `needed-word` displayed) and `throw` an exception #-268 ("needed, but not located"). Otherwise do nothing.

Source file: <src/lib/002.need.fs>.

?negate

```
?negate ( n1 n2 -- n1|n3 ) "question-negate"
```

If *n2* is negative, negate *n1*, giving its arithmetic inverse *n3*. Otherwise return *n1*.

`?negate` is written in Z80. Its equivalent definition in Forth is the following:

```
: ?negate ( n1 n2 -- n1|n3 ) 0< if negate then ;
```

Origin: fig-Forth's `+-`.

See also: `negate`, `?dnegate`.

Source file: <src/kernel.z80s>.

?next-bank

```
?next-bank ( a -- a|a' ) "question-next-bank"
```

If the actual far-memory address *a* ($C000 .. $FFFF) has increased to the next bank ($0000 .. $3FFF), convert it to the corresponding actual address *a'* ($C000 .. $FFFF) in the next bank and page in the next bank. Otherwise return *a*.

See also: `?next-bank_`, `?previous-bank`.

Source file: <src/kernel.z80s>.

?next-bank_

```
?next-bank_ ( -- a ) "question-next-bank-underscore"
```

Address of the `question_next_bank` routine of the kernel, which does the following:

If the actual far-memory address ($C000..$FFFF) in the HL register has increased to the next bank ($0000..$3FFF), convert it to the corresponding actual address ($C000..$FFFF) in the next bank and page in the next bank, else do nothing.

This is the routine called by `?next-bank`. `?next-bank_` is used in `code` words.

Input:

- HL = address in a paged bank ($C000..$FFFF) or higher ($0000..$BFFF).

Output when HL is above the paged bank:

- HL = corresponding address in the next bank, which is paged in
- A corrupted
- D = 0
- E = bank

Output when HL is an address in a paged bank:

- HL preserved
- A corrupted

Source file: <src/lib/memory.far.fs>.

?order

```
?order ( n -- ) "question-order"
```

If *n* is not a valid size for the search order, `throw` an exception #-49 ("search-order overflow").

Definition:

```
: ?order ( n -- )
  dup 0< #-50 ?throw  max-order < ?exit  #-49 throw ;
```

See also: `#order`, `set-order`, `>order`, `order`.

Source file: <src/kernel.z80s>.

?os-unused

```
?os-unused ( u -- ) "question-o-s-unused"
```

If *u* is less than the the amount of unused space by the OS and the BASIC interpreter, `throw` exception code #-291 (out of OS memory).

See also: `os-unused`.

Source file: <src/lib/os.fs>.

?pairs

```
?pairs ( x1 x2 -- ) "question-pairs"
```

If *x1* not equals *x2* `throw` an exception #-22 (control structure mismatch).

Source file: <src/lib/compilation.fs>.

?previous-bank

```
?previous-bank ( a -- a|a' ) "question-previous-bank"
```

If the actual far-memory address *a* ($C000 .. $FFFF) has decreased to the previous bank ($8000 .. $BFFF), convert it to the corresponding actual address *a'* ($C000 .. $FFFF) in the previous bank and page in the next bank. Otherwise return *a*.

See also: `?previous-bank_`, `?next-bank`.

Source file: <src/kernel.z80s>.

?previous-bank_

```
?previous-bank_ ( -- a ) "question-previous-bank-underscore"
```

Address of the `question_previous_bank` routine of the kernel, which does the followig:

If the actual far-memory address ($C000..$FFFF) in the HL register has decreased to the previous bank ($8000..$BFFF), convert it to the corresponding actual address ($C000..$FFFF) in the previous bank and page in the next bank, else do nothing.

This is the routine called by `?previous-bank`. `?previous-bank_` is used in `code` words.

Input:

- HL = address in a paged bank ($C000..$FFFF) or lower ($8000..$BFFF).

Output when HL is below the paged bank:

- HL = corresponding address in the previous bank, which is paged in
- A corrupted
- D = 0
- E = bank

Output when HL is an address in a paged bank:

- HL preserved
- A corrupted

Source file: <src/lib/memory.far.fs>.

?rel

```
?rel ( n -- ) "question-rel"
```

If Z80 `assembler` relative branch *n* is too long, `throw` exception #-269 (relative jump too long).

Source file: <src/lib/assembler.fs>.

?repeat

```
?repeat "question-repeat"
  Compilation: ( dest -- dest )
  Run-time:    ( f -- )
```

An alternative exit point for `begin` … `until` loops: If *f* is non-zero, continue execution at `begin`, otherwise continue execution after `until`.

`?repeat` is an `immediate` and `compile-only` word.

Usage example:

```
: test ( -- )
    begin
      ...
    flag ?repeat  \ Go back to ``begin`` if flag is non-zero
      ...
    flag 0repeat  \ Go back to ``begin`` if flag is zero
      ...
    flag until    \ Go back to ``begin`` if flag is false
    ...
  ;
```

See also: `0repeat`.

Source file: <src/lib/flow.MISC.fs>.

?ret,

```
?ret, ( op -- ) "question-ret-comma"
```

Compile a Z80 `assembler` conditional return instruction, being *op* the identifier of the condition, which was put on the stack by `z?`, `nz?`, `c?`, `nc?`, `po?`, `pe?`, `p?`, or `m?`.

See also: `ret,`, `?jp,`, `?call,`.

Source file: <src/lib/assembler.fs>.

?retry

```
?retry "question-retry"
  Compilation: ( -- )
  Run-time:    ( f -- )
```

If *f* is zero, do nothing. Otherwise do a branch to the start of the word.

`?retry` is an `immediate` and `compile-only` word.

See also: `retry`, `?repeat`, `0repeat`.

Source file: <src/lib/flow.MISC.fs>.

?rstack

```
?rstack ( -- ) "question-r-stack"
```

`throw` an error if the return stack is out of bounds.

Origin: fig-Forth's `?stack`.

See also: `?stack`.

Source file: <src/kernel.z80s>.

?seconds

```
?seconds ( u -- ) "question-seconds"
```

Wait at least *u* seconds or until a key is pressed.

See also: `seconds`, `ms`, `?ticks-pause`.

Source file: <src/lib/time.fs>.

?set-drive

```
?set-drive ( c -- ior )
```

If drive *c* is not equal to the current default drive, returned by `get-drive`, use `set-drive` to make *c* the current default drive, returning I/O result code *ior*. Otherwise do nothing, and *ior* is zero.

`?set-drive` is used by `(>drive-block`, in order to update the current default drive only when needed, i.e. when the desired block is not in the current default drive.

Source file: <src/lib/dos.COMMON.fs>.

?set-tape-filename

```
?set-tape-filename ( ca len -- ) "question-set-tape-filename"
```

If filename *ca len* is not empty, store it into the tape header by executing `set-tape-filename`; else use a wildcard instead, by executing `any-tape-filename`.

Source file: <src/lib/tape.fs>.

?shift

```
?shift ( x1 n -- x1 | x2 ) "question-shift"
```

If *n* is zero, drop it and return *x1*. If *n* is negative, convert it to its absolute value and execute `rshift`, returning *x2*. If *n* is positive execute `lshift`, returning *x2*.

Source file: <src/lib/math.operators.1-cell.fs>.

?stack

```
?stack ( -- ) "question-stack"
```

`throw` an error if the data stack is out of bounds.

Origin: fig-Forth.

See also: `?rstack`.

Source file: <src/kernel.z80s>.

?stringer

```
?stringer ( len -- ) "question-stringer"
```

If *len* is greater than `/stringer`, then `throw` error #-293 (string too long). Otherwise do nothing.

`?stringer` is provided as an optional check. for `allocate-stringer`.

Source file: <src/lib/strings.MISC.fs>.

?throw

```
?throw ( f n -- ) "question-throw"
```

If *f* is non-zero, `throw` exception code *n*

Definition:

```
: ?throw ( f n -- ) swap if throw else drop then ;
```

Source file: <src/kernel.z80s>.

?ticks-pause

```
?ticks-pause ( u -- ) "question-ticks-pause"
```

Stop execution during at least *u* clock ticks, or until a key is pressed.

See also: `ticks-pause`, `basic-pause`, `?seconds`, `ticks/second`.

Source file: <src/lib/time.fs>.

?user

```
?user ( -- ) "question-user"
```

`throw` an exception if the user area pointer is out of bounds.

See also: `udp`, `/user`.

Source file: <src/lib/data.user.fs>.

?warn

```
?warn ( ca len -- ca len | ca len xt ) "question-warn"
```

Check if a warning about the redefinition of the word name *ca len* is needed. If no warning is needed, unnest the calling definition and return *ca len*. If a warning is needed, return *ca len* and the *xt* of the word found in the current compilation wordlist.

`?warn` is factor of `error-code-warn`, `message-warn` and `error-warn`.

See also: `no-warnings?`, `not-redefined?`, `message-warn`, `error-code-warn`, `error-warn`.

Source file: <src/lib/compilation.fs>.

?wcr

```
?wcr ( -- ) "question-w-c-r"
```

If the column cursor coordinate of the `current-window` is not zero, cause subsequent output to the current window appear at the beginning of the next line.

WARNING When the end of the `window` is reached, the cursor is set to the top left corner with `whome`. In a future version of the code, the window will be scrolled.

See also: `wcr`.

Source file: <src/lib/display.window.fs>.

@

@

```
@ ( a -- x ) "fetch"
```

x is the value stored at *a*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `!`, `2@`, `c@`.

Source file: <src/kernel.z80s>.

@+

```
@+ ( a -- a' x ) "fetch-plus"
```

Fetch *x* from *a*. Return *a'*, which is *a* incremented by one cell. This is handy for stepping through cell arrays.

See also: `@`, `2@+`, `c@+`.

Source file: <src/lib/memory.MISC.fs>.

@a

```
@a ( -- x ) "fetch-a"
```

Fetch cell *x* stored at the address register.

See also: `a`, `!a`.

Source file: <src/lib/memory.address_register.fs>.

@a+

```
@a+ ( -- x ) "fetch-a-plus"
```

Fetch cell *x* stored at the address register and increment the address register by one cell.

See also: `a`, `!a+`.

Source file: <src/lib/memory.address_register.fs>.

@bank

```
@bank ( a n -- x ) "fetch-bank"
```

Fetch *x* from address *a* ($C000..$FFFF) of `bank` *n*.

`@bank` is written in Z80. Its equivalent definition in Forth is the following:

```
: @bank ( a n -- x ) bank @ default-bank ;
```

See also: `!bank`, `c@bank`.

Source file: <src/lib/memory.far.fs>.

@bit

```
@bit ( b ca -- f ) "fetch-bit"
```

Fetch *f* from an element of a bit-array, represented by address *ca* and bitmask *b*.

`@bit` is an alias of `c@and?`.

See also: `!bit`, `bit-array`.

Source file: <src/lib/data.array.bit.fs>.

@dos

```
@dos ( a -- x ) "fetch-dos"
```

Fetch the cell *x* stored at Plus D memory address *a*.

See also: `!dos`, `c@dos`.

Source file: <src/lib/dos.gplusdos.fs>.

@dosvar

```
@dosvar ( -- )
```

Fetch the contents *x* of G+DOS variable *n*.

See also: `!dosvar`, `c@dosvar`, `!dos`.

Source file: <src/lib/dos.gplusdos.fs>.

@order

```
@order ( a -- )
```

Restore the search order stored at *a* by executing `nn@` and `set-order`.

`@order` is a useful factor of `unmarker`.

See also: `order,`, `@wordlists`.

Source file: <src/lib/tool.marker.fs>.

@p

```
@p ( a -- b ) "fetch-p"
```

Input byte *b* from port *a*.

See also: `!p`, `@`, `c@`.

Source file: <src/lib/memory.ports.fs>.

@sound

```
@sound ( b1 -- b2 ) "fetch-sound"
```

Get the contents *b2* of sound register *b1* (0…13).

See also: `!sound`, `sound`, `play`, `sound-register-port`.

Source file: <src/lib/sound.128.fs>.

@volume

```
@volume ( b1 -- b2 ) "fetch-volume"
```

Fetch *b2* from the volume register of channel *b1* (0..2, equivalent to notation 'A'..'C').

> Registers 8..10 (Channels A..C Volume)
>
> | | |
> |---|---|
> | **Bits 0-4** | Channel volume level. |
> | **Bit 5** | 1=Use envelope defined by register 13 and ignore the volume setting. |
> | **Bits 6-7** | Not used. |

See also: `!volume`, `@sound`.

Source file: <src/lib/sound.128.fs>.

@wordlists

```
@wordlists ( a -- ) "fetch-wordlists"
```

Fetch the `wordlist` definitions from *a*.

`@wordlists` is a factor of `unmarker`.

See also: `wordlists,`, `last-wordlist`, `@order`.

Source file: <src/lib/tool.marker.fs>.

[

[

```
[ ( -- ) "left-bracket"
```

Enter interpretation `state`.

`[` is an `immediate` word.

Definition:

```
: [ ( -- ) state off ; immediate
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `]`.

Source file: <src/kernel.z80s>.

['']

```
['']
  Compilation: ( "name" -- ) "bracket-tick-tick"
```

If *name* is found in the current search order, compile its execution-token pointer as a literal, else `throw` an exception.

`['']` is an `immediate` and `compile-only` word.

See also: `literal`, `''`, `[']`.

Source file: <src/lib/compilation.fs>.

[']

```
['] "bracket-tick"
  Compilation: ( "name" -- )
```

Compilation: If *name* is found in the current search order, compile its execution token as a literal, else `throw` an exception.

`[']` is an `immediate` and `compile-only` word.

Definition:

```
: ['] \ Compilation: ( "name" -- )
  ' postpone literal ; immediate
```

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `literal`, `'`, `['']`.

Source file: <src/kernel.z80s>.

[+switch

```
[+switch ( "name" -- a ) "bracket-plus-switch"
```

Open the `[switch` structure *name* to include additional clauses. The default behavior remains unchanged. The additions, like the original clauses, are terminated by `switch]`. Leave the head *a* of the given `[switch` *name*, for clauses to append to.

Origin: SwiftForth.

See also: `runs`, `run:`.

Source file: <src/lib/flow.bracket-switch.fs>.

[2const]

```
[2const] ( "name" -- ) "bracket-two-const"
```

Evaluate *name*. Then compile the double-cell value left on the stack.

`[2const]` is intented to compile double-cell constants as literals, in order to gain execution speed.

Usage example:

```
48. 2constant zx
: test ( -- ) [2const] zx d. ;
```

`[2const]` is an `immediate` and `compile-only` word.

See also: `2const`, `[const]`, `[xconst]`, `[cconst]`, `eval`.

Source file: <src/lib/compilation.fs>.

[:

```
[: "bracket-colon"
  Compilation: ( -- orig xt )
```

Start a quotation.

Suspend compiling to the current definition, start a new nested definition and compilation continues with this nested definition. Return *orig* and the execution token *xt* of of the host definition, both to be consumed by `;]`.

NOTE Locals are not supported yet.

`[:` is an `immediate` and `compile-only` word.

Source file: <src/lib/flow.bracket-colon.fs>.

[cconst]

```
[cconst] ( "name" -- ) "bracket-c-const"
```

Evaluate *name*. Then compile the char left on the stack.

`[cconst]` is intented to compile char constants as literals, in order to gain execution speed.

Usage example:

```
48 cconstant zx
: test ( -- ) [cconst] zx emit ;
```

`[cconst]` is an `immediate` and `compile-only` word.

See also: `cconst`, `[2const]`, `[const]`, `[xconst]`, `eval`.

Source file: <src/lib/compilation.fs>.

[char]

```
[char]
  Compilation: ( "name" -- )
  Run-time: ( -- c )
"bracket-char"
```

Compilation: ("name" —)

Parse *name* and append the run-time semantics given below to the current definition.

Run-time: (— c)

Place *c*, the value of the first character of *name*, on the stack.

`[char]` is an `immediate` and `compile-only` word.

Solo Forth recognizes the standard notation for characters, so `[char]` is not needed:

```
: test ( -- ) 'x' emit ."  equals " [char] x emit ;
```

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `char`.

Source file: <src/lib/parsing.fs>.

[comp']

```
[comp'] "bracket-comp-tick"
  Compilation: ( "name" -- )
  Run-time:    ( -- x xt )
```

Compilation token *x xt* represents the compilation semantics of *name*.

`[comp']` is an `immediate` and `compile-only` word.

Origin: Gforth.

See also: `comp'`, `'`.

Source file: <src/lib/compilation.fs>.

[compile]

```
[compile] ( "name" -- ) "bracket-compile"
```

Parse *name*. Find *name*. If *name* has other than default compilation semantics, append them to the current definition; otherwise append the execution semantics of *name*.

In other words: Force compilation of *name*. This allows compilation of an `immediate` word when it would otherwise have been executed.

`[compile]` is an `immediate` word.

`[compile]` has been be superseded by `postpone`.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE EXT), Forth-2012 (CORE EXT, obsolescent).

See also: `compile`, `compile,`.

Source file: <src/lib/compilation.fs>.

[const]

```
[const] ( "name" -- ) "bracket-const"
```

Evaluate *name*. Then compile the single-cell value left on the stack.

`[const]` is intented to compile constants as literals, in order to gain execution speed. *name* can be any word, as long as its execution returns a single-cell value on the stack.

Usage example:

```
48 constant zx
: test ( -- ) [const] zx . ;
```

`[const]` is an `immediate` and `compile-only` word.

See also: `const`, `[2const]`, `[xconst]`, `[cconst]`, `eval`.

Source file: <src/lib/compilation.fs>.

[defined]

```
[defined] ( "name" -- f ) "bracket-defined"
```

Parse *name*. Return a true flag if *name* is the name of a word that can be found in the current search order; else return a false flag.

`[defined]` is an `immediate` word.

Origin: Forth-2012 (TOOLS EXT).

See also: `defined`, `[undefined]`.

Source file: <src/lib/compilation.fs>.

[else]

```
[else] ( "ccc" -- ) "bracket-else"
```

Parse and discard space-delimited words from the parse area, including nested occurrences of `[if]` … `[then]`, and `[if]` … `[else]` … `[then]`, until either the word `[else]` the word `[then]` (case ignored) has been parsed and discarded. If the parse area becomes exhausted, it is refilled as with `refill`.

Origin: Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

See also: `[if]`.

Source file: <src/lib/compilation.fs>.

[false]

```
[false]  ( -- false ) "bracket-false"
```

`[false]` is an `immediate` word.

See also: `[true]`, `false`.

Source file: <src/lib/compilation.fs>.

[if]

```
[if] ( f "ccc" -- ) "bracket-if"
```

If *flag* is true, do nothing. Otherwise, parse and discard space-delimited words from the parse area, including nested occurrences of `[if] … [then]`, and `[if] … [else] … [then]`, until either the word `[else]` or the word `[then]` (case ignored) has been parsed and discarded. If the parse area becomes exhausted, it is refilled as with `refill`.

`[if]` is an `immediate` word.

Origin: Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

See also: `?\`, `?(`.

Source file: <src/lib/compilation.fs>.

[is]

```
[is] "bracket-is"
  Compilation: ( xt "name" -- )
  Run-time:    ( xt -- )
```

Compilation: ("name" —)

Append the run-time semantics given below to the current definition.

Run-time: (xt —)

Set *name*, which was defined by `defer`, to execute *xt*.

`[is]` is an `immediate` and `compile-only` factor of `is`.

Origin: Gforth.

See also: `<is>`.

Source file: <src/lib/define.deferred.fs>.

[switch

```
[switch ( "name1" "name2" -- a ) "bracket-switch"
```

Start the definition of a switch structure *name1* consisting of a linked list of single-precision numbers and associated behaviors, with its default action *name2*. The head *a* of the switch is left on the stack for defining clauses. The switch definition will be terminated by `switch]`, and can be extended by `[+switch`.

Usage example:

```
: one   ( -- )   ." unu " ;
: two   ( -- )   ." du "  ;
: three ( -- )   ." tri " ;
  \ clauses

: many  ( n -- ) . ." is too much! " ;
  \ default action

[switch .number many
  1 runs one  2 runs two  3 runs three  switch]

cr 1 .number 3 .number 4 .number

: four  ." kvar " ;

[+switch .number  4 runs four  switch]
  \ add a new clause for number 4

cr 1 .number 3 .number 4 .number

[+switch .number  5 run: ." kvin" ;  switch]
  \ add a new unnamed clause for number 5

cr 1 .number 4 .number 5 .number
```

NOTE `[switch` is the syntactic-sugar variant of `:switch`.

Origin: SwiftForth.

See also: `runs`, `run:`.

Source file: <src/lib/flow.bracket-switch.fs>.

[then]

```
[then] ( -- ) "bracket-then"
```

Do nothing. `[then]` is parsed and recognized by `[if]`.

`[then]` is an `immediate` word.

Origin: Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

Source file: <src/lib/compilation.fs>.

[true]

```
[true]  ( -- true ) "bracket-true"
```

`[true]` is an `immediate` word.

See also: `[false]`, `true`.

Source file: <src/lib/compilation.fs>.

[undefined]

```
[undefined] ( "name" -- f ) "bracket-undefined"
```

Parse *name*. Return a false flag if *name* is the name of a word that can be found in the current search order; else return a true flag.

`[undefined]` is an `immediate` word.

Origin: Forth-2012 (TOOLS EXT).

See also: `[defined]`.

Source file: <src/lib/compilation.fs>.

[xconst]

```
[xconst] ( "name" -- ) "bracket-x-const"
```

Evaluate *name*. Then compile the single-cell value left on the stack, using `xliteral`.

`[xconst]` is intented to compile constants as literals, when it's uncertain if the literal is a character or a cell, in order to gain execution speed. *name* can be any word, as long as its execution returns a single-cell value on the stack.

Usage example:

```
48 constant zx
: test ( -- ) [xconst] zx . ;
```

`[xconst]` is an `immediate` and `compile-only` word.

See also: `[2const]`, `[const]`, `[cconst]`, `eval`.

Source file: <src/lib/compilation.fs>.

\

\

```
\ ( "ccc<eol>" -- ) "backslash"
```

If `blk` contains zero, parse and discard the remainder of the parse area; otherwise parse and discard the portion of the parse area corresponding to the remainder of the current line.

`\` is an `immediate` word.

Definition:

```
: \ ( "ccc" -- )
  loading? if ->in/l parsed exit then #tib @ >in ! ;
```

Origin: Forth-94 (BLOCK EXT), Forth-2012 (BLOCK EXT).

See also: `(`, `->in/l`.

Source file: <src/kernel.z80s>.

]

]

```
] ( -- ) "right-bracket"
```

Enter compilation `state`.

Definition:

```
: ] ( -- ) state on ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `[`, `]l`, `]2l`, `]cl`, `]xl`.

Source file: <src/kernel.z80s>.

]2l

```
]2l ( xd -- ) "right-bracket-two-l"
```

A short form of the idiom `] 2literal`.

`]2l` is an `immediate` and `compile-only` word.

See also: `]`, `2literal`, `]l`, `]xl`, `]cl`.

Source file: <src/lib/compilation.fs>.

]cl

```
]cl ( x -- ) "right-bracket-c-l"
```

A short form of the idiom `] cliteral`.

`]cl` is an `immediate` and `compile-only` word.

See also: `]`, `cliteral`, `]2l`, `]l`, `]xl`.

Source file: <src/lib/compilation.fs>.

]l

```
]l ( x -- ) "right-bracket-l"
```

A short form of the idiom `] literal`.

`]l` is an `immediate` and `compile-only` word.

See also: `]`, `literal`, `]2l`, `]xl`, `]cl`.

Source file: <src/lib/compilation.fs>.

]options

```
]options ( a1 a2 a3 -- ) "right-bracket-options"
```

End a `options[ … ]options` structure. Resolve the addresses left by `options[`:

- a1 = address of exit point
- a2 = address of default option xt
- a3 = address of number of options

See `options[` for a usage example.

Source file: <src/lib/flow.options-bracket.fs>.

]xl

```
]xl ( x -- ) "right-bracket-x-l"
```

A short form of the idiom `] xliteral`.

`]xl` is an `immediate` and `compile-only` word.

See also: `]`, `xliteral`, `]2l`, `]l`, `]cl`.

Source file: <src/lib/compilation.fs>.

_

_mod

```
_mod ( n1 n2 -- n3 ) "underscore-mod"
```

Divide *n1* by *n2* (doing a floored division), giving the remainder *n3*.

See also: `/_mod`, `/`, `-rem`.

Source file: <src/lib/math.operators.1-cell.fs>.

a

a

```
a ( -- reg )
```

Return the identifier *reg* of the Z80 `assembler` register "A", which is interpreted as register pair "AF" by `assembler` words that use register pairs (for example `push,` and `pop,`).

See also: `b`, `c`, `d`, `e`, `h`, `l`, `m`, `ix`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

a

```
a ( -- a )
```

A `variable`. *a* is the address of a cell containing the address register.

See also: `a!`, `a@`, `!a`, `@a`, `c!a`, `c@a`, `!a+`, `@a+`, `c!a+`, `c@a+`.

Source file: <src/lib/memory.address_register.fs>.

a

```
a ( -- )
```

A command of `gforth-editor`: Go to marked position, marking the current position first.

See also: `m`, `h`, `d`, `f`, `r`.

Source file: <src/lib/prog.editor.gforth.fs>.

a!

```
a! ( a -- ) "a-store"
```

Set the address register.

See also: `a`, `a@`.

Source file: <src/lib/memory.address_register.fs>.

a@

```
a@ ( -- a ) "a-fetch"
```

Get the address register.

See also: `a`, `a!`.

Source file: <src/lib/memory.address_register.fs>.

aagain

```
aagain ( dest cs-id -- ) "a-again"
```

`aagain` is part of the `assembler` absolute-address control-flow structure `abegin` .. `aagain`.

See also: `ragain`.

Source file: <src/lib/assembler.fs>.

abase

```
abase ( -- a ) "a-base"
```

A `variable`. *a* is the address of a cell where the current value of `base` is preserved by `asm`.

Source file: <src/kernel.z80s>.

abegin

```
abegin ( -- dest cs-id ) "a-begin"
```

`abegin` is part of the `assembler` absolute-address control-flow structure `abegin` .. `awhile` .. `arepeat`.

See also: `rbegin`.

Source file: <src/lib/assembler.fs>.

abort

```
abort ( -- )
```

Empty the data stack and perform the function of `quit`, which includes emptying the return stack, without displaying a message.

Definition:

```
abort ( -- ) -1 throw ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE, EXCEPTION EXT), Forth-2012 (CORE, EXCEPTION EXT).

See also: `(abort`, `abort"`, `throw`, `error`.

Source file: <src/kernel.z80s>.

abort"

```
abort" "abort-quote"
  Compilation: ( "ccc<quote>" -- )
  Run-time:    ( x -- )
```

Compile `(abort"`, parse *ccc* delimited by a double quote and compile it.

`abort"` is an `immediate` and `compile-only` word.

Origin: Forth-79 (Reference Word Set), Forth-83 (Required Word Set), Forth-94 (EXCEPTION EXT), Forth-2012 (EXCEPTION EXT).

See also: `abort-message`, `abort`, `throw`, `warning"`.

Source file: <src/lib/exception.fs>.

abort-message

```
abort-message ( -- a )
```

A `2variable`. *a* is the address of a cell pair containing the address and length of the `abort"` message.

Source file: <src/kernel.z80s>.

aborted?

```
aborted? ( c -- f ) "aborted-question"
```

If no key is pressed return `false`. If a key is pressed, discard it and wait for a second key. Then return `true` if it's *c*, else return `false`.

`aborted?` is a useful factor of `nuf?`.

Usage example:

```
: listing ( -- )
  begin  ." bla "  bl aborted?  until  ." Aborted" ;
```

Source file: <src/lib/keyboard.MISC.fs>.

abs

```
abs ( n -- u )
```

Leave the absolute value *u* of a number *n*.

Definition:

```
: abs ( n -- u ) dup ?negate ;
```

Source file: <src/kernel.z80s>.

acat

```
acat ( -- ) "a-cat"
```

Show an abbreviated disk catalogue of the current drive.

See also: `cat`, `wcat`, `wacat`, `(cat`, `set-drive`.

Source file: <src/lib/dos.gplusdos.fs>.

accept

```
accept ( ca1 len1 -- len2 )
```

Receive a string of at most *len1* characters. No characters are received or transferred if *len1* is zero. Display graphic characters as they are received.

Input terminates when an implementation-defined line terminator is received. When input terminates, nothing is appended to the string or displayed on the screen.

len2 is the length of the string stored at *ca1*.

In Solo Forth `accept` is a deferred word (see `defer`). Its default action is `simple-accept`, which provides only the basic editing options. Alternative definitions are provided in the library.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

Source file: <src/kernel.z80s>.

action-of

```
action-of ( -- )
  Interpretation: ( "name" -- xt )
  Compilation:    ( "name" -- )
  Run-time:       ( -- xt )
```

Interpretation

Parse *name*, which is a word defined by `defer`. Return *xt*, which is the execution token that name is set to execute.

Compilation

Parse *name*, which is a word defined by `defer`. Append the runtime semantics given below to the current definition.

Runtime

Return *xt*, which is the execution token that name is set to execute.

`action-of` is an `immediate` word.

Origin: Forth-2012 (CORE EXT).

See also: `defer@`, `defers`.

Source file: <src/lib/define.deferred.fs>.

actions-table

```
actions-table ( -- a )
```

A `variable`, *a* is the address of a cell containing the address of a cell array which holds the execution tokens of the current `menu` options. `actions-table` is set by `set-menu`.

See also: `options-table`.

Source file: <src/lib/menu.sinclair.fs>.

adc#,

```
adc#, ( b -- ) "a-d-c-number-sign-comma"
```

Compile the Z80 `assembler` instruction `ADC A,b`.

Source file: <src/lib/assembler.fs>.

adc,

```
adc, ( reg -- ) "a-d-c-comma"
```

Compile the Z80 `assembler` instruction `ADC reg`.

See also: `add,`, `sub,`, `sbc,`, `addp,`.

Source file: <src/lib/assembler.fs>.

adcp,

```
adcp, ( regp1 regp2 -- ) "a-d-c-p-comma"
```

Compile the Z80 `assembler` instruction `ADC regp2,regp1`.

See also: `adcp,`.

Source file: <src/lib/assembler.fs>.

adcx,

```
adcx, ( disp regpi --  ) "a-d-c-x-comma"
```

Compile the Z80 `assembler` instruction `ADC A,(regpi+disp)`.

See also: `addx,`, `sbcx,`.

Source file: <src/lib/assembler.fs>.

add#,

```
add#, ( b -- ) "add-number-sign-comma,"
```

Compile the Z80 `assembler` instruction `ADD A,b`.

Source file: <src/lib/assembler.fs>.

add,

```
add, ( reg -- ) "add-comma"
```

Compile the Z80 `assembler` instruction `ADD reg`.

See also: `sub,`, `sbc,`, `addp,`.

Source file: <src/lib/assembler.fs>.

addix,

```
addix, ( regp -- ) "add-i-x-comma"
```

Compile the Z80 `assembler` instruction `ADD IX,regp`.

See also: `addiy,`, `addp,`.

Source file: <src/lib/assembler.fs>.

addiy,

```
addiy, ( regp -- ) "add-i-y-comma"
```

Compile the Z80 `assembler` instruction `ADD IY,regp`.

See also: `addiy,`, `addp,`.

Source file: <src/lib/assembler.fs>.

addp,

```
addp, ( regp -- ) "add-p-comma"
```

Compile the Z80 `assembler` instruction `ADD HL,regp`.

See also: `add,`.

Source file: <src/lib/assembler.fs>.

address-unit-bits

```
address-unit-bits ( -- n )
```

n is the size of one address unit, in bits.

See also: `max-char`, `environment?`.

Source file: <src/lib/environment-question.fs>.

addx,

```
addx, ( disp regpi -- ) "add-x-comma"
```

Compile the Z80 `assembler` instruction `ADD A,(regpi+disp)`.

See also: `adcx,`, `subx,`.

Source file: <src/lib/assembler.fs>.

adraw176

```
adraw176 ( gx gy -- ) "a-draw-176"
```

Draw a line from the current coordinates to the given absolute coordinates *gx gy*, using only the top 176 pixel rows of the screen (the lower 16 pixel rows are not used). *gx* is 0..255; *gy* is 0..175.

See also: `rdraw176`.

Source file: <src/lib/graphics.lines.fs>.

aelse

```
aelse ( orig1 cs-id -- orig2 cs-id ) "a-else"
```

Check the Z80 `assembler` control-flow structure identifier *cs_id*, and resolve the forward reference *orig1*, both left by `aif`; then compile a Z80 `assembler` unconditional absolute-address jump, putting its unresolved forward reference *orig2* and control-flow structure identifier *cs-id* on the stack, to be resolved by `athen`.

Also put the location of a new unresolved forward reference *orig2* and the control-structure identifier *cs_id* onto the stack, to be consumed by `athen`.

`aelse` is part of the `assembler` absolute-address control-flow structure `aif` .. `aelse` .. `athen`, equivalent to Forth `if` .. `else` .. `then`.

See also: `relse`, `?pairs`, `(aif`.

Source file: <src/lib/assembler.fs>.

again

```
again
  Compilation: ( C: dest -- )
  Run-time:    ( -- )
```

Compilation: Compile an unconditional `branch` to the backward reference *dest*, usually left by `begin`.

Run-time: Continue execution at the location specified by *dest*.

`again` is an `immediate` and `compile-only` word.

Definition:

```
: again \ Compilation: ( C: dest -- )
        \ Run-time:    ( -- )
  compile branch <resolve ; immediate compile-only
```

Origin: fig-Forth, Forth-79 (Reference Word Set), Forth-83 (Uncontrolled Reference Words), Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `until`, `repeat`.

Source file: <src/kernel.z80s>.

ahead

```
ahead
  Compilation: ( C: -- orig )
  Run-time:    ( -- )
```

Compilation: Compile an unconditional `branch` and put the location *orig* of its unresolved destination on the control-flow stack.

Run-time: Continue execution at the location specified by the resolution of *orig*.

`ahead` is an `immediate` and `compile-only` word.

Definition:

```
: ahead \ Compilation: ( C: -- orig )
        \ Run-time:    ( -- )
  compile branch >mark ; immediate compile-only
```

Origin: Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

Source file: <src/kernel.z80s>.

aif

```
aif ( op -- orig cs-id ) "a-if"
```

Compile the Z80 `assembler` absolute-jump instruction *op* and put the location of a new unresolved forward reference *orig* and the control-structure identifier *cs_id* onto the stack, to be consumed by `aelse` or `athen`.

op was left by any of the following `assembler` conditions: `nz?`, `z?`, `nc?`, `c?`, `po?`, `pe?`, `p?`, `m?`.

`aif` is part of the `assembler` absolute-address control-flow structure `aif` .. `aelse` .. `athen`, equivalent to Forth `if` .. `else` .. `then`.

See also: `rif`, `(aif`, `inverse-cond`.

Source file: <src/lib/assembler.fs>.

al#

```
al# ( -- ) "a-l-number-sign"
```

Create an absolute reference to an `assembler` label defined by `l:`. The label number has been compiled in the last cell of the latest Z80 instruction. If the corresponding label is already defined, its value is patched into the latest Z80 instruction. Otherwise it will be patched when the label is defined by `l:`.

Usage example:

```
code my-code ( -- )
  #2 call, al#  \ a call to label #2
  nop,
  #2 l:         \ definition of label #2
  ret,
end-code
```

WARNING	`al#` is used after the Z80 command, while its counterpart `rl#` is used before the Z80 command.

Source file: <src/lib/assembler.labels.fs>.

al-id

```
al-id ( -- b ) "a-l-i-d"
```

b is the identifier of absolute references created by `al#`. `al-id` is used as a bitmask added to the `assembler` label number stored in `l-refs`.

See also: `rl-id`.

Source file: <src/lib/assembler.labels.fs>.

alias

```
alias ( xt "name" -- )
```

Create an alias *name* that will execute *xt*.

Aliases have the execution token *xt* of the original word, but, contrary to synonyms created by `synonym`, don’t inherit its attributes (`immediate` and `compile-only`).

See `realias`, `alias!`, `synonym`.

Origin: Gforth.

Source file: <src/lib/define.alias.fs>.

alias!

```
alias! ( xt nt -- ) "alias-store"
```

Set the alias *nt* to execute *xt*.

See `alias`, `realias`.

Source file: <src/lib/define.alias.fs>.

align

```
align ( -- )
```

If the data-space pointer is not aligned, reserve enough space to align it.

In Solo Forth, `align` does nothing (it’s an `immediate` alias of `noop`).

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `dp`, `aligned`.

Source file: <src/lib/memory.MISC.fs>.

aligned

```
aligned ( a1 -- a2 )
```

a2 is the first aligned address greater than or equal to *a1*.

In Solo Forth, `aligned` does nothing (it’s an `immediate` alias of `noop`).

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `align`.

Source file: <src/lib/memory.MISC.fs>.

aline176

```
aline176 ( gx gy -- ) "a-line-176"
```

Draw a line from the current coordinates to the given absolute coordinates *gx gy*, using only the top 176 pixel rows of the screen (the lower 16 pixel rows are not used) and preserving the current attributes of the screen. *gx* is 0..255; *gy* is 0..175.

`aline176` is faster than `adraw176`.

See also: `rdraw176`.

Source file: <src/lib/graphics.lines.fs>.

allocate

```
allocate ( u -- a ior )
```

Allocate *u* bytes of contiguous data space. The data-space pointer is unaffected by this operation. The initial content of the allocated space is undefined.

If the allocation succeeds, *a* is the starting address of the allocated space and *ior* is zero.

If the operation fails, *a* does not represent a valid address and *ior* is the I/O result code.

`allocate` is a deferred word (see `defer`) whose action can be `charlton-allocate` or `gil-allocate`, depending on the `heap` implementation used by the application.

Origin: Forth-94 (MEMORY), Forth-2012 (MEMORY).

See also: `free`, `resize`, `empty-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

allocate-stringer

```
allocate-stringer ( len -- ca )
```

Allocate *len* characters in the `stringer` and return the address *ca* of the allocated space. If *len* is greater than `unused-stringer`, `empty-stringer` is executed, no check is done whether *len* is greater than `/stringer` (the maximum capacity of the buffer).

Definition:

```
: allocate-stringer ( len -- ca )
  fit-stringer stringer unused-stringer + ;
```

See also: `>stringer`.

Source file: <src/kernel.z80s>.

allocate-xstack

```
allocate-xstack ( n -- a ) "allocate-x-stack"
```

Create an `xstack` in the heap. *n* is the size in cells. Return its address *a*.

See also: `xfree`, `allocate-xstack`.

Source file: <src/lib/data.xstack.fs>.

allot

```
allot ( n -- )
```

If *n* is greater than zero, reserve *n* bytes of data space. If *n* is less than zero, release *n* bytes of data space. If *n* is zero, leave the data-space pointer unchanged.

Definition:

```
: allot ( n -- ) dp +! ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `allotted`, `dp`, here`, `reserve`.

Source file: <src/kernel.z80s>.

allot-heap

```
allot-heap ( n -- a )
```

Create a `heap` of *n* bytes in the data space. Return its address *a*.

See also: `limit-heap`, `bank-heap`, `farlimit-heap`, `empty-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

allot-xstack

```
allot-xstack ( n -- a ) "allot-x-stack"
```

Create an `xstack` in data space. *n* is the size in cells. Return its address *a*.

See also: `allocate-xstack`.

Source file: <src/lib/data.xstack.fs>.

allotted

```
allotted ( n -- a )
```

Reserve *n* bytes of data space and return its address *a*.

See also: `reserve`, `buffer:`, `allot`, `here`.

Source file: <src/lib/memory.MISC.fs>.

also

```
also ( -- )
```

Duplicate the word list at the top of the search order.

Definition:

```
: also ( -- ) get-order over swap 1+ set-order ;
```

Origin: Forth-83 (Experimental proposals), Forth-94 (SEARCH EXT), Forth-2012 (SEARCH-EXT).

See also: `previous`, `get-order`, `set-order`, `only`, `order`, `>order`.

Source file: <src/kernel.z80s>.

ambulance

```
ambulance ( n -- )
```

Ambulance sound for ZX Spectrum 48. Make it *n* times.

Source file: <src/lib/sound.48.fs>.

and

```
and ( x1 x2 -- x3 )
```

x3 is the bit-by-bit logical "and" of *x1* with *x2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `or`, `xor`, `negate`, `0=`, `dand`.

Source file: <src/kernel.z80s>.

and#,

```
and#, ( b -- ) "and-number-sign-comma"
```

Compile the Z80 `assembler` instruction `AND b`.

See also: `or#,`, `xor#,`, `sub#,`.

Source file: <src/lib/assembler.fs>.

and,

```
and, ( reg -- ) "and-comma"
```

Compile the Z80 `assembler` instruction `AND reg`.

See also: `xor,`, `or,`.

Source file: <src/lib/assembler.fs>.

andif

```
andif "and-if"
  Compilation: ( C: -- orig )
  Run-time:    ( f -- )
```

Short-circuit `and` variant of `if`.

`andif` is an `immediate` and `compile-only` word.

Usage example:

```
: the-end? ( -- f ) cond  won-battle?     andif
                          found-treasure? andif
                          kill-dragon?    andif
                      thens ;
```

Compare with the following equivalent definition, where all three conditions are always checked:

```
: the-end? ( -- f ) won-battle?
                    found-treasure? and
                    kill-dragon?    and ;
```

See also: `orif`, `cond`, `thens`.

Source file: <src/lib/flow.MISC.fs>.

andx,

```
andx, ( disp regpi --  ) "and-x-comma"
```

Compile the Z80 `assembler` instruction `AND (regpi+disp)`.

See also: `xorx,`, `orx,`, `cpx,`.

Source file: <src/lib/assembler.fs>.

anew

```
anew ( "name" -- )
```

Parse *name*. If *name* is the name of a word in the current search order, execute it. Then restore `>in` to its value previous to the parsing of *name* and execute `marker`.

The function of `anew` is to execute a *name* already created by `marker` and then create it again.

See also: `possibly`.

Source file: <src/lib/tool.marker.fs>.

anon

```
anon
  Compilation:  ( n -- )
  Run-time:     ( -- a )
```

`anon` is an `immediate` and `compile-only` word.

Compilation:

Compile a reference to cell *n* (0 index) of the buffer pointed by `anon>` and initialized by `set-anon`.

Run-time:

Return address *a* of cell *n* (0 index) of the buffer that was pointed by `anon>` during the compilation.

See `set-anon` for a usage example.

See also: `arguments`, `local`.

Source file: <src/lib/locals.anon.fs>.

anon>

```
anon>  ( -- a ) "anon-to"
```

A `variable`. *a* contains the address of the buffer used by local variables defined by `set-anon` and accessed by `anon`.

`anon>` must be set by the application before compiling a word that uses `set-anon` and `anon`. One single buffer pointed by `anon>` can be shared by several words, provided they dont't need to use it at the same time, e.g. because of nesting.

Source file: <src/lib/locals.anon.fs>.

any-of

```
any-of
  Compilation: ( C: -- of-sys )
  Run-time:    ( x#0 x#1 ... x#n n -- | x#0 )
```

A variant of `of`.

Compilation:

Put *of-sys* onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *of-sys*, such as `endof`.

Run-time:

If *x#0* equals any of *x#1* … *x#n*, discard *x#1* … *x#n n* and continue execution at the location specified by the consumer of *of-sys*, e.g., following the next `endof`. Otherwise, consume also *x0* and continue execution in line.

`any-of` is an `immediate` and `compile-only` word.

Usage example:

```
: test ( n -- )
  case
    1 of             ." one"               endof
    2 7 10 3 any-of ." two, seven or ten" endof
    6 of             ." six"               endof
  endcase ;
```

See also: `case`, `or-of`, `(any-of`.

Source file: <src/lib/flow.case.fs>.

any-tape-filename

```
any-tape-filename ( -- )
```

Configure `tape-header` to load any filename, by replacing the first char of `tape-filename` with 255, which will be recognized as a wild card.

Source file: <src/lib/tape.fs>.

any?

```
any? ( x[0] x[1]..x[n] n -- f ) "any-question"
```

Is any *x[1]..x[n]* equal to *x[0]*?

Origin: John A. Peters' tools for CP/M F83 2.1.1, 1984.

See also: `either`, `neither`, `ifelse`.

Source file: <src/lib/math.operators.1-cell.fs>.

arepeat

```
arepeat ( dest cs-id1 orig cs-id2 ) "a-repeat"
```

`arepeat` is part of the `assembler` absolute-address control-flow structure `abegin` .. `awhile` .. `arepeat`.

See also: `rrepeat`.

Source file: <src/lib/assembler.fs>.

arg-action

```
arg-action ( -- a )
```

A `variable`. *a* holds the execution token of the action performed by the locals defined by `arguments`. Its default value is stored in `arg-default-action`. The content of `arg-default-action` is copied to `arg-action` by `arguments`, and also every time a local variable is used.

Source file: <src/lib/locals.arguments.fs>.

arg-default-action

```
arg-default-action ( -- a )
```

A `variable`. *a* holds the execution token of the default action performed by the locals defined by `arguments`. Its default value is zero, which means "no action" (`noop` can be used too, but `arg-default-action off` is simpler than `' noop arg-defaul-action !`).

`toarg` and `+toarg` change the content of `arg-default-action`.

The content of `arg-default-action` is copied to `arg-action` by `arguments`, and also every time a local variable is used.

See also: `arg-action`.

Source file: <src/lib/locals.arguments.fs>.

arguments

```
arguments ( i*x +n -- j*x )
```

Define the number *+n* of arguments to take from the stack and assign them to the first local variables from `l0` to `l9`. By default, local variables are manipulated with `@`, `!` and `+!`, like ordinary variables. They are returned with `results`.

Example: The phrase `3 arguments` assigns the names of local variables `l0` through `l9` to ten stack positions, with `l0`, `l1` and `l2` returning the address of the top 3 stack values that were there before `3 arguments` was executed. `l3` through `l9` are zero-filled and the stack pointer is set to just below `l9`. After all calculating is done, the phrase `3 results` leaves that many results on the stack relative to the stack position when `arguments` was executed. All intermediate stack values are lost, which is good because you can leave the stack "dirty" and it doesn’t matter.

Usage example:

```
: test ( length width height -- length' volume surface )
  3 arguments
  l0 @ l1 @ * l5 !       \ surface
  l5 @ l2 @ * l4 !       \ volume
  $2000 l0 +!            \ length+$2000
  l4 @ l1 !              \ volume
  l5 @ l2 !              \ surface
  3 results ;
```

When `toarg` or `+toarg` are loaded, they change the default behaviour of locals: Then `l0` through `l9` return their contents, not their addresses. To write them you precede them with the word `toarg`. For example `5 toarg l4` writes a 5 into `l4`. Execution of `l4` returns 5 to the stack. To add a number to a local variable, you precede it with the word `+toarg`. For example, `5 +toarg l4` adds 5 to the current content of `l4`.

Example:

```
need toarg  need +toarg

: test ( length width height -- length' volume surface )
  3 arguments
  l0 l1 * toarg l5       \ surface
  l5 l2 * toarg l4       \ volume
  $2000 +toarg l0        \ add $2000 to length
  l4 toarg l1            \ volume
  l5 toarg l2            \ surface
  3 results ;
```

The default action of local variables (either return its address or its value) is hold in `arg-default-action`, as an execution token.

`arguments` is a `compile-only` word.

See also: `local`, `anon`.

Source file: <src/lib/locals.arguments.fs>.

array<

```
array< ( a1 n -- a2 ) "array-from"
```

Return address *a2* of element *n* of a 1-dimension single-cell array *a1*.

`array<` is written in Z80. Its equivalent definition in Forth is the following:

```
: array< ( a1 n -- a2 ) cells + ;
```

See also: `array>`, `+perform`.

Source file: <src/lib/data.array.COMMON.fs>.

array>

```
array> ( n a1 -- a2 ) "array-to"
```

Return address *a2* of element *n* of a 1-dimension single-cell array *a1*. `array>` is a common factor of `avalue` and `avariable`.

`array>` is written in Z80. Its equivalent definition in Forth is the following:

```
: array> ( n a1 -- a2 ) swap cells + ;
```

See also: `2array>`, `array<`, `+perform`.

Source file: <src/lib/data.array.COMMON.fs>.

array>items

```
array>items ( a -- n ) "array-to-items"
```

Convert address of array *a* to its number of items *n*.

See also: `1array`.

Source file: <src/lib/data.array.noble.fs>.

ascii-char?

```
ascii-char? ( c -- f ) "ascii-char-question"
```

Is character *c* an ASCII character, i.e. in the range 0..126?

See also: `graphic-ascii-char?`, `control-char?`.

Source file: <src/lib/chars.fs>.

ascii-ocr

```
ascii-ocr ( -- ) "ascii-o-c-r"
```

Set `ocr` to work with the current ASCII charset, pointed by `os-chars`.

See also: `ocr-font`, `ocr-first`, `ocr-chars`, `udg-ocr`, `set-font`.

Source file: <src/lib/graphics.ocr.fs>.

asm

```
asm ( -- )
```

Enter the `assembler` mode. `asm` is executed by `code` and `;code`.

Definition:

```
: asm ( -- )
  !csp init-asm base @ abase ! hex assembler-wordlist >order ;
```

See also: `end-asm`, `init-asm`, `abase`, `!csp`, `hex`.

Source file: <src/kernel.z80s>.

assembler

```
assembler ( -- )
```

Replace the first word list in the search order with `assembler-wordlist`, which contains the assembler words (see the main ones in section Z80 instructions).

`need assembler` will load the assembler from the library, except the absolute-jump control-flow structures (`aif`, `athen`, `aelse`, `abegin`, `awhile`, `auntil`, `aagain`, `arepeat`), labels (`l:`, `rl#`, `al#`, etc.) macros (`macro`, `endm`) and some specific words (`execute-hl,`, `call-xt,`, `hook,`, `prt,`).

Origin: Forth-79 (Assembler Word Set), Forth-83 (Assembler Extension Word Set), Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

Source file: <src/lib/assembler.fs>.

assembler-wordlist

```
assembler-wordlist ( -- wid )
```

Return *wid*, the identifier of the word list that includes the words defined as part of the `assembler` (see the main ones in section Z80 instructions).

See also: `wordlist`, `set-order`, `forth-wordlist`, `root-wordlist`.

Source file: <src/kernel.z80s>.

assert(

```
assert( ( -- ) "assert-paren"
```

Start a normal assertion. Normal assertion are turned on by default. `assert(` is equivalent to `assert1(`.

`assert(` is an `immediate` word.

Origin: Gforth.

See also: `assert-level`, `assert0(`, `assert1(`, `assert2(`, `assert3(`, `)`.

Source file: <src/lib/tool.debug.assert.fs>.

assert-level

```
assert-level ( -- a )
```

A `variable`. *a* is the address of a cell containing the highest assertions that are turned on (0..3). Its default value is 1: all assertions above 1 are turned off.

Origin: Gforth.

See also: `assert(`.

Source file: <src/lib/tool.debug.assert.fs>.

assert0(

```
assert0( ( -- ) "assert-zero"
```

Start an important assertion. Important assertions should always be turned on.

`assert0(` is an `immediate` word.

Origin: Gforth.

See also: `assert-level`, `assert(`, `assert1(`, `assert2(`, `assert3(`, `)`.

Source file: <src/lib/tool.debug.assert.fs>.

assert1(

```
assert1( ( -- ) "assert-one"
```

Start a normal assertion. Normal assertions are turned on by default.

`assert1(` is an `immediate` word.

Origin: Gforth.

See also: `assert-level`, `assert(`, `assert0(`, `assert2(`, `assert3(`, `)`.

Source file: <src/lib/tool.debug.assert.fs>.

assert2(

```
assert2( ( -- ) "assert-two"
```

Start a debugging assertion.

`assert2(` is an `immediate` word.

Origin: Gforth.

See also: `assert-level`, `assert(`, `assert0(`, `assert1(`, `assert3(`, `)`.

Source file: <src/lib/tool.debug.assert.fs>.

assert3(

```
assert3( ( -- ) "assert-three"
```

Start a slow assertion. Slow assertions are those you may not want to turn on in normal debugging; you would turn them on mainly for thorough checking.

`assert3(` is an `immediate` word.

Origin: Gforth.

See also: `assert-level`, `assert(`, `assert0(`, `assert1(`, `assert2(`, `)`.

Source file: <src/lib/tool.debug.assert.fs>.

assertn

```
assertn ( n -- ) "assert-n"
```

If the contents of `assert-level` is greater than *n*, then parse and discard the input stream to the next right paren (the end of the assertion); else do nothing. `assertn` is the common factor of `assert0(`, `assert1(`, `assert2(`, and `assert3(`.

Origin: Gforth.

See also: `assert(`.

Source file: <src/lib/tool.debug.assert.fs>.

associative-case:

```
associative-case: ( "name" -- ) "associative-case-colon"
```

Create an associative case definition "name": `name ( i*x n -- j*x )`.

Usage example:

```
: red       ." red" ;
: blue      ." blue" ;
: orange    ." orange" ;
: pink      ." pink" ;
: black     ." black" ;

associative-case: color ( n -- )
  7 red  12 blue  472 orange  15 pink  0 black ;

7 color cr  472 color cr  3000 color cr
```

n for default must be 0 and the default pair must be last. Numbers can be in any order except 0 must be last. An actual zero or a no match causes the default to be executed. Numbers can't be constants.

See also: `associative:`, `associative-list`.

Source file: <src/lib/flow.associative-case-colon.fs>.

associative-list

```
associative-list ( "name" -- )
```

Create a new associative list "name".

See also: `entry:`, `centry:`, `2entry:`, `sentry:`, `item`, `item?`, `items`, `associative:`, `associative-case:`.

Source file: <src/lib/data.associative-list.fs>.

associative:

```
associative: ( n "name" -- ) "associative-colon"
```

Create a table lookup *name* with *n* entries.

An associative memory word. It must be followed by a set of values to be looked up. At runtime, the values stored in the data field are searched for a match. If a match is made, the index to that value is returned. If no match is made, then the number of entries is returned. This is the inverse of an array.

Usage example:

```
1000 constant zx1
200 constant zx2
30 constant zx3

3 associative: unzx ( value -- n ) zx1 , zx2 , zx3 ,

1000 unzx .  \ prints 0
200 unzx .   \ prints 1
30 unzx .    \ prints 2
```

See also: `associative-list`, `associative-case:`.

Source file: <src/lib/data.associative-colon.fs>.

at-wxy

```
at-wxy ( -- ) "at-w-x-y"
```

Set the cursor coordinates to the `current-window` cursor coordinates.

See also: `wat-xy`, `at-xy`.

Source file: <src/lib/display.window.fs>.

at-x

```
at-x ( col -- )
```

Set the cursor at the given column (x coordinate) *col* and the current row (y coordinate).

See also: `at-y`, `at-xy`, `row`, `column`.

Source file: <src/lib/display.cursor.fs>.

at-xy

```
at-xy ( col row -- ) "at-x-y"
```

Set the cursor coordinates to column *col* and row *row*. The upper left corner is column zero, row zero.

`at-xy` is a deferred word (see `defer`) whose default action is `mode-32-at-xy`.

Origin: Forth-94 (FACILITY), Forth-2012 (FACILITY).

See also: `home`.

Source file: <src/kernel.z80s>.

at-xy-display-udg

```
at-xy-display-udg ( c col row -- ) "at-x-y-display-u-d-g"
```

Display UDG *c* at cursor coordinates *col row*. `at-xy-display-udg` is much faster than using `at-xy` and `emit-udg`, because no ROM routine is used, the cursor coordinates are not updated and the screen attributtes are not changed (only the character bitmap is displayed).

See also: `udg-at-xy-display`.

Source file: <src/lib/graphics.udg.fs>.

at-y

```
at-y ( row -- )
```

Set the cursor at the current column (x coordinate) and the given row (y coordinate) *row*.

See also: `at-x`, `at-xy`, `row`, `column`.

Source file: <src/lib/display.cursor.fs>.

athen

```
athen ( orig cs-id -- ) "a-then"
```

Check the `assembler` control-structure identifier *cs_id*, then resolve the location of the unresolved forward reference *orig*; both parameters were left by `aif` or `aelse`.

`athen` is part of the `assembler` absolute-address control-flow structure `aif` .. `aelse` .. `athen`, equivalent to Forth `if` .. `else` .. `then`.

See also: `rthen`, `?pairs`, `>resolve`.

Source file: <src/lib/assembler.fs>.

ato

```
ato ( x n "name" -- ) "a-to"
```

Store *x* into element *n* of 1-dimension single-cell values array *name*.

`ato` is an `immediate` word.

See also: `avalue`, `(ato`.

Source file: <src/lib/data.array.value.fs>.

attr!

```
attr! ( b -- ) "attribute-store"
```

Set *b* as the current attribute.

See also: `attr@`, `perm-attr!`, `set-paper`, `set-ink`, `set-flash`, `set-bright`.

Source file: <src/lib/display.attributes.fs>.

attr-cls

```
attr-cls ( b -- ) "attr-c-l-s"
```

Clear the screen with the attribute *b*, reset the graphic coordinates at the lower left corner (x 0, y 0) and reset the cursor position at the top left corner (column 0, row 0).

See also: `cls`, `page`, `attr-wcls`.

Source file: <src/kernel.z80s>.

attr-mask!

```
attr-mask! ( b -- ) "attribute-mask-store"
```

Set *b* as the current attribute mask.

See also: `attr-mask@`, `perm-attr-mask!`.

Source file: <src/lib/display.attributes.fs>.

attr-mask@

```
attr-mask@ ( -- b ) "attribute-mask-fetch"
```

Get the current attribute mask *b*.

See also: `attr-mask!`, `perm-attr-mask@`.

Source file: <src/lib/display.attributes.fs>.

attr-setter

```
attr-setter ( b "name" -- ) "attribute-setter"
```

Create a definition *name* that, when executed, will set *b* as the current attribute.

See also: `mask+attr-setter`.

Source file: <src/lib/display.attributes.fs>.

attr-wcls

```
attr-wcls ( b -- ) "attr-w-c-l-s"
```

Clear the `current-window` with color attribute *b* and reset its cursor position at the upper left corner (column 0, row 0).

See also: `wcolor`, `wcls`, `wblank`, `whome`, `clear-rectangle`, `cls`.

Source file: <src/lib/display.window.fs>.

attr>ink

```
attr>ink ( b1 -- b2 ) "attribute-to-ink"
```

Convert attribute *b1* to its ink color number *b2*.

`attr>ink` is written in Z80. Its equivalent definition in Forth is the following:

```
: attr>ink ( b1 -- b2 ) ink-mask and ;
```

See also: `attr>paper`, `ink-mask`.

Source file: <src/lib/display.attributes.fs>.

attr>paper

```
attr>paper ( b1 -- b2 ) "attribute-to-paper"
```

Convert attribute *b1* to its paper color number *b2*.

`attr>paper` is written in Z80. Its equivalent definition in Forth is the following:

```
: attr>paper ( b1 -- b2 ) paper-mask and 3 rshift ;
```

See also: `attr>ink`, `papery`, `paper-mask`, `rshift`.

Source file: <src/lib/display.attributes.fs>.

attr@

```
attr@ ( -- b ) "attribute-fetch"
```

Get the current attribute *b*.

See also: `attr!`, `perm-attr@`.

Source file: <src/lib/display.attributes.fs>.

auntil

```
auntil ( dest cs-id op -- ) "a-until"
```

`auntil` is part of the `assembler` absolute-address control-flow structure `abegin` .. `auntil`.

See also: `runtil`, `(auntil`, `inverse-cond`.

Source file: <src/lib/assembler.fs>.

avalue

```
avalue ( n "name" -- ) "a-value"
```

Create a 1-dimension single-cell values array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — x)

Return contents *x* of element *n*.

See also: `ato`, `+ato`.

Source file: <src/lib/data.array.value.fs>.

avariable

```
avariable ( n "name" -- ) "a-variable"
```

Create a 1-dimension single-cell variables array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — a)

Return address *a* of element *n*.

See also: `2avariable`, `cavariable`, `faravariable`.

Source file: <src/lib/data.array.variable.fs>.

awhile

```
awhile ( op -- orig cs-id ) "a-while"
```

Compile a Z80 `assembler` absolute-jump instruction *op*, which was put on the stack by `z?`, `nz?`, `c?`, `nc?`, `po?`, `pe?`, `p?`, or `m?`. Put the location of a forward reference *orig* onto the stack, to be resolved by `arepeat`, and the control-structure identifier *cs-id*.

`awhile` is part of the `assembler` absolute-address control-flow structure `abegin` .. `awhile` .. `arepeat`.

See also: `rwhile`.

Source file: <src/lib/assembler.fs>.

b

b

```
b ( -- reg )
```

Return the identifier *reg* of the Z80 `assembler` register "B", which is interpreted as register pair "BC" by `assembler` words that use register pairs (for example `ldp,`).

See also: `a`, `c`, `d`, `e`, `h`, `l`, `m`, `ix`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

b

```
b ( -- )
```

A command of `specforth-editor`: Used after `f` to backup the cursor by the length of the most recent text hold in `pad`.

See also: `c`, `d`, `e`, `f`, `h`, `i`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `x`.

Source file: <src/lib/prog.editor.specforth.fs>.

b/buf

```
b/buf ( -- n ) "b-slash-buf"
```

A `constant`. *n* is the number of bytes per `block buffer`: 1024.

Origin: fig-Forth[5], Forth-79 (Reference Word Set), Forth-83 (Uncontrolled Reference Words).

See also: `c/l`, `l/scr`.

Source file: <src/kernel.z80s>.

b/sector

```
b/sector ( -- n ) "b-slash-sector"
```

A `constant`. *n* is the number of bytes per sector.

See also: `sectors/block`, `sectors/track`.

Source file: <src/lib/dos.COMMON.fs>.

back-from-dos-error_

```
back-from-dos-error_ ( -- a )
```

Return the address *a* of Z80 code that can be set to be executed when a G+DOS routine throws an error, therefore preventing the DOS from returning to BASIC. This is needed only when a Forth word calls G+DOS routines directly instead of using hook codes.

`back-from-dos-error_` works like an alternative end to words that use direct G+DOS calls: The routine at `back-from-dos-error_` will leave the proper error result on the stack.

Usage example:

```
b push, dos-in,
  \ Save the Forth IP.
  \ Page in the G+DOS memory.

back-from-dos-error_ h ldp#, h push, 2066 sp stp,
  \ Set G+DOS D_ERR_SP ($2066) so an error will go to
  \   ``back&#45;from&#45;dos&#45;error_`` instead of returning to BASIC.
  \   This is needed because we are using direct calls to the
  \   G+DOS ROM instead of hook codes.

\ ... do whatever G+DOS direct call here ...

dos-out, h pop, b pop, next ix ldp#, ' false jp, end-code
  \ Page out the G+DOS memory.
  \ Consume the address of ``back&#45;from&#45;dos&#45;error_``
  \   that was pushed at the start.
  \ Restore the Forth IP.
  \ Restore the Forth IX.
  \ Return ``false`` _ior_ (no error).
```

Source file: <src/lib/dos.gplusdos.fs>.

backspace

```
backspace ( -- )
```

Emit a backspace character (character code 8).

See also: `'bs'`.

Source file: <src/lib/display.control.fs>.

backspaces

```
backspaces ( n -- )
```

Emit *n* number of backspace characters (character code 8).

See also: `backspace`, `'bs'`.

Source file: <src/lib/display.control.fs>.

baden-sqrt

```
baden-sqrt ( n1 -- n2 ) "baden-square-root"
```

Integer square root *n2* of radicand *n1*. Original code by Wil Baden, published on Forth Dimensions (volume 18, number 5, page 27, 1997-01). This method is 7..8 times faster than `newton-sqrt`.

Loading `baden-sqrt` makes it the action of `sqrt`.

See also: `(baden-sqrt`.

Source file: <src/lib/math.operators.1-cell.fs>.

bank

```
bank ( +n -- )
```

Page in the 16-KiB memory bank *+n* at $C000 .. $FFFF.

The range of *+n* depends on the computer:

Table 16. Range of memory banks per computer.

Computer	Memory banks
ZX Spectrum 128	0 .. 7
ZX Spectrum +2/+2A/+2B	0 .. 7
ZX Spectrum +3/+3e	0 .. 7
Pentagon 128	0 .. 7
Scorpion ZS 256	0 .. 15
Pentagon 512	0 .. 31
Pentagon 1024	0 .. 63

See also: `default-bank`, `banks`, `far-banks`.

Source file: <src/kernel.z80s>.

bank-heap

```
bank-heap ( n b a -- a )
```

Create a `heap` of *n* bytes at address *a* of bank *b*. *a* is the actual address ($C000..$FFFF) when bank *b* is paged in, which is stored in `heap-bank`.

`allocate`, `resize` and `free` page in bank *b* at the start and restore the default bank at the end.

See also: `heap-in`, `heap-out`, `allot-heap`, `limit-heap`, `farlimit-heap`, `empty-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

bank-index

```
bank-index ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the bank index (0 .. 3) calculated by the latest execution of `far`.

See also: `far-banks`, `bank`, `banks`.

Source file: <src/kernel.z80s>.

bank-start

```
bank-start ( -- a )
```

a is the memory address where banks are paged in: $C000.

See also: `/bank`, `bank`, `banks`, `far-banks`, `default-bank`.

Source file: <src/lib/memory.far.fs>.

banks

```
banks ( -- n )
```

A `cconstant`. *n* is the number of 16-KiB RAM memory banks:

Table 17. Number of memory banks per computer.

Computer	**Banks**
ZX Spectrum 128	8
ZX Spectrum +2/+2A/+2B	8
ZX Spectrum +3/+3e	8
Pentagon 128	8
Scorpion ZS 256	16
Pentagon 512	32
Pentagon 1024	64

See also: `bank`, `far-banks`, `default-bank`, `ram`.

Source file: <src/kernel.z80s>.

base

```
base ( -- a )
```

A `user` variable. *a* is the address of a cell containing the current number-conversion radix.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `>number`, `number?`, `abase`.

Source file: <src/kernel.z80s>.

base'

```
base' ( -- a ) "base-tick"
```

A temporary variable used by `<hex`, `hex>`, `<bin` and `bin>`. to store the current value of `base`.

See also: `abase`.

Source file: <src/lib/display.numbers.fs>.

base-execute

```
base-execute ( xt n -- )
```

Execute *xt* with the content of `base` being *n* and restoring the original `base` afterwards.

Source file: <src/lib/flow.MISC.fs>.

base>

```
base> ( -- ) "base-from"
```

Restore the previous value of `base` from `base'`. `base>` is executed by `bin>` and `hex>`.

Source file: <src/lib/display.numbers.fs>.

basic-pause

```
basic-pause ( u -- )
```

If *u* is zero, stop execution until a key is pressed. Otherwise stop execution during at least *u* clock `ticks`, or until a key is pressed.

`basic-pause` is a convenience that works like Sinclair BASIC's `PAUSE`.

See also: `ticks-pause`, `?ticks-pause`, `?seconds`, `ticks/second`.

Source file: <src/lib/time.fs>.

beep

```
beep ( duration pitch -- )
```

Produce a tone in the internal beeper, with parameters that are equivalent to those of the homonymous Sinclair BASIC command:

duration is in miliseconds (instead of seconds used by BASIC).

pitch is identical to the BASIC parameter: number of semitones from middle C (positive number for notes above, negative number for notes below).

Here is a diagram to show the pitch values of all the notes in one octave on the piano (extracted from the manual of the ZX Spectrum +3 transcripted by Russell et al.):

```
  |   | | | C#| D#| | | F#| G#| A#| | |   |   |
  |   | | | Db| Eb| | | Gb| Ab| Bb| | |   |   |
  |-2 | | | 1 | 3 | | | 6 | 8 |10 | | |13 |15 |
__|___| | |___|___| | |___|___|___| | |___|___|
    |   |   |   |   |   |   |   |   |   |   |
 -3 |-1 | 0 | 2 | 4 | 5 | 7 | 9 |11 |12 |14 |16
____|___|___|___|___|___|___|___|___|___|___|____
          C   D   E   F   G   A   B   C
```

Hence, to play the A above middle C for half a second, you would use:

```
500 9 beep
```

And to play a scale (for example, C major) a complete (albeit short) program is needed:

```
create scale
  0 c, 2 c, 4 c, 5 c, 7 c, 9 c, 11 c, 12 c,

8 constant /scale

: play-scale ( -- ) /scale 0 ?do
                      500 scale i + c@ beep
                    loop ;

play-scale
```

See also: `beep>bleep`, `bleep`, `beep>dhz`.

Source file: <src/lib/sound.48.fs>.

beep>bleep

```
beep>bleep ( duration1 pitch1 -- pitch2 duration2 ) "beep-to-bleep"
```

Convert *duration1* and *pitch1* of `beep`, which are equivalent to the parameters used by Sinclair BASIC's `BEEP` command, to *pitch2* and *duration2*, which are the parameters required by `bleep`.

NOTE *duration1* is in miliseconds (instead of seconds used by Sinclair BASIC).

pitch1 is identical to the Sinclair BASIC parameter: number of semitones from middle C (positive number for notes above, negative number for notes below).

See also: `beep>dhz`, `beep>note`.

Source file: <src/lib/sound.48.fs>.

beep>dhz

```
beep>dhz ( n -- u ) "beep-to-decihertz"
```

Convert a pitch *n* of `beep` to its corresponding frequency in dHz (tenths of hertzs) *u*.

See also: `beep>note`, `beep>bleep`.

Source file: <src/lib/sound.48.fs>.

beep>note

```
beep>note ( n1 -- n2 +n3 ) "beep-to-note"
```

Convert a pitch *n1* of `beep` to its corresponding note *+n3* (0..11) in octave *n2*, being zero the middle octave.

See also: `-beep>note`, `+beep>note`, `beep>dhz`, `beep>bleep`, `/octave`.

Source file: <src/lib/sound.48.fs>.

begin

```
begin
  Compilation: ( C: -- dest )
  Run-time:    ( -- )
```

Mark the start of a sequence for repetitive execution, leaving *dest* to be resolved by the corresponding `until`, `again` or `repeat`.

`begin` is an `immediate` and `compile-only` `alias` of `<mark`.

Definition:

```
' <mark alias begin immediate compile-only
  \ Compilation: ( C: -- dest )
  \ Run-time:    ( -- )
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `while`, `do`.

Source file: <src/kernel.z80s>.

begin-stringtable

```
begin-stringtable ( "name" -- a1 a2 )
```

Start a named stringtable definition "name", returning *a1* (containing the address of the strings index) and *a2* (the address of the compiled strings), to be consumed by `end-stringtable`.

Usage example:

```
begin-stringtable esperanto-number
  s" nulo" s,
  s" unu"  s,
  s" du"   s,
  s" tri"  s,
end-stringtable

0 esperanto-number type
3 esperanto-number type
```

See also: `sconstants`.

Source file: <src/lib/data.begin-stringtable.fs>.

begin-structure

```
begin-structure ( "name" -- struct-sys 0 )
```

Parse *name*. Create a definition for *name* with the execution semantics defined below. Return a *struct-sys* that will be used by `end-structure` and an initial offset of 0.

name execution: `( -- +n )`

+*n* is the size in memory expressed in bytes of the data structure.

Example usage:

```
begin-structure /record
  field:  ~year
  cfield: ~month
  cfield: ~day
end-structure

10 #records
create records #records /record * allot

: record> ( n -- a ) /record * records + ;
  \ Address _a_ of record _n_.

1887  0 record> ~year !    \ store a year into record 0
      9 record> ~month c@  \ fetch the month from record 9
```

NOTE `begin-structure` and `end-structure` are not necessary to create a structure. Only the initial offset 0 is needed at the start, and saving the structure size at the end, e.g. using a `constant` or a `value`:

```
0
  field:  ~the-cell
  cfield: ~the-char
constant /record
```

Origin: Forth-2012 (FACILITY EXT).

See also: `end-structure`, `field:`, `cfield:`, `2field:`, `+field`.

Source file: <src/lib/data.begin-structure.fs>.

bench.

```
bench. ( d -- ) "bench-dot"
```

Display the timing result *d*, which is a number of clock ticks, in ticks and seconds.

See also: `bench{`, `}bench`, `}bench.`.

Source file: <src/lib/time.fs>.

benched

```
benched ( xt n -- d )
```

Execute *n* times the benchmark *xt* and return the timer result *d*.

See also: `bench{`, `}bench`, `benched.`.

Source file: <src/lib/time.fs>.

benched.

```
benched. ( xt n -- d ) "benched-dot"
```

Execute *n* times the benchmark *xt* and display the result.

See also: `bench{`, `}bench.`, `benched`.

Source file: <src/lib/time.fs>.

bench{

```
bench{ ( -- ) "bench-curly-bracket"
```

Start timing, setting the clock ticks to zero.

See also: `}bench`, `reset-dticks`.

Source file: <src/lib/time.fs>.

between

```
between ( n1|u1 n2|u2 n3|u3 -- f )
```

Perform a comparison of a test value *n1*|*u1* with a lower limit *n2*|*u2* and an upper limit *n3*|*u3*, returning `true` if either (*n2*|*u2* ⇐ *n3*|*u3* and (*n2*|*u2* ⇐ *n1*|*u1* and *n1*|*u1* ⇐ *n3*|*u3*)) or (*n2*|*u2* > *n3*|*u3* and (*n2*|*u2* < *n1*|*u1* or *n1*|*u1* < *n3*|*u3*)) is true, returning `false` otherwise.

See also: `within`, `polarity`.

Source file: <src/lib/math.operators.1-cell.fs>.

between-of

```
between-of
  Compilation: ( C: -- of-sys )
  Run-time:    ( x1 x2 x3 -- | x1 )
```

A variant of `of`.

Compilation:

Put *of-sys* onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *of-sys*, such as `endof`.

Run-time:

If *x1* is not in range *x2 x3*, as calculated by `between`, discard *x2 x3* and continue execution at the location specified by the consumer of *of-sys*, e.g., following the next `endof`. Otherwise, consume also *x1* and continue execution in line.

`between-of` is an `immediate` and `compile-only` word.

Usage example:

```
: test ( n -- )
  case
    1           of  ." one"                  endof
    2 5 between-of  ." between two and five" endof
    6           of  ." six"                  endof
  endcase ;
```

See also: `case`, `within-of`, `(between-of`.

Source file: <src/lib/flow.case.fs>.

bin

```
bin ( fam1 -- fam2 )
```

Modify file access method *fam1* to additionally select a "binary", i.e., not line oriented, file access method, giving file access method *fam2*.

See also: `r/o`, `w/o`, `r/w`.

Origin: Forth-94 (FILE), Forth-2012 (FILE).

Source file: <src/lib/dos.gplusdos.fs>.

bin.

```
bin. ( n -- ) "bin-dot"
```

Display *n* as an unsigned binary number, followed by one space.

See also: `dec.`, `hex.`, `u.`, `.`.

Source file: <src/lib/display.numbers.fs>.

bin>

```
bin> ( -- ) "end-bin"
```

End a code zone where binary radix is the default, by restoring the value of `base` from `base'`. The zone was started by `<bin`.

Source file: <src/lib/display.numbers.fs>.

binary

```
binary ( -- )
```

Set contents of `base` to two.

See also: `decimal`, `hex`.

Source file: <src/lib/display.numbers.fs>.

bit,

```
bit, ( reg b -- ) "bit-comma"
```

Compile the Z80 `assembler` instruction `BIT b,reg`.

See also: `res,`, `set,`, `cp#,`.

Source file: <src/lib/assembler.fs>.

bit-array

```
bit-array ( n "name" -- )
```

Create a bit-array *name* to hold *n* bits, with the execution semantics defined below. The bits are stored in order: array bit 0 is bit 7 of the first byte of the array; array bit 7 is bit 0 of the first byte of the array; array bit 8 is bit 7 of the second byte of the array; array bit 15 is bit 0 of the second byte of the array, etc.

name (n — b ca)

Return bitmak *b* and address *ca* of bit *n* of the array.

See also: `@bit`, `!bit`, `bits>bytes`, `bitmasks`.

Source file: <src/lib/data.array.bit.fs>.

bit>mask

```
bit>mask ( n -- b ) "bit-to-mask"
```

Convert bit number *n* to a bitmask *b* with bit *n* set.

See also: `bit?`, `set-bit`, `reset-bit`.

Source file: <src/lib/memory.MISC.fs>.

bit?

```
bit? ( b n -- f ) "bit-question"
```

Is bit *n* of *b* set?

See also: `bit?`, `set-bit`, `bit>mask`.

Source file: <src/lib/memory.MISC.fs>.

bitmasks

```
bitmasks ( -- ca )
```

Address of an 8-byte table containing the bitmasks for bits 0..7 as used by `bit-array`.

Source file: <src/lib/data.array.bit.fs>.

bits

```
bits ( ca len -- u )
```

Count the number *u* of bits that are set in memory zone *ca len*.

See also: `pixels`.

Source file: <src/lib/math.operators.1-cell.fs>.

bits>bytes

```
bits>bytes ( n1 -- n2 ) "bits-to-bytes"
```

Return the number of bytes *n2* needed to hold *n1* bits. Used by `bit-array`.

Source file: <src/lib/data.array.bit.fs>.

bitx,

```
bitx, ( disp regpi b --  ) "bit-x-comma"
```

Compile the Z80 `assembler` instruction `BIT b,(regpi+disp)`.

See also: `resx,`, `setx,`, `cpx,`.

Source file: <src/lib/assembler.fs>.

bl

```
bl ( -- c ) "b-l"
```

A `cconstant`. *c* is the character value for a space.

Because space is used throughout Forth as the standard delimiter, `bl` is the only way a program has to find and use the character value of a space.

See also: `space`, `emit`.

Source file: <src/kernel.z80s>.

black

```
black ( -- b )
```

A `cconstant` that returns 0, the value that represents the black color.

See also: `blue`, `red`, `magenta`, `green`, `cyan`, `yellow`, `white`, `contrast`, `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

blackout

```
blackout ( -- )
```

Erase the screen (bitmap and the attributes) with zeros.

See also: `fade-display`, `cls`, `attr-cls`.

Source file: <src/lib/graphics.display.fs>.

blank

```
blank ( ca len -- )
```

If *len* is greater than zero, store the character value for space (`bl`) in *len* consecutive character positions of memory beginning at *ca*.

Origin: Forth-94 (STRING), Forth-2012 (STRING).

Source file: <src/kernel.z80s>.

bleep

```
bleep ( duration pitch -- )
```

Produce a tone in the internal beeper.

`bleep` calls the BEEPER ROM routine with *pitch* in the HL register and *duration* in the DE register.

> (…) but while there is greater flexibility than is directly available in BASIC the system is more difficult to use. Precalculation is necessary to obtain musical scales, on the following basis:
>
> To generate a frequency of F Hz, *pitch* must be set to:
>
> ```
> pitch = (437500/F)-30
> ```
>
> Looking in the opposite direction:
>
> ```
> F = 437500/(pitch+30)
> ```
>
> The duration of the note is determined as a number of cycles, so *duration* must be set to `F x T`, where *T* is the duration in seconds.
>
> A point to note is that if a very low frequency is selected, with a high duration, the system may appear to hang up, because the BEEPER ROM routine goes on and on…; whithout the user being able to use BREAK.
>
> — Don Thomasson, Spectrum Advanced Forth (Melbourne House, 1984), page 26

> Output a square wave of given duration and frequency to the loudspeaker.
>
> Enter with:
>
> - DE = #cycles - 1
> - HL = tone period as described next
>
> The tone period is measured in T states and consists of three parts: a coarse part (H register), a medium part (bits 7..2 of L) and a fine part (bits 1..0 of L) which contribute to the waveform timing as follows:
>
> ```
>                          coarse    medium       fine
> duration of low  = 118 + 1024*H + 16*(L>>2) + 4*(L&$3)
> duration of hi   = 118 + 1024*H + 16*(L>>2) + 4*(L&$3)
> Tp = tone period = 236 + 2048*H + 32*(L>>2) + 8*(L&$3)
>                  = 236 + 2048*H + 8*L = 236 + 8*HL
> ```
>
> As an example, to output five seconds of middle C (261.624 Hz):
>
> 1. Tone period = 1/261.624 = 3.822ms
> 2. Tone period in T-States = 3.822ms*fCPU = 13378 (where fCPU = clock frequency of the CPU = 3.5MHz)
> 3. Find H and L for desired tone period: HL = (Tp - 236) / 8 = (13378 - 236) / 8 = 1643 = $066B
> 4. Tone duration in cycles = 5s/3.822ms = 1308 cycles
> 5. DE = 1308 - 1 = $051B
>
> The resulting waveform has a duty ratio of exactly 50%.
>
> — Dr. Ian Logan, Dr. Frank O'Hara et al., ZX Spectrum disassembly

See also: `hz>bleep`, `dhz>bleep`, `beep`.

Source file: <src/lib/sound.48.fs>.

blk

```
blk ( -- a ) "b-l-k"
```

A `user` variable. *a* is the address of a cell containing zero or the number of the disk `block` being interpreted. If `blk` contains zero, the input source is not a block and can be identified by `source-id`.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (BLOCK), Forth-2012 (BLOCK).

See also: `load`, `loading?`, `?loading`.

Source file: <src/kernel.z80s>.

blk-line

```
blk-line ( -- ca len )
```

Return the current line *ca len* of the block being interpreted. No check is done whether any block is actually being interpreted.

See also: `blk`, `block`, `>in/l`, `->in/l`, `c/l`.

Source file: <src/lib/tool.list.blocks.fs>.

block

```
block ( u -- a )
```

If the block *u* is already in memory, leave the address *a* of the first cell in the disk `buffer` for data storage.

If the block *u* is not already in memory, transfer it from disk to the buffer. If the block occupying that buffer has been marked as updated, rewrite it to disk before block *u* is read into the buffer. Finally leave the address *a* of the first cell in the disk buffer for data storage.

Definition:

```
: block ( u -- a )
  dup buffer-block =
  if    drop
  else  save-buffers dup read-block disk-buffer !
  then  buffer-data ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `buffer-block`, `save-buffers`, `read-block`, `disk-buffer`, `buffer-data`.

Source file: <src/kernel.z80s>.

block-chars

```
block-chars ( -- )
```

A phoney word used only to do `need block-chars`. The loading of the correspondent source block will define characters 128..143 as block characters, with the shape they have in Sinclair BASIC. The current value of `os-udg` is used.

See also: `make-block-chars`, `set-udg`, `udg!`, `default-udg-chars`.

Source file: <src/lib/graphics.udg.fs>.

block-drive!

```
block-drive! ( c n -- ) "block-drive-store"
```

Set drive *c* (DOS dependent) as block drive number *n* (0 index).

See also: `block-drive@`, `set-block-drives`.

Source file: <src/lib/dos.COMMON.fs>.

block-drive@

```
block-drive@ ( n -- c ) "block-drive-fetch"
```

Get drive *c* (DOS dependent) currently used as block drive number *n* (0 index).

See also: `block-drive!`, `get-block-drives`.

Source file: <src/lib/dos.COMMON.fs>.

block-drives

```
block-drives ( -- ca )
```

ca is the address of a character table that holds the disk drives used as block drives. This table can be configured manually or using `set-block-drives`.

The length of the table is `max-drives`. The first element of the table (offset 0) is the disk drive used for blocks from number 0 to number `blocks/disk 1-`; the second element of the table (offset 1) the disk drive used for blocks from number `blocks/disk` to number `blocks/disk 2 * 1-`; and so on.

The number of used block drives is hold in `#block-drives`.

The block ranges not associated to disk drives are marked with $FF (the `not-block-drive` optional constant is provided for convenience), and all of them should be at the end of the table. In theory it's possible to define gaps in the whole range of blocks associated to disk drives, but this would cause trouble with `set-block-drives` and `get-block-drives`, which use `#block-drives` as the drives count from the start of `block-drives`.

The default configuration of `block-drives` is: use only the first disk drive for blocks.

Source file: <src/lib/dos.COMMON.fs>.

block-indexed

```
block-indexed ( block -- )
```

Mark block *block* as indexed.

See also: `use-fly-index`.

Source file: <src/lib/blocks.indexer.fly.fs>.

block-sector#>dos

```
block-sector#>dos ( n -- x ) "block-sector-number-sign-to-dos"
```

Convert the sequential disk sector *n* of a block drive to the disk sector id *x*, in the format required by G+DOS: The high byte of *x* is the track (0..79 for side 0; 128..207 for side 1); its low byte is the sector (1..10).

In G+DOS all sectors of a block disk can be used for blocks. Therefore `block-sector#>dos` is an alias of `sector#>dos`.

See also: `transfer-block`.

Source file: <src/kernel.gplusdos.z80s>.

block>source

```
block>source ( u -- ) "block-to-source"
```

Set `block` *u* as the current source.

Definition:

```
: block>source ( u -- ) blk ! >in off ;
```

See also: `terminal>source`, `blk`, `>in`, `set-source`, `lineblock>source`.

Source file: <src/kernel.z80s>.

block?

```
block? ( u -- f ) "block-question"
```

f is true if *u* is a valid `block` number.

Definition:

```
: block? ( u -- f ) max-blocks u< ;
```

Source file: <src/kernel.z80s>.

blocks/disk

```
blocks/disk ( -- n ) "blocks-slash-disk"
```

A `constant`. *n* is the number of blocks per disk.

See also: `sectors/block`, `sectors/track`.

Source file: <src/kernel.z80s>.

blue

```
blue ( -- b )
```

A `cconstant` that returns 1, the value that represents the blue color.

See also: `black`, `red`, `magenta`, `green`, `cyan`, `yellow`, `white`, `contrast`, `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

body>

```
body> ( dfa -- xt ) "body-from"
```

Convert *dfa* into its correspoding *xt*.

See also: `>body`, `body>name`.

Source file: <src/lib/compilation.fs>.

body>name

```
body>name ( dfa -- nt|0 ) "body-to-name"
```

Try to find the name token *nt* of the word represented by data field address *dfa*. Return 0 if it fails.

> **NOTE** `body>name` searches all word lists, from newest to oldest; and the searching of every word list is done also from the newest to the oldest definition. The first header whose execution token pointer contains the *xt* associated to *dfa* is a match. Therefore, when a word has additional headers created by `alias` or `synonym`, the *nt* of its latest alias or synonym is found first.

See also: `name>body`, `link>name`, `>name`.

Source file: <src/lib/compilation.fs>.

boot

```
boot ( -- )
```

A deferred word (see `defer`) executed by `abort`. By default it does nothing. It is changed by `turnkey`.

See also: `cold`.

Source file: <src/kernel.z80s>.

border

```
border ( n -- )
```

Set the border of the screen to color to *n*. Only the 3 lower bits of *n* are used (for colors 0 .. 7).

Source file: <src/kernel.z80s>.

bounds

```
bounds ( ca len -- ca2 ca )
```

Convert the string identifier *ca len* to *ca2 ca*, being *ca2* the address after the last character of the string. *ca2 ca* are the parameters needed by `do` or `?do` to traverse the string *ca len*.

`bounds` is written in Z80. Its equivalent definition in Forth is the following:

```
: bounds ( ca len -- ca2 ca ) over + swap ;
```

Origin: Comus.

See also: `count`.

Source file: <src/kernel.z80s>.

branch

```
branch ( -- )
```

The run-time procedure to branch unconditionally. The following in-line address is copied to IP to branch forward or backward.

Origin: Forth-83 (System Extension Word Set).

See also: `?branch`, `0branch`, `-branch`, `+branch`.

Source file: <src/kernel.z80s>.

break-key?

```
break-key? ( -- f ) "break-key-question"
```

f is true if the break key is pressed. `break-key?` is a deferred word (see `defer`) whose default action is `default-break-key?`.

See also: `key?`.

Source file: <src/kernel.z80s>.

bright-mask

```
bright-mask ( -- b )
```

A `cconstant`. *b* is the bitmask of the bit used to indicate the bright status in an attribute byte.

See also: `unbright-mask`, `brighty`, `set-bright`, `attr!`, `flash-mask`, `paper-mask`, `ink-mask`.

Source file: <src/lib/display.attributes.fs>.

bright.

```
bright. ( n -- ) "bright-dot"
```

Set bright *n* by printing the corresponding control characters. If *n* is zero, turn bright off; if *n* is one, turn bright on; if *n* is eight, set transparent bright. Other values of *n* are converted as follows:

- 2, 4 and 6 are converted to 0.

- 3, 5 and 7 are converted to 1.
- Values greater than 8 or less than 0 are converted to 8.

`bright.` is much slower than `set-bright` or `attr!`, but it can handle pseudo-color 8 (transparent), setting the corresponding system variables accordingly.

See also: `flash.`, `(0-1-8-color.`.

Source file: <src/lib/display.attributes.fs>.

brighty

```
brighty ( b1 -- b2 )
```

Convert attribute *b1* to its brighty equivalent *b2*.

`brighty` is written in Z80. Its equivalent definition in Forth is the following:

```
: brighty ( b1 -- b2 ) bright-mask or ;
```

See also: `bright-mask`, `papery`, `flashy`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

buffer

```
buffer ( u -- a )
```

Assign the `block` buffer to block *u*. If the contents of the buffer were marked as updated, it is written to the disk. The block *u* is not read from the disk. The address *a* left on stack is the first cell in the buffer for data storage.

Definition:

```
: buffer ( u -- a )
  dup buffer-block =
  if drop else free-buffer then buffer-data ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `buffer-block`, `free-buffer`, `buffer-data`.

Source file: <src/kernel.z80s>.

buffer-block

```
buffer-block ( -- n )
```

Return the block *n* associated with the disk buffer.

: buffer-block (— n) buffer-id $7FFF literal and ; ---

See also: `buffer-id`, `buffer`, `block`.

Source file: <src/kernel.z80s>.

buffer-data

```
buffer-data ( -- ca )
```

A `constant`. *ca* is the address of the disk `buffer` data.

See also: `disk-buffer`, `b/buf`.

Source file: <src/kernel.z80s>.

buffer-id

```
buffer-id ( -- x ) "buffer-i-d"
```

x is the identifier of the disk buffer.

See also: `disk-buffer`.

Source file: <src/kernel.z80s>.

buffer:

```
buffer: ( u "name" -- ) "buffer-colon"
```

Define a named uninitialized buffer as follows: Reserve *u* bytes of data space. Create a definition for *name* that will return the address of the space reserved by `buffer:` when it defined *name*. The program is responsible for initializing the contents.

Origin: Forth-2012 (CORE EXT).

See also: `reserve`, `allotted`, `create`, `allot`.

Source file: <src/lib/data.MISC.fs>.

bye

```
bye ( -- )
```

Return control to the host OS.

Definition:

```
: bye ( -- ) save-mode default-mode (bye ;
```

Origin: Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

See also: `save-mode`, `default-mode`, `(bye`, `warm`, `cold`.

Source file: <src/kernel.z80s>.

byte?

```
byte? ( x -- f ) "byte-question"
```

f is true if *x* is an 8-bit number. Used by `xliteral`.

Source file: <src/kernel.z80s>.

C

c

```
c ( -- reg )
```

Return the identifier *reg* of the Z80 `assembler` register "C".

See also: `a`, `b`, `d`, `e`, `h`, `l`, `m`, `ix`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

c

```
c ( n -- )
```

A command of `gforth-editor`: Move cursor by *n* chars.

See also: `a`, `g`, `n`, `p`, `t`.

Source file: <src/lib/prog.editor.gforth.fs>.

c

```
c ( "ccc<eol>" -- )
```

A command of `specforth-editor`: Copy in *ccc* to the cursor line at the cursor position.

See also: `b`, `d`, `e`, `f`, `h`, `i`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `x`, `(c`, `text`.

Source file: <src/lib/prog.editor.specforth.fs>.

c!

```
c! ( c ca -- ) "c-store"
```

Store *c* at *ca*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `!`, `2!`, `c@`.

Source file: <src/kernel.z80s>.

c!>

```
c!>
  Interpretation: ( c "name" -- )
  Compilation:    ( "name" -- )
  Run-time:       ( c -- )
"c-store-to"
```

A simpler and faster alternative to standard `to` and `value`.

`c!>` is an `immediate` word.

Interpretation:

Parse *name*, which is the name of a word created by `cconstant` or `cconst`, and make *c* its value.

Compilation:

Parse *name*, which is a word created by `cconstant` or `cconst`, and append the run-time semantics given below to the current definition.

Run-time:

Make *c* the current value of the character constant *name*.

Origin: IsForth’s `!>`.

See also: `!>`, `2!>`.

Source file: <src/lib/data.store-to.fs>.

c!a

```
c!a ( c -- ) "c-fetch-a"
```

Store *c* at the address register.

See also: `a`, `c@a`.

Source file: <src/lib/memory.address_register.fs>.

c!a+

```
c!a+ ( c -- ) "c-store-a-plus"
```

Store *c* at the address register and increment the address register by one address unit.

See also: `a`, `c@a+`.

Source file: <src/lib/memory.address_register.fs>.

c!bank

```
c!bank ( c ca n -- ) "c-store-bank"
```

Store *c* into address *ca* ($C000..$FFFF) of `bank` *n*.

`c!bank` is written in Z80. Its equivalent definition in Forth is the following:

```
: c!bank ( c ca n -- ) bank c! default-bank ;
```

See also: `c@bank`, `!bank`.

Source file: <src/lib/memory.far.fs>.

c!dos

```
c!dos ( b ca -- ) "c-store-dos"
```

Store *b* at the Plus D memory address *ca*.

See also: `c@dos`, `!dos`.

Source file: <src/lib/dos.gplusdos.fs>.

c!dosvar

```
c!dosvar ( b n -- ) "c-store-dos-var"
```

Store *b* into the G+DOS variable *n*.

See also: `c@dosvar`, `!dosvar`, `c!dos`.

Source file: <src/lib/dos.gplusdos.fs>.

c!exchange

```
c!exchange ( c1 ca -- c2 ) "c-store-exchange"
```

Store *c1* into *ca* and return its previous contents *c2*.

`c!exchange` is written in Z80. An equivalent definition in Forth is the following:

```
: c!exchange ( c1 ca -- c2 ) dup c@ rot rot c! ;
```

See also: `!exchange`, `cexchange`.

Source file: <src/lib/memory.MISC.fs>.

c"

```
c"
  Compilation: ( "ccc<quote>" -- )
  Run-time:    ( -- ca )
"c-quote"
```

Parse a string *ccc* delimited by double quotes and compile it into the current definition. At run-time the string will be returned as a counted string *ca*.

`c"` is an `immediate` and `compile-only` word.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `csliteral`.

Source file: <src/lib/strings.c-quote.fs>.

c#

```
c# ( "name" -- c ) "c-number-sign"
```

Parse *name* and return the code *c* of the its first character.

`c#` is a short and state-smart alternative to the standard words `char` and `[char]`.

`c#` is an `immediate` word.

WARNING `c#` is a state-smart word (see: `state`).

Source file: <src/lib/math.number.prefix.fs>.

c+!

```
c+! ( c ca - ) "c-plus-store"
```

Add *c* to the character stored at *ca*

See also: `c-!`, `c1+!`, `+!`.

Source file: <src/lib/memory.MISC.fs>.

c,

```
c, ( c -- ) "c-comma"
```

Reserve space for one character in the data space and store *c* in the space.

Definition:

```
: c, ( c -- ) here c! 1 allot ;
```

Origin: fig-Forth, Forth-79 (Reference Word Set), Forth-83 (Controlled Reference Words), Forth-94 (CORE), Forth-2012 (CORE).

See also: `,`, `2,`, `here`, `c!`, `allot`.

Source file: <src/kernel.z80s>.

c-!

```
c-! ( c ca - ) "c-minus-store"
```

Subtract *c* from the character stored at *ca*

See also: `c+!`, `c1-!`, `-!`.

Source file: <src/lib/memory.MISC.fs>.

c/l

```
c/l ( -- b ) "c-slash-l"
```

A `cconstant`. *b* is the number of characters per line in a `block` source: 64.

See also: `l/scr`.

Source file: <src/kernel.z80s>.

c1+!

```
c1+! ( ca - ) "c-one-plus-store"
```

Increment the character stored at *ca*.

See also: `c1-!`, `c+!`, `1+!`.

Source file: <src/lib/memory.MISC.fs>.

c1-!

```
c1-! ( ca - ) "c-one-minus-store"
```

Decrement the character stored at *ca*.

See also: `?c1-!`, `c1+!`, `c-!`, `1-!`.

Source file: <src/lib/memory.MISC.fs>.

c?

```
c? ( -- op ) "c-question"
```

Return the opcode *op* of the Z80 `assembler` instruction `jp c`, to be used as condition and consumed by `?ret,`, `?jp,`, `?call,`, `?jr,`, `aif`, `rif`, `awhile`, `rwhile`, `auntil` or `runtil`.

See also: `z?`, `nz?`, `nc?`, `po?`, `pe?`, `p?`, `m?`.

Source file: <src/lib/assembler.fs>.

c?

```
c? ( ca -- ) "c-question"
```

Display the 1-byte unsigned integer stored at *ca*, using the format of `.`.

See also: `?`, `2?`, `c@`.

Source file: <src/lib/memory.MISC.fs>.

c@

```
c@ ( ca -- c ) "c-fetch"
```

Fetch the character *c* stored at *ca*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `@`, `2@`, `c!`, `c@1+`, `c@1-`, `c@2+`, `c@2-`.

Source file: <src/kernel.z80s>.

c@+

```
c@+ ( ca -- ca' c ) "c-fetch-plus"
```

Fetch the character *c* at *ca*. Return *ca'*, which is *ca* incremented by one character. This is handy for stepping through character arrays.

`c@+` is an `alias` of `count`.

See also: `c@`, `2@+`, `@+`.

Source file: <src/lib/memory.MISC.fs>.

c@1+

```
c@1+ ( ca -- c ) "c-fetch-one-plus"
```

Fetch the character stored at *ca*, add 1 to it, according to the operation of `+`, giving *c*.

`c@1+` is a faster alternative to `c@ 1+`.

See also: `c@1-`, `c@2+`, `c@`, `1+`.

Source file: <src/lib/memory.MISC.fs>.

c@1-

```
c@1- ( ca -- c ) "c-fetch-one-minus"
```

Fetch the character stored at *ca*, subtract 1 from it, according to the operation of `-`, giving *c*.

`c@1-` is a faster alternative to `c@ 1-`.

See also: `c@1+`, `c@2-`, `c@`, `1-`.

Source file: <src/lib/memory.MISC.fs>.

c@2+

```
c@2+ ( ca -- c ) "c-fetch-two-plus"
```

Fetch the character stored at *ca*, add 2 to it, according to the operation of `+`, and return the result *c*.

`c@2+` is a faster alternative to `c@ 2+`.

See also: `c@2-`, `c@1+`, `c@`, `2+`.

Source file: <src/lib/memory.MISC.fs>.

c@2-

```
c@2- ( ca -- c ) "c-fetch-two-minus"
```

Fetch the character stored at *ca*, subtract 2 from it, according to the operation of `-`, and giving *c*.

`c@2-` is a faster alternative to `c@ 2-`.

See also: `c@2+`, `c@1-`, `c@`, `2-`.

Source file: <src/lib/memory.MISC.fs>.

c@a

```
c@a ( -- c ) "c-fetch-a"
```

Fetch the character *c* stored at the address register.

See also: `a`, `c!a`.

Source file: <src/lib/memory.address_register.fs>.

c@a+

```
c@a+ ( -- c ) "c-fetch-a-plus"
```

Fetch character *c* stored at the address register and increment the address register by one address unit.

See also: `a`, `c!a+`.

Source file: <src/lib/memory.address_register.fs>.

c@and

```
c@and ( b1 ca -- b2 ) "c-fetch-and"
```

Fetch the caracter at *ca* and do a bit-by-bit logical `and` of it with *b1*, returning the result *b2*.

See also: `c@and?`, `ctoggle`, `cset`, `creset`.

Source file: <src/lib/memory.MISC.fs>.

c@and?

```
c@and? ( b ca -- f ) "c-fetch-and-question"
```

Fetch the caracter at *ca* and do a bit-by-bit logical "and" of it with *b*. Return false if the result is zero, else true.

`c@and` is written in Z80. Its equivalent definition in Forth is the following:

```
: c@and? ( b ca -- f ) c@ and 0<> ;
```

See also: `c@and`.

Source file: <src/kernel.z80s>.

c@bank

```
c@bank ( ca n -- c ) "c-fetch-bank"
```

Fetch *c* from address *ca* ($C000..$FFFF) of `bank` *n*.

`c@bank` is written in Z80. Its equivalent definition in Forth is the following:

```
: c@bank ( ca n -- c )
  bank c@ default-bank ;
```

See also: `c!bank`, `@bank`.

Source file: <src/lib/memory.far.fs>.

c@dos

```
c@dos ( ca -- b ) "c-fetch-dos"
```

Fetch byte *b* stored at Plus D memory address *ca*.

See also: `c!dos`, `@dos`.

Source file: <src/lib/dos.gplusdos.fs>.

c@dosvar

```
c@dosvar ( n -- b ) "c-fetch-dos-var"
```

Fetch the contents *b* of G+DOS variable *n*.

See also: `c!dosvar`, `c!dosvar`, `@dos`.

Source file: <src/lib/dos.gplusdos.fs>.

calculator

```
calculator ( -- )
```

Start compilation of ROM calculator commands: Add `calculator-wordlist` to the search `order` and compile the following assembly instructions to start the ROM calculator:

```
push bc ; save the Forth IP
rst $28 ; call the ROM calculator
```

See also: `end-calculator`.

Source file: <src/lib/math.calculator.fs>.

calculator-command

```
calculator-command ( b -- )
```

Compile the assembly instructions needed to execute the *b* command of the ROM calculator.

See also: `end-calculator-flag`.

Source file: <src/lib/math.floating_point.rom.fs>.

calculator-command>flag

```
calculator-command>flag ( b -- ) "calculator-command-to-flag"
```

Compile the assembly instructions needed to execute the *b* command of the ROM calculator and to return the floating-point result as a flag on the data stack.

Source file: <src/lib/math.floating_point.rom.fs>.

calculator-wordlist

```
calculator-wordlist  ( -- wid )
```

The word list that contains the `calculator` commands.

Source file: <src/lib/math.calculator.fs>.

call

```
call ( a -- )
```

Call a machine code subroutine at *a*.

See also: `execute-hl,`, `call-xt,`.

Source file: <src/lib/flow.MISC.fs>.

call,

```
call, ( a -- ) "call-comma"
```

Compile the Z80 opcode to call *a*.

Definition:

```
: call, ( a -- ) $CD c, , ;
```

See also: `jp,`.

Source file: <src/kernel.z80s>.

call-xt,

```
call-xt, ( xt -- ) "call-x-t-comma"
```

Compile a Z80 `assembler` call to *xt*, by compiling the Z80 instruction that loads the HL register with *xt*, and then executing `execute-hl,` to compile the rest of the necessary code.

`call-xt,` is the low-level equivalent of `execute`: it's used to call a colon word from a code word.

See also: `call`, `call,`.

Source file: <src/lib/assembler.fs>.

capslock

```
capslock ( -- b ca )
```

Return address *ca* of system variable FLAGS2 and bitmask *b* of the bit that controls the status of capslock.

See also: `set-capslock`, `unset-capslock`, `capslock?`, `os-flags2`.

Source file: <src/lib/keyboard.caps_lock.fs>.

capslock?

```
capslock? ( -- f )
```

Is capslock set?

See also: `set-capslock`, `unset-capslock`, `toggle-capslock`, `capslock`, `c@and?`.

Source file: <src/lib/keyboard.caps_lock.fs>.

case

```
case
  Compilation: ( C: -- case-sys )
  Run-time:    ( -- )
```

Compilation: Mark the start of a `case` … `endcase` structure.

Run-time: Continue execution.

`case` is an `immediate` and `compile-only` word.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `of`, `endof`, `default-of`, `less-of`, `greater-of`, `between-of`, `within-of`, `or-of`, `any-of`, `cond`, `thens`.

Source file: <src/lib/flow.case.fs>.

case-sensitive

```
case-sensitive ( -- a )
```

A `variable`. *a* is the address of a cell containing a flag that turns case-sensitive mode on and off.

When the contents of `case-sensitive` are zero, case-sensitive mode is off (this is the default): the name of new words defined will be stored in lowercase into the dictionary; and any name searched for in the dictionary will be converted to lowercase first (the conversion is done at low level, not affecting the name string passed as parameter).

When the contents of `case-sensitive` are non-zero, case-sensitive mode is on: the name of new words defined will be stored as they are parsed from the input stream, without modification; and any name searched for in the dictionary will not be modified, therefore it will be found only if it's identical to the name stored in the definition header.

WARNING	Words that are defined when case-sensitive mode is on, and that have uppercase characters in their names, will not be found when case-sensitive mode is off.

Source file: <src/kernel.z80s>.

case-sensitive-esc-chars

```
case-sensitive-esc-chars ( -- a )
```

A `variable`. *a* is the address of a cell containing a flag that turns case-sensitive mode on and off only during the parsing of escaped strings, e.g. `s\"` and `.\"`. The contents of this variable are temporarily stored into `case-sensitive` by `parse-esc-string`. The current contents of `case-sensitive` are preserved and restored at the end.

When the contents of `case-sensitive` are non-zero, escaped characters case-sensitive mode is on (this is the default): any escaped character searched for in the configured word list will not be modified, therefore it will be found only if it's identical to the name stored in the definition header.

When the contents of `case-sensitive-esc-chars` are zero, escaped characters case-sensitive mode is off: any escaped character searched for in the correspondent word list will be converted to lowercase first (the conversion is done at low level, not affecting the name string passed as parameter).

NOTE	In order to create upper-case case-sensitive escaped chars, their correspondent words must be created when `case-sensitive` is on. See the words defined in `esc-udg-chars-wordlist`.

Source file: <src/lib/strings.escaped.fs>.

case>

```
case> ( orig counter selector "name" -- orig counter' ) "case-from"
```

Compile an option into a `cases:` structure. The given *selector* will cause the word *name* to be executed.

See `cases:` for an usage example.

Source file: <src/lib/flow.cases-colon.fs>.

cases:

```
cases: ( "name" -- orig 0 ) "cases-colon"
```

Define a `cases:` structure "name", built as an array of pairs (value and associated vector).

Usage example:

```
: say-10     ." dek" ;
: say-100    ." cent" ;
: say-1000   ." mil" ;
: say-other  ." alia" ;

cases: say ( n -- )
    10 case>      say-10
   100 case>      say-100
  1000 case>      say-1000
       othercase> say-other

10 say  100 say  1000 say  1001 say
```

Source file: <src/lib/flow.cases-colon.fs>.

cat

```
cat ( -- )
```

Show a disk catalogue of the current drive.

See also: `acat`, `wcat`, `wacat`, `(cat`, `set-drive`.

Source file: <src/lib/dos.gplusdos.fs>.

catch

```
catch ( i*x xt -- j*x 0 | i*x n )
```

Push an exception frame on the exception stack and then execute *xt* (as with `execute`) in such a way that control can be transferred to a point just after `catch` if `throw` is executed during the execution of *xt*.

If the execution of *xt* completes normally (i.e., the exception frame pushed by this `catch` is not popped by an execution of `throw`) pop the exception frame and return zero on top of the data stack, above whatever stack items would have been returned by the execution of *xt*. Otherwise, the remainder of the execution semantics are given by `throw`.

Solo Forth uses the return stack as exception stack. An exception frame includes the source specification saved by `nest-source`, the stack pointer returned by `sp@` and the contents of the previous `catcher`, which item is pointed by `catcher`.

Origin: Forth-94 (EXCEPTION), Forth-2012 (EXCEPTION).

Source file: <src/lib/exception.fs>.

catcher

```
catcher ( -- a )
```

A `user` variable. *a* is the address of a cell containing the return stack pointer for error handling. Used by `throw` and `catch`.

Source file: <src/kernel.z80s>.

cato

```
cato ( c n "name" -- ) "c-a-to"
```

Store *c* into element *n* of 1-dimension character values array *name*.

`cato` is an `immediate` word.

See also: `cavalue`, `(cato`.

Source file: <src/lib/data.array.value.fs>.

cavalue

```
cavalue ( n "name" -- ) "c-a-value"
```

Create a 1-dimension character values array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — c)

Return contents *c* of element *n*.

See also: `cato`, `+cato`.

Source file: <src/lib/data.array.value.fs>.

cavariable

```
cavariable ( n "name" -- ) "c-a-variable"
```

Create a 1-dimension character variables array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — ca)

Return address *ca* of element *n*.

See also: `avariable`, `2avariable`, `farcavariable`.

Source file: <src/lib/data.array.variable.fs>.

ccase

```
ccase "c-case"
  Compilation: ( C: -- orig1 orig2 )
  Run-time: ( c ca len -- )
```

Start a `ccase`..`endccase` structure. If *c* is in the string *ca len*, execute the n-th word compiled after `ccase`, where *n* is the position of the first *c* in the string (0..len-1) plus 1, then continue after `endccase`. If *c* is not in *ca len*, execute the word compiled right before `endccase`, then continue after `endccase`.

`ccase` is an `immediate` and `compile-only` word.

Usage example:

```
: .a    ( -- ) ." Letter A" ;
: .b    ( -- ) ." Letter B" ;
: .c    ( -- ) ." Letter C" ;
: .nope ( -- ) ." Nope!" ;

: letter ( c -- )
  s" abc" ccase  .a .b .c .nope  endccase
  ."  The End" cr ;
```

See also: `ccase0`, `?ccase`.

Source file: <src/lib/flow.ccase.fs>.

ccase0

```
ccase0 "c-case-zero"
  Compilation: ( C: -- orig )
  Run-time: ( c ca len -- )
```

Start a `ccase0`..`endccase` structure. If *c* is in the string *ca len*, execute the n-th word compiled after `ccase0`, where *n* is the position of the first *c* in the string (0..len-1) plus 1, then continue after `endccase0`. If *c* is not in *ca len*, execute the word compiled right after `ccase0`, then continue after `endccase0`.

`ccase0` is an `immediate` and `compile-only` word.

Usage example:

```
: .a    ( -- ) ." Letter A" ;
: .b    ( -- ) ." Letter B" ;
: .c    ( -- ) ." Letter C" ;
: .nope ( -- ) ." Nope!" ;

: letter ( c -- )
  s" abc" ccase0  .nope .a .b .c  endccase0
  ."  The End" cr ;
```

See also: `ccase` `?ccase`.

Source file: <src/lib/flow.ccase.fs>.

ccf,

```
ccf, ( -- ) "c-c-f-comma"
```

Compile the Z80 `assembler` instruction `CCF`.

See also: `cpl,`, `scf,`, `neg,`, `bit,`, `set,`, `cp,`.

Source file: <src/lib/assembler.fs>.

cconst

```
cconst ( c "name" -- ) "c-const"
```

Create a character fast constant *name*, with value *c*.

A character fast constant works like an ordinary `cconstant`, except its value is compiled as a literal.

Origin: IsForth’s `const`.

See also: `[cconst]`, `const`, `2const`.

Source file: <src/lib/data.const.fs>.

cconstant

```
cconstant ( c "name" -- ) "c-constant"
```

Parse *name*. Create a definition for *name* that will place *c* on the stack. *name* is referred to as a "c-constant".

Origin: Comus.

See also: `constant`, `2constant`, `c!>`, `cconst`, `[cconst]`, `cvalue`, `cvariable`.

Source file: <src/kernel.z80s>.

cell

```
cell ( -- n )
```

n is the size in bytes of one cell. `cell` returns 2 in Solo Forth.

Origin: Comus.

See also: `cells`, `cell+`, `cell-`, `cell/`, `cell-bits`.

Source file: <src/kernel.z80s>.

cell+

```
cell+ ( a1 -- a2 ) "cell-plus"
```

Add the size in bytes of a cell to *a1*, giving *a2*.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `cell`, `cells`, `cell-`, `cell/`.

Source file: <src/kernel.z80s>.

cell-

```
cell- ( a1 -- a2 ) "cell-minus"
```

Subtract the size in bytes of a cell from *a1*, giving *a2*.

Origin: Comus.

See also: `cell`, `cell+`, `cells`, `cell/`.

Source file: <src/kernel.z80s>.

cell-bits

```
cell-bits  ( -- n )
```

A `cconstant`. *n* is the number of bits in a cell.

See also: `cell`, `environment?`.

Source file: <src/lib/math.number.conversion.fs>.

cell/

```
cell/ ( n1 -- n2 ) "cell-slash"
```

Divide *n1* by the size of a cell, returning the result *n2*.

See also: `cell`, `cells`, `cell+`, `cell-`.

Source file: <src/lib/math.operators.1-cell.fs>.

cells

```
cells ( n1 -- n2 )
```

n2 is the size in bytes of *n1* cells.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `cell`, `cell+`, `cell-`, `cell/`.

Source file: <src/kernel.z80s>.

centry:

```
centry: ( c wid "name" -- ) "c-entry-colon"
```

Create a character entry *name* in the `associative-list` *wid*, with value *c*.

See also: `entry:`, `2entry:`, `sentry:`, `create-entry`.

Source file: <src/lib/data.associative-list.fs>.

cenum

```
cenum ( n "name" -- n+1 ) "c-enum"
```

Create a cconstant *name* with value *n* and return *n+1*.

Usage example:

```
0 cenum first
  cenum second
  cenum third
  cenum fourth
drop
```

See also: `enum`, `enumcell`.

Source file: <src/lib/data.MISC.fs>.

cexchange

```
cexchange ( ca1 ca2 -- ) "c-exchange"
```

Exchange the characters stored in *ca1* and *ca2*.

`cexchange` is written in Z80. An equivalent definition in Forth is the following:

```
: cexchange ( ca1 ca2 -- ) 2dup c@ swap c@ rot c! swap c! ;
```

See also: `exchange`, `c!exchange`.

Source file: <src/lib/memory.MISC.fs>.

cfield:

```
cfield: ( n1 "name" -- n2 ) "c-field-colon"
```

Parse *name*. *offset* is the first character aligned value greater than or equal to *n1*. *n2* = *offset* + 1 character.

Create a definition for *name* with the execution semantics defined below.

name execution: `( a1 -- a2 )`

Add the *offset* calculated during the compile-time action to *a1* giving the address *a2*.

Origin: Forth-2012 (FACILITY EXT).

See also: `begin-structure`, `+field`.

Source file: <src/lib/data.begin-structure.fs>.

chan>

```
chan> ( n -- a ) "chan-to"
```

Convert channel offset *n* in `os-chans`, fetched from an element of `os-strms`, to its address *a*.

See also: `chan>id`, `os-chans`.

Source file: <src/lib/os.fs>.

chan>id

```
chan>id ( n -- c ) "chan-to-id"
```

Convert channel offset *n* in `os-chans`, fetched from an element of `os-strms`, to its character identifier *c*.

See also: `chan>`, `os-chans`.

Source file: <src/lib/os.fs>.

change-octave

```
change-octave ( u n -- u' )
```

Change the note frequency *u* of the middle octave (octave zero) to its corresponding note frequency *u'* in octave *n*. If *n* is zero, *u'* equals *u*.

See also: `octave-changer`, `beep>dhz`, `middle-octave`.

Source file: <src/lib/sound.48.fs>.

channel

```
channel ( n -- )
```

Open channel *n* for output. Store *n* into `current-channel`.

See also: `terminal`, `printer`, `printing`.

Source file: <src/kernel.z80s>.

char

```
char ( "name" -- c )
```

Parse *name* and put the value of its first character on the stack.

Solo Forth recognizes the standard notation for characters, so `char` is not needed:

```
'x' emit .(  equals ) char x emit
```

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `[char]`.

Source file: <src/lib/parsing.fs>.

char+

```
char+ ( n1 -- n2 ) "char-plus"
```

Add the size in bytes of a character to *n1*, giving *n2*.

`char+` is an `alias` of `1+`.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `char-`, `chars`.

Source file: <src/kernel.z80s>.

char-

```
char- ( n1 -- n2 ) "char-minus"
```

Subtract the size in bytes of a character to *n1*, giving *n2*.

`char-` is an `alias` of `1-`.

Origin: Comus.

See also: `char+`.

Source file: <src/kernel.z80s>.

char-in-string?

```
char-in-string? ( c ca len -- f ) "char-in-string-question"
```

Is char *c* in string *ca len*? `char-in-string?` is a factor of `string-char?`: Its only difference is the order of the input parameters.

See also: `char-position?`, `contains`, `compare`, `#chars`.

Source file: <src/lib/strings.MISC.fs>.

char-position?

```
char-position? ( ca len c -- +n true | false ) "char-position-question"
```

If char *c* is in string *ca len*, return its first position +*n* and `true`; else return `false`.

See also: `char-in-string?`, `contains`, `compare`.

Source file: <src/lib/strings.MISC.fs>.

char>string

```
char>string ( c -- ca len ) "char-to-string"
```

Convert the char *c* to a string *ca len* in the `stringer`.

See also: `chars>string`, `ruler`, `u>str`, `d>str`, `ud>str`, `>bstring`, `2>bstring`.

Source file: <src/lib/strings.MISC.fs>.

char?

```
char? ( ca len -- c true | false ) "char-question"
```

If the string *ca len* is the representation of a character, return the character *c* and `true`; else return `false`.

Definition:

```
: char? ( ca len -- c true | false )
  3 = if
    dup c@ ''' <> if
      dup [ 2 chars ] cliteral + c@ ''' <>
      if char+ c@ true exit then
    then
  then
  drop false ;
```

Source file: <src/kernel.z80s>.

charlton-allocate

```
charlton-allocate ( u -- a ior )
```

Allocate *u* bytes of contiguous data space. The data-space pointer is unaffected by this operation. The initial content of the allocated space is undefined.

If the allocation succeeds, *a* is the starting address of the allocated space and *ior* is zero.

If the operation fails, *a* does not represent a valid address and the I/O resul code *ior* is #-59, the `throw` code for `allocate`.

`charlton-allocate` is the action of `allocate` in the memory `heap` implementation adapted from code written by Gordon Charlton, whose words are defined in `charlton-heap-wordlist`.

See also: `charlton-resize`, `charlton-free`.

Source file: <src/lib/memory.allocate.charlton.fs>.

charlton-empty-heap

```
charlton-empty-heap ( -- )
```

Empty the current `heap`, which was created by `allot-heap`, `limit-heap`, `bank-heap` or `farlimit-heap`.

`charlton-empty-heap` is the action of `empty-heap` in the memory `heap` implementation adapted from code written by Gordon Charlton, whose words are defined in `charlton-heap-wordlist`.

See also: `charlton-allocate`, `charlton-resize`, `charlton-free`.

Source file: <src/lib/memory.allocate.charlton.fs>.

charlton-free

```
charlton-free ( a -- ior )
```

Return the contiguous region of data space indicated by *a* to the system for later allocation. *a* shall indicate a region of data space that was previously obtained by `charlton-allocate` or `charlton-resize`.

As there is no compelling reason for this to fail, *ior* is zero.

`charlton-free` is the action of `free` in the memory `heap` implementation adapted from code written by Gordon Charlton, whose words are defined in `charlton-heap-wordlist`.

Source file: <src/lib/memory.allocate.charlton.fs>.

charlton-heap-wordlist

```
charlton-heap-wordlist ( -- wid )
```

wid is the word-list identifier of the word list that holds the words the memory `heap` implementation adapted from code written by Gordon Charlton (1994-09-12).

`need charlton-heap-wordlist` is used to load the memory heap implementation and configure `allocate`, `resize`, `free` and `empty-heap` accordingly.

An alternative, simpler and smaller implementation of the memory heap is provided by `gil-heap-wordlist`.

The actual heap must be created with `allot-heap`, `limit-heap`, `farlimit-heap` or `bank-heap`, which are independent from the heap implemention.

Source file: <src/lib/memory.allocate.charlton.fs>.

charlton-resize

```
charlton-resize ( a1 u -- a2 ior )
```

Change the allocation of the contiguous data space starting at the address *a1*, previously allocated by `charlton-allocate` or `charlton-resize`, to *u* bytes. *u* may be either larger or smaller than the current size of the region. The data-space pointer is unaffected by this operation.

If the operation succeeds, *a2* is the starting address of *u* bytes of allocated memory and *ior* is zero. *a2* may be, but need not be, the same as *a1*. If they are not the same, the values contained in the region at *a1* are copied to *a2*, up to the minimum size of either of the two regions. If they are the same, the values contained in the region are preserved to the minimum of *u* or the original size. If *a2* is not the same as *a1*, the region of memory at *a1* is returned to the system according to the operation of `free`.

If the operation fails, *a2* equals *a1*, the region of memory at *a1* is unaffected, and the I/O result code *ior* is #-61, the `throw` code for `resize`.

`charlton-resize` is the action of `resize` in the memory `heap` implementation adapted from code written by Gordon Charlton, whose words are defined in `charlton-heap-wordlist`.

Source file: <src/lib/memory.allocate.charlton.fs>.

chars

```
chars ( n1 -- n2 )
```

n2 is the size in bytes of *n1* characters. In Solo Forth `chars` does nothing, therefore *n1* equals *n2*.

`chars` is an `immediate` word.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

Source file: <src/kernel.z80s>.

chars>string

```
chars>string ( c#1..c#n n -- ca len ) "chars-to-string"
```

Convert *n* chars to a string *ca len* in the `stringer`, being *c#1* the last character of the string and *c#n* the first one.

See also: `char>string`, `2>bstring`, `>bstring`, `ruler`, `s+`.

Source file: <src/lib/strings.MISC.fs>.

chop

```
chop ( ca len -- ca' len' )
```

Remove the last character from string *ca len*.

See also: `-suffix`, `/string`, `string/`.

Source file: <src/lib/strings.MISC.fs>.

circle

```
circle ( gx gy b -- )
```

Draw a circle at center coordinates *gx gy* and with radius *b*.

`circle` does not use the ROM routine and it’s much faster.

`circle` does no error checking: the whole circle must fit the screen. Otherwise, strange things will happen when other parts of the screen bitmap, the screen attributes or even the system variables will be altered.

NOTE	By default `circle` does nothing. Its factor routine `circle-pixel` must be configured first with `set-circle-pixel`, in order to choose the routine that creates the pixels of the circle: `uncolored-circle-pixel`, `colored-circle-pixel` or a routine provided by the application.

Source file: <src/lib/graphics.circle.fs>.

circle-pixel

```
circle-pixel ( -- a )
```

a is the address of a subroutine used by `circle` to set its pixels. This routine does a jump to the actual routine, which by default does nothing. The desired routine must be set by `set-circle-pixel`.

Also any routine provided by the application can be used as the action of `circle-pixel`, provided the following requirements:

- HL, DE and BC must be preserved.
- Input parameters: B=gy and C=gx.

Source file: <src/lib/graphics.circle.fs>.

class

```
class ( class -- class methods vars )
```

Start the definition of a class.

Source file: <src/lib/objects.mini-oof.fs>.

classic-number-point?

```
classic-number-point? ( c -- f )
"classic-number-point-question"
```

Is character *c* a classic number point? Allowed points are: comma, hyphen, period, slash and colon.

`classic-number-point?` is an alternative action for the deferred word `number-point?` (see `defer`), which is used in `number?`, and whose default action is `standard-number-point?`.

See also: `extended-number-point?`.

Source file: <src/lib/math.number.point.fs>.

clear

```
clear ( n -- )
```

A command of `specforth-editor`: Clear block *n* with blanks and select for editing.

See also: `e`, `l/scr`.

Source file: <src/lib/prog.editor.specforth.fs>.

clear-rectangle

```
clear-rectangle ( column row width height color -- )
```

Clear a screen rectangle at the given character coordinates and of the given size in characters. The bitmap is erased and the color attributes are changed with the given color attribute.

`clear-rectangle` is written in Z80 and it combines the functions of `wipe-rectangle` and `color-rectangle`. It may be defined also this way (with slower but much smaller code):

```
: clear-rectangle ( column row width height color -- )
  >r 2over 2over wipe-rectangle r> color-rectangle ;
```

See also: `attr-wcls`.

Source file: <src/lib/graphics.rectangle.fs>.

clit

```
clit ( -- b ) "c-lit"
```

Return *b*, which was compiled by `cliteral` after `clit`.

`clit` is a `compile-only` word.

See also: `lit`, `2lit`.

Source file: <src/kernel.z80s>.

cliteral

```
cliteral ( b -- ) "c-literal"
```

Compile *b* in the current definition.

`cliteral` does the same as `literal` but saves one byte of data space and *b* is put on the stack a bit faster (0.97 of execution speed).

`cliteral` is an `immediate` and `compile-only` word.

Definition:

```
: cliteral ( b -- ) postpone clit c, ; immediate compile-only
```

Origin: Comus.

See also: `clit`, `2literal`, `xliteral`, `]cl`.

Source file: <src/kernel.z80s>.

clocal

```
clocal ( ca -- ) "c-local"
```

Save the value of the character variable *ca*, which will be restored at the end of the current definition.

`clocal` is a `compile-only` word.

Usage example:

```
cvariable v
1 v c!  v c? \ default value

: test ( -- )
  v clocal
  v c? 1887 v c!  v c? ;

v c? \ default value
```

See also: `local`, `2local`, `arguments`, `anon`.

Source file: <src/lib/locals.local.fs>.

clr,

```
clr, ( reg -- ) "c-l-r-comma"
```

Compile the Z80 `assembler` instruction `LD reg,0`.

See also: `clrp,`, `ld#,`.

Source file: <src/lib/assembler.fs>.

clrp,

```
clrp, ( regp -- ) "c-l-r-p-comma"
```

Compile the Z80 `assembler` instruction `LD regp,0`.

See also: `clr,`, `ldp#,`.

Source file: <src/lib/assembler.fs>.

cls

```
cls ( -- ) "c-l-s"
```

Clear the screen with the current attribute, reset the graphic coordinates at the lower left corner (gx 0, gy 0) and reset the cursor position at the upper left corner (column 0, row 0).

See also: `attr!`, `attr-cls`, `page`, `wcls`.

Source file: <src/kernel.z80s>.

cls-chars1

```
cls-chars1 ( -- ) "c-l-s-chars-one"
```

Clear the screen by rotating all bytes of the bitmap.

Source file: <src/lib/graphics.cls.fs>.

clshift

```
clshift ( b1 u -- b2 ) "c-l-shift"
```

Perform a logical left shift of *u* bit-places on *b1*, giving *b2*. Put zeroes into the least significant bits vacated by the shift.

See also: `lshift`.

Source file: <src/lib/math.operators.1-cell.fs>.

cmove

```
cmove ( ca1 ca2 u -- ) "c-move"
```

If *u* is greater than zero, copy *u* consecutive characters from the data space starting at *ca1* to that starting at *ca2*, proceeding character-by-character from lower addresses to higher addresses.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (STRING), Forth-2012 (STRING).

See also: `cmove>`, `move`.

Source file: <src/kernel.z80s>.

cmove<far

```
cmove<far ( ca1 ca2 len -- ) "c-move-from-far"
```

If *len* is greater than zero, copy *len* consecutive characters from far-memory address *ca1* to main-memory address *ca2*.

Source file: <src/lib/memory.far.fs>.

cmove>

```
cmove> ( ca1 ca2 u -- ) "c-move-up"
```

If *u* is greater than zero, copy *u* consecutive characters from the data space starting at *ca1* to that starting at *ca2*, proceeding character-by-character from higher addresses to lower addresses.

Origin: Forth-83 (Required Word Set), Forth-94 (STRING), Forth-2012 (STRING).

See also: `cmove`, `move`.

Source file: <src/kernel.z80s>.

cmove>far

```
cmove>far ( ca1 ca2 len -- ) "c-move-to-far"
```

If *len* is greater than zero, copy *len* consecutive characters from main-memory address *ca1* to far-memory address *ca2*.

Source file: <src/lib/memory.far.fs>.

code

```
code ( "name -- )
```

Parse *name*. Create a definition for *name*, called a code definition, and execute `asm` to enter `assembler` mode.

Definition:

```
: code ( "name -- ) header asm ;
```

Origin: Forth-79 (Assembler Word Set), Forth-83 (Assembler Extension Word Set), Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

Source file: <src/kernel.z80s>.

coff

```
coff ( ca -- ) "c-off"
```

Store `false` at *ca*.

`coff` is written in Z80. Its equivalent definition in Forth is the following:

```
: coff ( ca -- ) false swap c! ;
```

See also: `off`.

Source file: <src/lib/memory.MISC.fs>.

cold

```
cold ( -- )
```

Restore the Forth system to its default status, i.e. as if it were just booted the first time, except the background picture is not displayed.

Origin: fig-Forth.

See also: `warm`, `greeting`.

Source file: <src/kernel.z80s>.

color-rectangle

```
color-rectangle ( column row width height color -- )
```

Color a screen rectangle at the given character coordinates and of the given size in characters with the given color attribute. Only the color attributes are changed; the bitmap remains unchanged.

See also: `wcolor`, `wipe-rectangle`, `clear-rectangle`.

Source file: <src/lib/graphics.rectangle.fs>.

colored-circle-pixel

```
colored-circle-pixel ( -- a )
```

a is the address of a subroutine that `circle` can use to draw its pixels. This routine sets a pixel, changing its color attributes on the screen (like `plot`). Therefore it’s slower than its alternative `uncolored-circle-pixel` (1.64 its execution speed).

`set-circle-pixel` sets the routine used by `circle`. See the requirements of such routine in the documentation of `circle-pixel`.

Source file: <src/lib/graphics.circle.fs>.

column

```
column ( -- col )
```

Current column (x coordinate).

See also: `row`, `last-column`, `columns`.

Source file: <src/lib/display.cursor.fs>.

columns

```
columns ( -- n )
```

Return the number of columns in the current screen mode. The default value is 32.

See also: `rows`, last-column`, `column`.

Source file: <src/lib/display.mode.COMMON.fs>.

comp'

```
comp' ( "name" -- x xt ) "comp-tick"
```

Compilation token *x xt* represents the compilation semantics of *name*.

Origin: Gforth.

See also: `[comp']`, `name>compile`, `[']`.

Source file: <src/lib/compilation.fs>.

compare

```
compare ( ca1 len1 ca2 len2 -- n )
```

Compare the string *ca1 len1* to the string *ca2 len2*. The strings are compared, beginning at the given addresses *ca1* and *ca2*, character by character, up to the length of the shorter string or until a difference is found. If the two strings are identical, *n* is zero. If the two strings are identical up to the length of the shorter string, *n* is minus-one (-1) if *len1* is less than *len2* and one (1) otherwise. If the two strings are not identical up to the length of the shorter string, *n* is minus-one (-1) if the first non-matching character in the string *ca1 len1* has a lesser numeric value than the corresponding character in the string *ca2 len2* and one (1) otherwise.

Origin: Forth-94 (STRING), Forth-2012 (STRING).

See also: `str=`, `str<`, `str>`.

Source file: <src/kernel.z80s>.

compilation-only

```
compilation-only ( -- )
```

`throw` exception code #-14 ("interpreting a compile-only word").

`compilation-only` is used in `interpret-table`.

See also: `not-understood`, `?compiling`.

Source file: <src/kernel.z80s>.

compile

```
compile ( -- )
```

Compile the cell following the compilation address of `compile` into the dictionary.

`compile` allows specific compilation situations to be handled in addition to simply compiling an execution token (which the interpreter already does).

`compile` is a `compile-only` word.

Definition:

```
: compile ( -- ) r> dup cell+ >r  @ compile, ;
```

Typically used in the form:

```
: name compile namex ;
```

When *name* is executed, the execution token of *namex* is compiled, not executed. *name* is tipically an `immediate` word and *namex* is typically not an `immediate` word.

`compile` has been superseded by `postpone`.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set).

See also: `[compile]`, `compile,`.

Source file: <src/kernel.z80s>.

compile,

```
compile, ( xt -- ) "compile-comma"
```

Append the execution semantics of the definition represented by *xt* to the execution semantics of the current definition.

`compile,` is the compilation equivalent of `execute`.

Since Solo Forth is a threaded-code implementation, `compile,` is an `alias` of `,`.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

Source file: <src/kernel.z80s>.

compile-only

```
compile-only ( -- )
```

Make the most recent definition a compile-only word.

Definition:

```
: compile-only ( -- ) compile-only-mask latest lex! ;
```

See also: `compile-only?`, `compile-only-mask`, `?compiling`, `lex!`, `latest`, `immediate`.

Source file: <src/kernel.z80s>.

compile-only-mask

```
compile-only-mask ( -- b )
```

A `cconstant`. *b* is the bitmask of the compile-only bit, set by `compile-only`.

See also: `immediate-mask`, `smudge-mask`, `word-length-mask`.

Source file: <src/kernel.z80s>.

compile-only?

```
compile-only? ( nt -- f ) "compile-only-question"
```

f is true if the word *nt* is compile-only.

Definition:

```
: compile-only? ( nt -- f ) compile-only-mask lex? ;
```

See also: `compile-only`, `immediate?`.

Source file: <src/kernel.z80s>.

compiling?

```
compiling? ( -- f ) "compiling-question"
```

f is true if `state` is not zero, i.e. the Forth system is in compilation state.

Definition:

```
: compiling? ( -- f ) state @ 0<> ;
```

Source file: <src/kernel.z80s>.

con

```
con ( ca -- ) "c-on"
```

Store `true` at *ca*.

`con` is written in Z80. Its equivalent definition in Forth is the following:

```
: con ( ca -- ) true swap c! ;
```

NOTE The value actually stored is not `true`, which is a cell, but its 8-bit equivalent $FF.

See also: `coff`, `on`.

Source file: <src/lib/memory.MISC.fs>.

cond

```
cond
  Compilation: ( C: -- cs-mark )
  Run-time:    ( -- )
```

Compilation: Mark the start of a `cond` … `thens` structure. Leave *cs-mark* on the control-flow stack, to be checked by `thens`.

Run-time: Continue execution.

`cond` is an `immediate` and `compile-only` word.

Generic usage example:

```
: test ( x -- )
  cond
    test1 if action1 else
    test2 if action2 else
    test3 if action3 else
    default-action
  thens ;
```

NOTE The tested value must be preserved and discarded by the application. Example:

```
: test ( ca len -- )
  cond
    2dup s" first"  str= if 2drop ." unua"  else
    2dup s" second" str= if 2drop ." dua"   else
    2dup s" third"  str= if 2drop ." tria"  else
    2dup s" fourth" str= if 2drop ." kvara" else
    type ." ?"
  thens ;
```

See also: `case`, `cs-mark`, `andif`, `orif`.

Source file: <src/lib/flow.MISC.fs>.

const

```
const ( x "name" -- )
```

Create a fast constant *name*, with value *x*.

A fast constant works like an ordinary `constant`, except its value is compiled as a literal.

Origin: IsForth.

See also: `[const]`, `cconst`, `2const`.

Source file: <src/lib/data.const.fs>.

constant

```
constant ( x "name" -- )
```

Parse *name*. `create` a definition for *name* that will place *x* on the stack. *name* is referred to as a "constant".

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `2constant`, `cconstant`, `fconstant`, `!>`, `const`, `[const]`, `value`, `variable`.

Source file: <src/kernel.z80s>.

contains

```
contains ( ca1 len1 ca2 len2 -- f )
```

Does string *ca1 len1* contain string *ca2 len2*?

See also: `char-position?`, `char-in-string?`, `compare`, `#chars`,

Source file: <src/lib/strings.MISC.fs>.

context

```
context ( -- a )
```

A `user` variable. *a* is the address of an array of cells that represents the search `order`; its maximum length is hold in the `max-order` constant, and its current length is hold in the `#order` variable. *a* holds the word list at the top of the search order.

See also: `>order`, `get-order`, `set-order`.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (System Extension Word Set).

Source file: <src/kernel.z80s>.

continued

```
continued ( u -- )
```

Continue interpretation at block *u*.

Origin: Forth-79 (Reference Word Set), Forth-83 (Appendix B. Uncontrolled Reference Words).

See also: `-->`, `load`.

Source file: <src/lib/blocks.fs>.

contrast

```
contrast ( b1 -- b2 )
```

Convert color *b1* to its contrast color *b2*. *b2* is `white` (7) if *b1* is a dark color (`black`, `blue`, `red` or `magenta`); *b2* is `black` (0) if *b1* is a light colour (`green`, `cyan`, `yellow` or `white`).

See also: `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

control-char?

```
control-char? ( c -- f ) "control-char-question"
```

Is character *c* a control character, i.e. in the range 0..31?

See also: `ascii-char?`.

Source file: <src/lib/chars.fs>.

copy

```
copy ( n1 n2 -- )
```

A command of `specforth-editor`: Copy `block` *n1* to block *n2*.

See also: `update`, `save-buffers`.

Source file: <src/lib/prog.editor.specforth.fs>.

count

```
count ( ca1 -- ca2 len2 )
```

Return the character string specification for the counted string stored at *ca1*. *ca2* is the address of the first character after *ca1*. *len* is the contents of the character at *c1*, which is the length in characters of the string at *c2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `farcount`.

Source file: <src/kernel.z80s>.

counted>stringer

```
counted>stringer ( ca1 len1 -- ca2 ) "counted-to-stringer"
```

Copy string *ca1 len1* to the `stringer` as a counted string and return it as *ca2*.

See also: `>stringer`, `allocate-stringer`.

Source file: <src/lib/strings.MISC.fs>.

cp#,

```
cp#, ( b -- ) "c-p-number-sign-comma"
```

Compile the Z80 `assembler` instruction `CP b`.

Source file: <src/lib/assembler.fs>.

cp,

```
cp, ( reg -- ) "c-p-comma"
```

Compile the Z80 `assembler` instruction `CP reg`.

See also: `tstp,`, `cpl,`.

Source file: <src/lib/assembler.fs>.

cpir,

```
cpir, ( -- ) "c-p-i-r-comma"
```

Compile the Z80 `assembler` instruction `CPIR`.

See also: `cp,`, `ldir,`, `djnz,`.

Source file: <src/lib/assembler.fs>.

cpl,

```
cpl, ( -- ) "c-p-l-comma"
```

Compile the Z80 `assembler` instruction `CPL`.

See also: `scf,`, `ccf,`, `neg,`, `and,`, `cp,`.

Source file: <src/lib/assembler.fs>.

cpx,

```
cpx, ( disp regpi --  ) "c-p-x-comma"
```

Compile the Z80 `assembler` instruction `CP (regpi+disp)`.

See also: `addx,`, `adcx,`, `subx,`, `sbcx,`, `andx,`, `xorx,`, `orx,`, `incx,`, `decx,`.

Source file: <src/lib/assembler.fs>.

cr

```
cr ( -- ) "c-r"
```

Transmit a carriage return to the selected output device.

`cr` is a deferred word (see `defer`) whose default action is `(cr`.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

Source file: <src/kernel.z80s>.

create

```
create ( "name" -- )
```

Parse *name*. Create a definition for *name*. After *name* is created, the data-space pointer (returned by `here`), points to the first byte of *name*'s data field. When *name* is subsequently executed, the address of the first byte of *name*'s data field is left on the stack.

`create` does not allocate data space in *name*'s data field. Reservation of data field space is tipically done with `allot`.

The execution semantics of *name* may be expanded by using `does>`.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `,`, `c,`, `2,`.

Source file: <src/kernel.z80s>.

create-entry

```
create-entry ( i*x wid xt "name" -- )
```

Create an entry *name* in the `associative-list` *wid*, using *xt* to store its value *i*x*.

`create-entry` is a factor of `entry:`, `centry:`, `2entry:` and `sentry:`.

Source file: <src/lib/data.associative-list.fs>.

create-fid

```
create-fid ( -- fid )
```

Create a new file identifier *fid* and link its structure to the previous one, pointed by `latest-fid`. Also update `latest-fid`.

See also: `latest-fid`, `/fid`, `~fid-link`.

Source file: <src/lib/dos.gplusdos.fs>.

create:

```
create: ( "name" -- ) "create-colon"
```

Create a word *name* which is compiled as a colon word but, when executed, will return the address of its data field address.

Source file: <src/lib/define.MISC.fs>.

creset

```
creset ( b ca -- ) "c-reset"
```

Reset the bits at *ca* specified by the bitmask *b*.

`creset` is written in Z80. Its equivalent definition in Forth is the following.

```
: creset ( b ca -- ) tuck c@ swap invert and swap c! ;
```

See also: `cset`, `ctoggle`, `c@and`.

Source file: <src/kernel.z80s>.

crnd

```
crnd ( -- b ) "c-r-n-d"
```

Return a random 8-bit number *b* (0..255).

See also: `rnd`.

Source file: <src/lib/random.fs>.

crs

```
crs ( n -- ) "c-r-s"
```

Emit *n* number of cr characters (character code 13).

See also: `cr`, `'cr'`.

Source file: <src/lib/display.control.fs>.

cs-drop

```
cs-drop ( C: x -- ) "c-s-drop"
```

Remove *x* from the control-flow stack.

`cs-drop` is a `compile-only` word.

NOTE In Solo Forth the control-flow stack is implemented using the data stack.

See also: `cs-pick`, `cs-roll`, `cs-swap`, `cs-dup`, `cs-mark`, `cs-test`.

Source file: <src/lib/flow.stack.fs>.

cs-dup

```
cs-dup ( C: x -- x x ) "c-s-dup"
```

Duplicate *x* on the control-flow stack.

`cs-dup` is a `compile-only` word.

NOTE In Solo Forth the control-flow stack is implemented using the data stack.

See also: `cs-pick`, `cs-roll`, `cs-swap`, `cs-drop`, `cs-mark`, `cs-test`.

Source file: <src/lib/flow.stack.fs>.

cs-mark

```
cs-mark ( C: -- cs-mark ) "c-s-mark"
```

Place a marker *cs-mark* on the control-flow stack. The marker ocuppies the same width as an *orig|dest* but is distinguishable using `cs-test`.

See also: `cs-pick`, `cs-roll`, `cs-swap`, `cs-dup`, `cs-drop`.

Source file: <src/lib/flow.stack.fs>.

cs-pick

```
cs-pick "c-s-pick"
  ( S: u -- )
  ( C: x#u ... x#1 x#0 -- x#u ... x#1 x#0 x#u )
```

Remove *u*. Copy *x#u* to the top of the control-flow stack.

`cs-pick` is a `compile-only` word.

NOTE In Solo Forth the control-flow stack is implemented using the data stack.

Origin: Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

See also: `cs-roll`, `cs-swap`, `cs-drop`, `cs-dup`, `cs-mark`, `cs-test`.

Source file: <src/lib/flow.stack.fs>.

cs-roll

```
cs-roll "c-s-roll"
  ( S: u -- )
  ( C: x#u x#u-1 ... x#0 -- x#u-1 ... x#0 x#u )
```

Remove *u*. Rotate *u+1* items on top of the control-flow stack so that *x#u* is on top of the control-flow stack.

`cs-roll` is a `compile-only` word.

NOTE In Solo Forth the control-flow stack is implemented using the data stack. Therefore `cs-roll` is an `alias` of `roll`.

Origin: Forth-94 (TOOLS EXT), Forth-2012 (TOOLS EXT).

See also: `cs-pick`, `cs-swap`, `cs-drop`, `cs-dup`, `cs-mark`, `cs-test`.

Source file: <src/lib/flow.stack.fs>.

cs-swap

```
cs-swap "c-s-swap"
  ( C: orig#1|dest#1 orig#2|dest#2 -- orig#2|dest#2 orig#1|dest#1 )
```

Exchange the top two control-flow stack items.

`cs-swap` is a `compile-only` word.

NOTE In Solo Forth the control-flow stack is implemented using the data stack. Therefore `cs-swap` is an `alias` of `swap`.

See also: `cs-pick`, `cs-roll`, `cs-drop`.

Source file: <src/kernel.z80s>.

cs-test

```
cs-test "c-s-test"
  Compilation: ( -- f ) ( C: x -- x )
```

Return a true flag if *x* is an *orig*|*dest*, and false if a marker placed by `cs-mark`.

See also: `cs-pick`, `cs-roll`, `cs-swap`, `cs-dup`, `cs-drop`.

Source file: <src/lib/flow.stack.fs>.

cset

```
cset ( b ca -- ) "c-set"
```

Set the bits at *ca* specified by the bitmask *b*.

`cset` is written in Z80. Its equivalent definition in Forth is the following.

```
: cset ( b ca -- ) tuck c@ or swap c! ;
```

See also: `creset`, `ctoggle`, `c@and`.

Source file: <src/kernel.z80s>.

cslit

```
cslit ( -- ca ) "c-s-lit"
```

Return a string that is compiled after the calling word, and adjust the instruction pointer to step over the inline string.

`cslit` is compiled by `csliteral`.

See also: `slit`.

Source file: <src/lib/strings.c-quote.fs>.

csliteral

```
csliteral
  Compilation: ( ca1 len1 -- )
  Run-time:    ( -- ca2 )
"c-s-literal"
```

Compile `cslit` and string *ca1 len1* in the current definition. At run-time `cslit` will return string *ca1 len1* as a counted string *ca2*.

`csliteral` is an `immediate` and `compile-only` word.

See also: `sliteral`.

Source file: <src/lib/strings.c-quote.fs>.

csp

```
csp ( -- a ) "c-s-p"
```

A `user` variable. *a* is the address of a cell containing the current data stack position saved by `!csp`.

Origin: fig-Forth.

Source file: <src/kernel.z80s>.

csprite

```
csprite ( width height a "name..." -- ) "c-sprite"
```

Parse a character sprite and store it at *a*. *width* and *height* are in characters. The maximum *width* is 7 (imposed by the size of Forth source blocks). *height* has no maximum, as the UDG block can ocuppy more than one Forth block (provided the Forth block has no index line, i.e. `load-program` is used to load the source).

The scans can be formed by binary digits, by the characters hold in `udg-blank` and `udg-dot`, or any combination of both notations.

The difference with `udg-block` and `,udg-block` is `csprite` stores the graphic by whole scans, not by characters.

Usage example:

```
create ship-sprite 3 2 * /udg* allot
3 2 ship-sprite csprite

..XX.X.X........X.X.XX..
..XXX.X.X......X.X.XXX..
..XX.....X....X.....XX..
...XX.....XXXX.....XX...
....XX.....XX.....XX....
.....XXX........XXX.....
......XX........XX......
.......XX......XX.......
.......XX......XX.......
........XX....XX........
........XX....XX........
X.........XXXX.........X
X........XXXXXX........X
.XXXXXXXXXXXXXXXXXXXXXX.
..........XXXX..........
...........XX...........
```

Source file: <src/lib/graphics.udg.fs>.

cstorer

```
cstorer ( c ca "name" -- ) "c-storer"
```

Define a word *name* which, when executed, will cause that *c* be stored at *ca*.

Origin: variant of the word `set` found in Forth-79 (Reference Word Set) and Forth-83 (Appendix B. Uncontrolled Reference Words).

Source file: <src/lib/data.storer.fs>.

cswitch

```
cswitch ( c switch -- ) "c-switch"
```

Execute the switch *switch* for the key *c*.

See also: `switch:`, `:cclause`.

Source file: <src/lib/flow.switch-colon.fs>.

ctoggle

```
ctoggle ( b ca -- ) "c-toggle"
```

Invert the bits at *ca* specified by the bitmask *b*.

See also: `cset`, `creset`, `c@and`.

Source file: <src/lib/memory.MISC.fs>.

ctoval

```
ctoval ( -- ) "c-to-val"
```

Change the default behaviour of words created by `cval`: make them store a new value instead of returning its actual one.

See also: `toval`, `2toval`.

Source file: <src/lib/data.val.fs>.

current

```
current ( -- a )
```

A `user` variable. *a* is the address of a cell containing the word list identifier of the compilation word list.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (System Extension Word Set).

See also: `get-current`.

Source file: <src/kernel.z80s>.

current-channel

```
current-channel ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the number of the latest output channel set by `channel`.

Source file: <src/kernel.z80s>.

current-latest

```
current-latest ( -- nt )
```

nt is the name token of the topmost word in the current compilation word list.

Definition:

```
: current-latest ( -- nt ) get-current @ ;
```

Origin: fig-Forth's `latest`.

See also: `latest`, `fyi`.

Source file: <src/kernel.z80s>.

current-mode

```
current-mode ( -- a )
```

A `variable`. *a* is the address of a cell containing the execution token of the word that activates the current screen mode (e.g. `mode-32`, `mode-32iso`, `mode-42pw`, `mode-42rs`, `mode-64es`, `mode-64ao`). It's set to `noop` until the first mode change is done.

See also: `save-mode`, `restore-mode`.

Source file: <src/kernel.z80s>.

current-window

```
current-window ( -- a )
```

A `variable`. *a* is the address of a cell containing the address of the `current-window`.

See also: `wx`, `wy`, `wx0`, `wy0`, `wcolumns`, `wrows`.

Source file: <src/lib/display.window.fs>.

cursor-char

```
cursor-char ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the character code of the cursor used by `xkey`. Note this is a character variable, thus it has to be fetched with `c@` and modified with `c!`.

Source file: <src/kernel.z80s>.

cval

```
cval ( c "name" -- ) "c-val"
```

Create a definition for *name* that will place *c* on the stack (unless `ctoval` is used first) and then will execute `init-cval`.

See also: `val`, `2val`, `cvariable`, `cconstant`.

Source file: <src/lib/data.val.fs>.

cvalue

```
cvalue ( c "name" -- ) "c-value"
```

Create a definition *name* with initial value *c*. When *name* is later executed, *c* will be placed on the stack. `to` can be used to assign a new value to *name*.

See also: `value`, `2value`, `cconstant`, `cvariable`, `cval`.

Source file: <src/lib/data.value.fs>.

cvariable

```
cvariable ( "name" -- ) "c-variable"
```

Create a character variable *name* and reserve one character of data space. When *name* is executed, it returns the address of the reserved space.

See also: `c!`, `c@`, `variable`.

Source file: <src/lib/data.MISC.fs>.

cyan

```
cyan ( -- b )
```

A `cconstant` that returns 5, the value that represents the cyan color.

See also: `black`, `blue`, `red`, `magenta`, `green`, `yellow`, `white`, `contrast`, `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

d

d

```
d ( -- reg )
```

Return the identifier *reg* of the Z80 `assembler` register "D", which is interpreted as register pair "DE" by `assembler` words that use register pairs (for example `ldp,`).

See also: `a`, `b`, `c`, `e`, `h`, `l`, `m`, `ix`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

d

```
d ( -- )
```

A command of `gforth-editor`: `delete` marked area.

See also: `dl`, `m a`, `h`, `f`, `r`, `y`, `l`.

Source file: <src/lib/prog.editor.gforth.fs>.

d

```
d ( n -- )
```

A command of `specforth-editor`: Delete line *n* but hold it in `pad`. Line 15 becomes free as all statements move up one line.

See also: `b`, `c`, `e`, `f`, `h`, `i`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `x`.

Source file: <src/lib/prog.editor.specforth.fs>.

d*

```
d* ( d|ud1 d|ud2 -- d|ud3 ) "d-star"
```

Multiply *d1|ud1* by *d2|ud2* giving the product *d3|ud3*.

See also: `ud*`, `um*`, `m*`, `*`.

Source file: <src/lib/math.operators.2-cell.fs>.

d+

```
d+ ( d1|ud1 d2|ud2 -- d3|ud3 ) "d-plus"
```

Add *d2|ud2* to *d1|ud1*, giving the sum *d3|ud3*.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `d-`, `+`, `dmax`.

Source file: <src/kernel.z80s>.

d-

```
d- ( d1|ud1 d2|ud2 -- d3|ud3 ) "d-minus"
```

Subtract *d2|ud2* from *d1|ud1*, giving the difference *d3|ud3*.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `d+`, `-`, `dmin`.

Source file: <src/lib/math.operators.2-cell.fs>.

d.

```
d. ( d -- ) "d-dot"
```

Display *d* according to current base, followed by one blank.

Origin: fig-Forth, Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `ud.`, `.`, `f.`.

Source file: <src/kernel.z80s>.

d.r

```
d.r ( d n -- ) "d-dot-r"
```

Display *d* right aligned in a field *n* characters wide. If the number of characters required to display *d* is greater than *n*, all digits are displayed with no leading spaces in a field as wide as necessary.

Definition:

```
: d.r ( d n -- ) >r d>str r> over - spaces type ;
```

Origin: fig-Forth, Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set)[6], Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `d>str`, `ud.r`, `.r`, `0d.r`, `<#`.

Source file: <src/kernel.z80s>.

d0<

```
d0< ( d -- f ) "d-zero-less"
```

f is true if and only if *d* is less than zero.

See also: `0<`.

Source file: <src/lib/math.operators.2-cell.fs>.

d0=

```
d0= ( d -- f ) "d-zero-equals"
```

f is true if and only if *d* is equal to zero.

`d0=` is written in Z80. Its equivalent definition in Forth is the following:

```
: d0= ( d -- f ) + 0= ;
```

See also: `0=`.

Source file: <src/lib/math.operators.2-cell.fs>.

d10*

```
d10* ( ud1 -- ud2 ) "d-ten-star"
```

Multiply *ud1* per 10, resulting *ud2*.

See also: `d2*`, `d*`, `2*`, `8*`.

Source file: <src/lib/math.operators.2-cell.fs>.

d2*

```
d2* ( xd1 -- xd2 ) "d-two-star"
```

xd2 is the result of shifting *xd1* one bit toward the most-significant bit, filling the vacated bit with zero.

Origin: Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `d2/`, `2*`, `lshift`.

Source file: <src/lib/math.operators.2-cell.fs>.

d2/

```
d2/ ( xd1 -- xd2 ) "d-two-slash"
```

xd2 is the result of shifting *xd1* one bit toward the least-significant bit, leaving the most-significant bit unchanged.

Origin: Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `d2*`, `2/`, `rshift`.

Source file: <src/lib/math.operators.2-cell.fs>.

d<

```
d< ( d1 d2 -- f ) "d-less"
```

f is true only if and only if *d1* is less than *d2*.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE EXT), Forth-2012 (DOUBLE EXT).

See also: `du<`, `<`, `dmin`.

Source file: <src/lib/math.operators.2-cell.fs>.

d<>

```
d<> ( xd1 xd2 -- f ) "d-not-equals"
```

f is true if and only if *xd1* is not bit-for-bit the same as *xd2*.

See also: `<>`.

Source file: <src/lib/math.operators.2-cell.fs>.

d=

```
d= ( xd1 xd2 -- f ) "d-equals"
```

f is true if and only if *xd1* is equal to *xd2*.

See also: `=`.

Source file: <src/lib/math.operators.2-cell.fs>.

d>s

```
d>s ( d -- n ) "d-to-s"
```

n is the equivalent of *d*. The high cell of *d* is discarded.

Origin: Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `s>d`, `u>ud`.

Source file: <src/kernel.z80s>.

d>str

```
d>str ( d>str -- ca len ) "d-to-s-t-r"
```

Convert *d* to string *ca len* in the pictured numeric output string buffer.

`d>str` is a factor of `d.r`.

Definition:

```
: d>str ( d -- ca len ) tuck dabs <# #s rot sign #> ;
```

WARNING	The pictured numeric output string buffer is a transient region (with maximum size `/hold` bytes below `pad`, and start pointed by `hld`). Therefore the string *ca len* must be used or preserved before the buffer is moved or overwritten.

See also: `<#`, `#s`, `sign` ,`#>`, `>stringer`, `s,`, `cmove`.

Source file: <src/kernel.z80s>.

d>str

```
d>str ( d -- ca len ) "d-to-s-t-r"
```

Convert *d* to string *ca len*.

See also: `n>str`, `ud>str`, `char>string`.

Source file: <src/lib/strings.MISC.fs>.

daa,

```
daa, ( -- ) "d-a-a-comma"
```

Compile the Z80 `assembler` instruction `DAA`.

Source file: <src/lib/assembler.fs>.

dabs

```
dabs ( d -- ud ) "d-abs"
```

Leave the absolute value *ud* of a double number *d*.

Definition:

```
: dabs ( d -- ud ) dup ?dnegate ;
```

Source file: <src/kernel.z80s>.

dand

```
dand ( xd1 xd2 -- xd3 ) "d-and"
```

xd3 is the bit-by-bit logical "and" of *xd1* and *xd2*.

See also: `and`, `dor`, `dxor`.

Source file: <src/lib/math.operators.2-cell.fs>.

data

```
data ( n "name" -- n orig )
```

Create a definition for *name*, in order to compile data items of *n* bytes each, finished by `end-data`. Leave *n* and *orig* to be consumed by `end-data`. When *name* is executed, it will leave the start address of the data and the number of items, which depends on *n*.

Usage example:

```
cell data my-cells ( -- a u )
  1 , 2 , 3 , 4 , 5 ,  end-data

2 cells data my-double-cells ( -- a u )
  0. 2, 1. 2, 2. 2,  end-data

1 chars data my-characters ( -- a u )
  'a' c, 'b' c, 'c' c,  end-data
```

Source file: <src/lib/data.data.fs>.

date

```
date ( -- a )
```

a is the address of a 3-cell table containing the date used by `set-date` and `get-date`, with the following structure:

```
+0 day   (1 byte)
+1 month (1 byte)
+2 year  (1 cell)
```

See also: `set-date`, `get-date`.

Source file: <src/lib/time.fs>.

dec,

```
dec, ( reg -- ) "dec-comma"
```

Compile the Z80 `assembler` instruction `DEC reg`.

See also: `decp,`, `inc,`.

Source file: <src/lib/assembler.fs>.

dec.

```
dec. ( n -- ) "dec-dot"
```

Display *n* as a signed decimal number, followed by a space.

Origin: Gforth.

See also: `hex.`, `bin.`, `.`.

Source file: <src/kernel.z80s>.

decimal

```
decimal ( -- )
```

Set contents of `base` to ten.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `hex`, `binary`.

Source file: <src/kernel.z80s>.

decp,

```
decp, ( regp -- ) "dec-p-comma"
```

Compile the Z80 `assembler` instruction `DEC regp`.

See also: `incp,`, `dec,`.

Source file: <src/lib/assembler.fs>.

decx,

```
decx, ( disp regpi --  ) "dec-x-comma"
```

Compile the Z80 `assembler` instruction `DEC (regp+disp)`.

See also: `addx,`, `subx,`, `sbcx,`.

Source file: <src/lib/assembler.fs>.

default-bank

```
default-bank ( -- )
```

Page in the default memory `bank`, wich can be configured with `default-bank#`, at $C000 .. $FFFF.

See also: `banks`.

Source file: <src/kernel.z80s>.

default-bank#

```
default-bank# ( -- ca ) "default-bank-number-sign"
```

A `constant`. *ca* is the address of a byte containing the value of the default `bank` paged in at $C000 .. $FFFF. Its default value is zero.

See also: `banks`, `far-banks`.

Source file: <src/kernel.z80s>.

default-bank_

```
default-bank_ ( -- a ) "default-bank-underscore"
```

Return address *a* of a routine that pages in the default bank. This is the routine `default-bank` runs into, after pushing IX on the return stack to force a final return to `next`.

Output of the routine: A and E corrupted.

See also: `e-bank_`.

Source file: <src/lib/memory.far.fs>.

default-break-key?

```
default-break-key? ( -- f ) "default-break-key-question"
```

f is true if the default break key (Shift+Space) is pressed. `default-break-key?` is the default action of the deferred word `break-key?` (see `defer`).

Source file: <src/kernel.z80s>.

default-colors

```
default-colors ( -- )
```

Set the screen colors to the default values.

See also: `default-display`, `default-mode`, `default-font`.

Source file: <src/kernel.z80s>.

default-display

```
default-display ( -- )
```

Set the default values of the display: mode, font and colors. `default-display` is executed by `cold`.

Definition:

```
: default-display ( -- )
  default-mode default-font default-colors ;
```

See also: `default-mode`, `default-font`, `default-colors`.

Source file: <src/kernel.z80s>.

default-first-locatable

```
default-first-locatable ( -- a )
```

A `variable`. *a* is the address of a cell containing the default number of the first block to be searched by `located` and its descendants.

See also: `first-locatable`.

Source file: <src/lib/002.need.fs>.

default-font

```
default-font ( -- )
```

Set the default font, which is the ROM font, by setting the system variable `os-chars` to 15360 ($3C00).

See also: `set-font`, `rom-font`, `default-display`, `default-mode`, `default-colors`.

Source file: <src/kernel.z80s>.

default-graphic-ascii-char

```
default-graphic-ascii-char ( -- c )
```

A character constant. *c* is the default ASCII graphic character used by `>graphic-ascii-char`. The value can be changed with `c!>`.

Source file: <src/lib/chars.fs>.

default-header

```
default-header ( -- )
```

Set `header` to its default action: `input-stream-header`.

Definition:

```
: default-header ( -- )
  ['] input-stream-header ['] header defer! ;
```

Source file: <src/kernel.z80s>.

default-mode

```
default-mode ( -- )
```

A deferred word (see `defer`) that activates the default screen mode. It's set to `noop` until the first mode change is done. Then it's vectored to `mode-32`. It's used by `bye` and `cold`.

See also: `reset-default-mode`, `defer`, `default-display`, `default-font`, `default-colors`.

Source file: <src/kernel.z80s>.

default-of

```
default-of
  Compilation: ( C: -- of-sys )
  Run-time:    ( x -- )
```

An alternative to mark the default clause of a `case` structure.

Compilation:

Put *of-sys* onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *of-sys*, such as `endof`.

Run-time:

Discard *x* and continue execution.

`default-of` is an `immediate` and `compile-only` word.

Usage example:

```
: test ( x -- )
  case
    1 of        ." one"   endof
    2 of        ." two"   endof
    default-of ." other" endof
  endcase ;
```

Source file: <src/lib/flow.case.fs>.

default-option

```
default-option ( "name" -- )
```

Set the default option *name* of an `options[ … ]options` structure. It can be anywhere inside the structure.

See `options[` for a usage example.

Source file: <src/lib/flow.options-bracket.fs>.

default-stringer

```
default-stringer  ( -- )
```

Set the default values of `stringer` and `/stringer`. `default-stringer` is executed by `cold`.

Source file: <src/kernel.z80s>.

default-udg-chars

```
default-udg-chars ( -- ) "default-u-d-g-chars"
```

A phoney word used only to do `need default-udg-chars` in order to define UDG 144..164 as letters 'A'..'U', copied from the ROM font, the shape they have in Sinclair BASIC by default. The current value of `os-udg` is used.

WARNING In Solo Forth `os-udg` points to bitmap of UDG 0, while in Sinclair BASIC it points to bitmap of UDG 144.

See also: `block-chars`, `set-udg`, `rom-font`.

Source file: <src/lib/graphics.udg.fs>.

default-ufia

```
default-ufia ( -- ) "default-u-f-i-a"
```

Set the default value of `ufia`, which is `ufia0`.

Source file: <src/lib/dos.gplusdos.fs>.

defer

```
defer ( "name" -- )
```

Create a deferred word *name*, whose action can be configured with `defer!` or `is`. The default action of *name* is `(defer`.

Origin: Forth-2012 (CORE EXT).

See also: `defer@`, `action-of`, `>action`.

Source file: <src/kernel.z80s>.

defer!

```
defer! ( xt1 xt2 -- ) "defer-store"
```

Set the deferred word *xt2* to execute *xt1*.

Origin: Forth-2012 (CORE EXT).

See also: `defer@`, `defer`, `>action`.

Source file: <src/kernel.z80s>.

defer@

```
defer@ ( xt1 -- xt2 ) "defer-fetch"
```

Return the word *xt2* currently associated to the deferred word *xt1*.

Origin: Forth-2012 (CORE EXT).

See also: `defer!`, `defer`, `>action`.

Source file: <src/lib/define.deferred.fs>.

deferred

```
deferred ( xt "name" -- )
```

Create a deferred word *name* that will execute *xt*. Therefore `xt deferred name` is equivalent to `defer name xt ' name defer!`.

See also: `defer`, `defer!`.

Source file: <src/lib/define.deferred.fs>.

deferred?

```
deferred? ( xt -- f ) "deferred-question"
```

Is *xt* a deferred word?

NOTE The code of a deferred word starts with a Z80 jump ($C3) to the word it's associated to. This is what `deferred?` checks.

See also: `defer`, `defer@`, `action-of`.

Source file: <src/lib/define.deferred.fs>.

defers

```
defers
  Interpretation: ( "name" -- )
  Compilation:    ( "name" -- )
  Run-time:       ( -- )
```

Compile the present contents of the deferred word *name* into the current definition. I.e. this produces static binding as if *name* was not deferred.

`defers` is an `immediate` word.

Origin: Gforth.

See also: `defer`, `defer@`, `action-of`, `compile,`.

Source file: <src/lib/define.deferred.fs>.

defined

```
defined ( "name" -- nt | 0 )
```

Parse *name* and find its definition. If the definition is not found after searching all the word lists in the search order, return zero. If the definition is found, return its *nt*.

Definition:

```
: defined ( "name" -- nt | 0 ) parse-name find-name ;
```

See also: `undefined?`, `[defined]`, `parse-name`, `find-name`.

Source file: <src/kernel.z80s>.

defined?

```
defined? ( ca len -- f ) "defined-question"
```

Find name *ca len*. If the definition is found after searching the active search order, return true, else return false.

See also: `undefined?`, `defined`, `find-name`.

Source file: <src/lib/parsing.fs>.

defines

```
defines ( xt class "name" -- )
```

Bind *xt* to the selector *name* in class *class*.

Source file: <src/lib/objects.mini-oof.fs>.

definitions

```
definitions ( -- )
```

Make the compilation word list the same as the first word list in the search order. The names of subsequent definitions will be placed in the compilation word list. Subsequent changes in the search order will not affect the compilation word list.

Definition:

```
: definitions ( -- ) context @ set-current ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (SEARCH), Forth-2012 (SEARCH).

See also: `context`, `set-current`, `wordlist`, `vocabulary`.

Source file: <src/kernel.z80s>.

delapsed

```
delapsed ( d1 -- d2 ) "d-elapsed"
```

For the time *d1* in `dticks` return the elapsed time *d2* since then, also in `dticks`.

See also: `dtimer`, `elapsed`, `dticks>seconds`, `dticks>cs`, `dticks>ms`.

Source file: <src/lib/time.fs>.

delete

```
delete ( n -- )
```

A command of `specforth-editor`: Delete *n* characters prior to the cursor.

See also: `#lag`, `r#`, `#lead`.

Source file: <src/lib/prog.editor.specforth.fs>.

delete

```
delete ( ca1 len1 len2 -- )
```

Delete *len2* characters at the start of string *ca1 len1*, moving the rest of the string to the left (*ca1*) and filling the end with blanks.

See also: `insert`, `replace`.

Source file: <src/lib/strings.MISC.fs>.

delete-file

```
delete-file ( ca len -- ior )
```

Delete the disk file named in the string *ca len* and return the I/O result code *ior*.

Origin: Forth-94 (FILE), Forth-2012 (FILE).

See also: `(delete-file`.

Source file: <src/lib/dos.gplusdos.fs>.

delimited

```
delimited ( ca1 len1 -- ca2 len2 )
```

Add one leading space and one trailing space to string *ca1 len1*, returning the result *ca2 len2* in the `stringer`.

Source file: <src/lib/002.need.fs>.

depth

```
depth  ( -- +n )
```

+*n* is the number of single-cell values contained in the data stack before +*n* was placed on the stack.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `sp@`, sp0`, `cell`, `rdepth`, `fdepth`, `.depth`.

Source file: <src/lib/tool.list.stack.fs>.

device

```
device ( -- ca )
```

Return address *ca* of the device in the current `ufia`, containing 'D' or 'd'.

NOTE In G+DOS, this UFIA field is called "lstr1".

See also: `dstr1`, `fstr1`, `sstr1`, `nstr1`, `nstr2`, `hd00`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

dfalign

```
dfalign ( -- ) "d-f-align"
```

If the data space is not double-float aligned, reserve enough space to make it so.

In Solo Forth, `dfalign` does nothing: it's an `immediate` `alias` of `noop`.

Origin: Forth-94 (FLOATING EXT), Forth-2012 (FLOATING EXT).

See also: `dfaligned`, `falign`, `sfalign`, `float`.

Source file: <src/lib/math.floating_point.rom.fs>.

dfaligned

```
dfaligned ( a -- fa ) "d-f-aligned"
```

fa is the first double-float-aligned address greater than or equal to *a*

In Solo Forth, `dfaligned` does nothing: it's an `immediate` `alias` of `noop`.

Origin: Forth-94 (FLOATING EXT), Forth-2012 (FLOATING EXT).

See also: `dfalign`, `faligned`, `sfaligned`, `float`.

Source file: <src/lib/math.floating_point.rom.fs>.

dfca

```
dfca ( -- a ) "d-f-c-a"
```

a is the address of G+DOS DFCA (Disk File Channel Area) in the Plus D memory.

See also: `ufia1`, `ufia2`.

Source file: <src/lib/dos.gplusdos.fs>.

dfor

```
dfor "d-for"
  Compilation: ( R: -- dest )
  Run-time: ( ud -- )
```

Start of a `dfor`..`dstep` loop, that will iterate *ud+1* times, starting with *du* and ending with 0.

`dfor` is an `immediate` and `compile-only` word.

The current value of the index can be retrieved with `dfor-i`.

See also: `for`, `dtimes`, `?do`, `executions`.

Source file: <src/lib/flow.dfor.fs>.

dfor-i

```
dfor-i ( -- d ) "d-for-i"
```

Return the current index *d* of a `dfor` loop.

Source file: <src/lib/flow.dfor.fs>.

dhz>bleep

```
dhz>bleep ( frequency duration1 -- duration2 pitch ) "decihertz-to-bleep"
```

Convert *frequency* (in dHz, i.e. tenths of hertzs) and *duration1* (in ms) to the parameters *duration2 pitch* needed by `bleep`.

See also: `hz>bleep`.

Source file: <src/lib/sound.48.fs>.

di,

```
di, ( -- ) "d-i-comma"
```

Compile the Z80 `assembler` instruction `DI`.

See also: `ei,`, `im1,`, `im2,`, `halt,`.

Source file: <src/lib/assembler.fs>.

digit?

```
digit? ( c n -- u true | false ) "digit-question"
```

Convert the ascii character *c* (using base *n*) to its binary equivalent *u*, accompanied by a true flag. If the conversion is invalid, leave only a false flag.

Origin: fig-Forth’s `digit`.

Source file: <src/kernel.z80s>.

dip

```
dip ( x1 x2 -- x2 x2 )
```

This word is defined in Z80. Its equivalent definition in Forth is the following:

```
: dip ( x1 x2 -- x2 x2 ) nip dup ;
```

See also: `nip`, `dup`, `tuck`, `drup`.

Source file: <src/lib/data_stack.fs>.

discard-key

```
discard-key ( -- )
```

Wait for a valid key and discard it.

Source file: <src/kernel.z80s>.

disk-buffer

```
disk-buffer ( -- a )
```

A `constant`. *a* is the address of the disk `buffer`. The cell stored at *a* is the disk buffer identifier.

See also: `buffer-data`.

Source file: <src/kernel.z80s>.

display-char-bitmap_

```
display-char-bitmap_ ( -- a ) "display-char-bitmap-underscore"
```

Return address *a* of a Z80 routine that displays the bitmap of a character at given cursor coordinates.

Input registers:

- HL = address of the character bitmap
- B = y coordinate (0..23)
- C = x coordinate (0..31)

Source file: <src/lib/graphics.udg.fs>.

display>tape-file

```
display>tape-file ( ca len -- ) "display-to-tape-file"
```

Write the display memory into a tape file *ca len*.

See also: `tape-file>display`, `>tape-file`.

Source file: <src/lib/tape.fs>.

djnz,

```
djnz, ( a -- ) "d-j-n-z-comma"
```

Compile the Z80 `assembler` instruction `DJNZ n`, being *n* an offset from the current address to address *a*.

See also: `?jr,`, `dec,`.

Source file: <src/lib/assembler.fs>.

dl

```
dl ( -- )
```

A command of `gforth-editor`: `delete` a line at the cursor position.

See also: `d c m r`, `y`, `l`.

Source file: <src/lib/prog.editor.gforth.fs>.

dmax

```
dmax ( d1 d2 -- d3 ) "d-max"
```

d3 is the lesser of *d1* and *d2*.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `dmin`, `max`, `umax`.

Source file: <src/lib/math.operators.2-cell.fs>.

dmin

```
dmin ( d1 d2 -- d3 ) "d-min"
```

d3 is the greater of *d1* and *d2*.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `dmax`, `min`, `umin`.

Source file: <src/lib/math.operators.2-cell.fs>.

dnegate

```
dnegate ( d1 -- d2 ) "d-negate"
```

Negate *d1*, giving its arithmetic inverse *d2*.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `negate`, `?dnegate`.

Source file: <src/kernel.z80s>.

do

```
do
  Compilation: ( -- do-sys )
```

Compile `(do` and leave *do-sys* to be consumed by `loop` or `+loop`.

`do` is an `immediate` and `compile-only` word.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `?do`, `-do`.

Source file: <src/lib/flow.do.fs>.

do>

```
do> "do-from"
  Compilation: ( C: dest -- orig dest )
```

Part of the `{do` control structure.

Source file: <src/lib/flow.dijkstra.fs>.

docolon

```
docolon ( -- a ) "do-colon"
```

A `constant`. *a* is the address of the colon interpreter.

`dolocon` is used by `does>`.

Source file: <src/kernel.z80s>.

doer

```
doer ( "name" -- )
```

Define a word *name* whose action is configurable. By default *name* executes `doer-noop`, which does nothing.

The action of *name* can be changed by `make`.

NOTE `doer` is superseded by the standard word `defer`.

Source file: <src/lib/flow.doer.fs>.

doer-noop

```
doer-noop ( -- )
```

Do nothing. `does-noop` is an empty colon definition which is the default action of words created by `doer`.

Source file: <src/lib/flow.doer.fs>.

does>

```
does> "does"
  Compilation: ( -- )
  Run-time:    ( -- ) ( R: nest-sys -- )
```

Define the execution-time action of a word created by a high-level defining word. Used in the form:

```
: namex ... create ... does> ... ;

namex name
```

where `create` could be also any user defined word which executes `create`.

`does>` marks the termination of the defining part of the defining word *namex* and then begins the definition of the execution-time action for words that will later be defined by *namex*. When *name* is later executed, the address of *name*'s parameter field is placed on the stack and then the sequence of words between `does>` and `;` are executed.

`does>` is an `immediate` and `compile-only` word.

Definition:

```
: does> \ Compilation: ( -- )
        \ Run-time:    ( -- ) ( R: nest-sys -- )
  postpone (;code docolon call, ; immediate compile-only
```

Detailed description:

Compilation:

Append the run-time semantics below to the current definition. Append the initiation semantics given below to the current definition.

Run-time:

Replace the execution semantics of the most recent definition, referred to as *name*, with the *name* execution semantics given below. Return control to the calling definition specified by *nest-sys1*.

Initiation: `( i*x -- i*x dfa ) ( R: -- nest-sys2 )`

Save information *nest-sys2* about the calling definition. Place *name*'s data field address *dfa* on the stack. The stack effects *i*x* represent arguments to name.

name execution: `( i*x -- j*x )`

Execute the portion of the definition that begins with the initiation semantics appended by the `does>` which modified *name*. The stack effects *i*x* and *j*x* represent arguments to and results from *name*, respectively.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `;`, `;code`, `create`, `(;code`, `call,`.

Source file: <src/kernel.z80s>.

don’t

```
don't ( n1 n2 -- | n1 n2 )
```

If *n1* equals *n2*, remove them and exit the definition that called `don’t`, else leave *n1 n2* on the stack.

`don’t` is a `compile-only` word.

`don’t` is intended to be used before `do`, as an alternative to `?do`, when the do-loop structure is factored in its own word.

Usage example:

```
: (.range ( n1 n2 -- ) don't do i . loop ;
: .range ( n1 n2 -- ) (.range ;
```

`don’t` is superseded by the standard word `?do`.

Source file: <src/lib/flow.MISC.fs>.

dor

```
dor ( xd1 xd2 -- xd3 ) "d-or"
```

xd3 is the bit-by-bit inclusive-or of *xd1* and *xd2*.

See also: `or`, `dxor`, `dand`.

Source file: <src/lib/math.operators.2-cell.fs>.

dos

```
dos ( -- ca len )
```

Return the name of the DOS in string *ca len*. It can be "+3DOS", "G+DOS" or "TR-DOS".

See also: `g+dos`, `tr-dos`, `+3dos`.

Source file: <src/kernel.z80s>.

dos-in

```
dos-in ( -- )
```

Page in the Plus D memory.

See also: `dos-out`, `dos-in,`, `@dos`, `!dos`.

Source file: <src/lib/dos.gplusdos.fs>.

dos-in,

```
dos-in, ( -- ) "dos-in-comma"
```

Compile the Z80 instruction `in a,(231)`, which pages in the Plus D memory.

See also: `dos-out,`, `dos-in`.

Source file: <src/lib/dos.gplusdos.fs>.

dos-out

```
dos-out ( -- )
```

Page out the Plus D memory.

See also: `dos-in`, `dos-out,`.

Source file: <src/lib/dos.gplusdos.fs>.

dos-out,

```
dos-out, ( -- ) "dos-out-comma"
```

Compile the Z80 instruction `out (231),a`, which pages out the Plus D memory.

See also: `dos-in,`, `dos-out`.

Source file: <src/lib/dos.gplusdos.fs>.

dos-vars

```
dos-vars ( -- a )
```

Address of the G+DOS variables in the Plus D memory.

See also: `@dosvar`, `c@dosvar`, `!dosvar`, `c!dosvar`.

Source file: <src/lib/dos.gplusdos.fs>.

dosior>ior

```
dosior>ior ( dosior -- ior ) "dos-I-O-R-to-I-O-R"
```

Convert a DOS *dosior* to a Forth *ior*.

dosior = the error number returned by G+DOS commands in the AF register:

- bit 0 (Fc) = set: error; unset: no error
- bits 8-14 = error code
- bit 15 = set: OS error; unset: G+DOS error

ior = zero (no error) or Forth exception code:

- -1031..-1000: Equivalent to G+DOS error codes 0..31.
- -1154..-1128: Equivalent to OS error codes 0..27.

Definition:

```
: dosior>ior ( dosior -- ior )
  dup 1 and negate          \ error?
  swap flip %11111111 and   \ get upper 8 bits
  1000 + negate and ;
```

Source file: <src/kernel.gplusdos.z80s>.

dovocabulary

```
dovocabulary ( -- ) "do-vocabulary"
```

Change the behaviour of the latest word defined: Replace the first word list in the search order with the *wid* stored in its body.

Definition:

```
: dovocabulary ( -- ) does> ( -- ) ( dfa ) @ context ! ;
```

See also: `vocabulary`, `wordlist>vocabulary`, `wordlist`.

Source file: <src/kernel.z80s>.

do}

```
do} "do-curly-bracket"
  Compilation: ( C: orig dest -- )
  Run-time:    ( -- )
```

Terminate a `{do` control structure.

Source file: <src/lib/flow.dijkstra.fs>.

dp

```
dp ( -- a ) "d-p"
```

A `user` variable. *a* is the address of a cell containing the data-space pointer. The value may be read by `here` and altered by `there` and `allot`.

Origin: fig-Forth.

Source file: <src/kernel.z80s>.

dpast?

```
dpast? ( ud -- f ) "d-past-question"
```

Return true if the `dticks` clock has passed *ud*.

Usage example: The following word will execute the hypothetical word `test` for *ud* clock `dticks`:

```
: dtry ( ud -- )
  dticks + begin test 2dup dpast? until 2drop ;
```

Origin: lina's `past?`.

See also: `past?`, `delapsed`, `dtimer`.

Source file: <src/lib/time.fs>.

dpl

```
dpl ( -- a ) "d-p-l"
```

A `user` variable. *a* is the address of a cell containing the number of places after the decimal point on double-integer input conversion.

If `dpl` contains zero, the decimal point is the last character. The default value of `dpl` on single-number input is -1.

Origin: fig-Forth, Forth-83 (Uncontrolled Reference Words).

See also: `number-point?`, `>number`, `number?`.

Source file: <src/kernel.z80s>.

drive

```
drive ( c1 -- c2 )
```

Convert drive number *c1* (0 index) to actual drive identifier *c2* (DOS dependent).

`drive` is used in order to make the code portable, abstracting the DOS drive identifiers.

Usage example:

```
\ Set the second disk drive as default:

2 set-drive       \ on G+DOS only
1 set-drive       \ on TR-DOS only
'B' set-drive     \ on +3DOS only

1 drive set-drive \ on any DOS -- portable code
```

See also: `set-drive`, `first-drive`, `max-drives`.

Source file: <src/lib/dos.COMMON.fs>.

drive-unused

```
drive-unused ( c -- n ior )
```

Return unused kibibytes *n* in drive *c*, and the I/O result code *ior*.

NOTE In order to count the sectors used, G+DOS has to do a catalogue. Therefore the execution of `drive-unused` includes `acat`. This problem may be solved in a future version of Solo Forth.

See also: `drive-used`, `max-disk-capacity`, `sectors-used`, `get-drive`, `set-drive`, `unused`, `farunused`.

Source file: <src/lib/dos.gplusdos.fs>.

drive-used

```
drive-used ( c -- n ior )
```

Return the number *n* of used kibibytes in drive *c*, and the I/O result code *ior*.

NOTE In order to count the sectors used, G+DOS has to do a catalogue. Therefore the execution of `drive-used` includes `acat`. This problem may be solved in a future version of Solo Forth.

See also: `drive-unused`, `max-disk-capacity`, `sectors-used`, `get-drive`, `set-drive`, `unused`, `farunused`.

Source file: <src/lib/dos.gplusdos.fs>.

drop

```
drop ( x -- )
```

Remove *x* from the stack.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `2drop`, `nip`.

Source file: <src/kernel.z80s>.

drop-type

```
drop-type ( ca len x -- )
```

Remove *x* from the stack and display the string *ca len*.

`drop-type` is one of the possible actions of `type-right-field` and `type-center-field`.

Source file: <src/lib/display.type.fs>.

drup

```
drup ( x1 x2 -- x1 x1 )
```

This word is defined in Z80. Its equivalent definition in Forth is the following:

```
: drup ( x1 x2 -- x1 x1 ) drop dup ;
```

See also: `dup`, `tuck`, `nup`, `dip`.

Source file: <src/lib/data_stack.fs>.

dstep

```
dstep "d-step"
  Compilation: ( dest -- )
  Run-time:    ( R: ud -- ud' | )
```

`dstep` is an `immediate` and `compile-only` word.

Compilation:

Append the run-time semantics given below to the current definition. Resolve the destination *dest* of `dfor`.

Run-time:

If the loop index *ud* is zero, discard the loop parameters and continue execution after the loop. Otherwise decrement the loop index and continue execution at the beginning of the loop.

Source file: <src/lib/flow.dfor.fs>.

dstr1

```
dstr1 ( -- ca )
```

Return address *ca* of the drive in the current `ufia`, containing 1, 2 or '*'.

See also: `fstr1`, `sstr1`, `device`, `nstr1`, `nstr2`, `hd00`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

dticks

```
dticks ( -- ud ) "d-ticks"
```

Return the current count of clock ticks *ud*, which is updated by the OS.

NOTE `dticks`returns the OS frames counter, which is increased by the OS interrupts routine every 20th ms. The counter is a 24-bit value.

See also: `ticks`, `set-dticks`, `reset-dticks`, `ticks/second`, `dticks>seconds`, `bench{`.

Source file: <src/lib/time.fs>.

dticks>cs

```
dticks>cs ( d1 -- d2 ) "d-ticks-to-cs"
```

Convert clock ticks *d1* to centiseconds *d2*.

See also: `ticks>cs`, `dticks>seconds`, `dticks>ms`, `ticks/second`, `ticks`.

Source file: <src/lib/time.fs>.

dticks>ms

```
dticks>ms ( d1 -- d2 ) "d-ticks-to-ms"
```

Convert clock ticks *d1* to milliseconds *d2*.

See also: `ticks>ms`, `dticks>seconds`, `dticks>cs`, `ticks/second`, `ticks`.

Source file: <src/lib/time.fs>.

dticks>seconds

```
dticks>seconds ( d -- n ) "d-ticks-to-seconds"
```

Convert clock ticks *d* to seconds *n*.

See also: `ticks>seconds`, `dticks>cs`, `dticks>ms`, `ticks/second`, `ticks`.

Source file: <src/lib/time.fs>.

dtimer

```
dtimer ( d -- ) "d-timer"
```

For the time *d* in `dticks` display the elapsed time since then, also in `dticks`.

See also: `timer`, `delapsed`.

Source file: <src/lib/time.fs>.

dtimes

```
dtimes ( d -- ) "d-times"
```

Repeat the next compiled instruction *d* times. If *d* is zero, continue executing the following instruction.

This structure is not nestable.

Usage example:

```
: blink ( -- ) 7 0 ?do  i border  loop  0 border ;
: blinking ( -- ) 100000. dtimes blink  ." Done" cr ;
```

See also: `times`, `executions`, `dfor`, `?do`.

Source file: <src/lib/flow.times.fs>.

du/mod

```
du/mod ( ud1 ud2 -- ud3 ud4 ) "d-u-slash-mod"
```

Divide *ud1* by *ud2*, giving the remainder *ud3* and the quotient *ud4*.

See also: `um/mod`, `/mod` ,`*/mod`.

Source file: <src/lib/math.operators.2-cell.fs>.

du<

```
du< ( ud1 ud2 -- f ) "d-u-less"
```

f is true only if and only if *du1* is less than *du2*.

Origin: Forth-79 (Double Number Word Set), Forth-83 (Double Number Extension Word Set), Forth-94 (DOUBLE EXT), Forth-2012 (DOUBLE EXT).

See also: `d<`, `<`, `dmin`.

Source file: <src/lib/math.operators.2-cell.fs>.

dump

```
dump ( ca len -- )
```

Show the contents of *len* bytes from *ca*.

Source file: <src/lib/tool.dump.fs>.

dump-fs

```
dump-fs ( F: i*r -- i*r )
```

See also: `.fs`.

Source file: <src/lib/math.floating_point.rom.fs>.

dump-wordlist

```
dump-wordlist ( wid -- )
```

Dump the data of the `wordlist` identified by *wid*, with labels: its associated name (or, if none, just the *wid*) and the name of the latest definition created in the word list.

See also: `.wordlist`, `dump-wordlists`, `wordlist>last`, `.name`.

Source file: <src/lib/tool.list.word_lists.fs>.

dump-wordlists

```
dump-wordlists ( -- )
```

Dump the data of all the word lists defined in the system, starting from the `wordlist` pointed by `last-wordlist`.

See also: `dump-wordlist`, `dump-wordlists>`, `wordlists`.

Source file: <src/lib/tool.list.word_lists.fs>.

dump-wordlists>

```
dump-wordlists> ( wid -- ) "dump-wordlists-from"
```

Dump the data of all the word lists defined in the system, starting from the `wordlist` identified by *wid*.

`dump-wordlists>` is a useful factor of `dump-wordlists`.

See also: `dump-wordlist`, `wordlists`, `wordlist>link`.

Source file: <src/lib/tool.list.word_lists.fs>.

dup

```
dup ( x -- x x )
```

Duplicate *x*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `?dup`, `2dup`, `tuck`, `over`, `0dup`, `-dup`, `nup`, `3dup`, `dup>r`.

Source file: <src/kernel.z80s>.

dup>r

```
dup>r ( x -- x ) ( R: -- x ) "dup-to-r"
```

Move a copy of *x* to the return stack. `dup>r` is a faster alternative to the idiom `dup >r`.

Origin: IsForth.

See also: `dup`, >r`.

Source file: <src/lib/return_stack.fs>.

dxor

```
dxor ( xd1 xd2 -- xd3 ) "d-x-or"
```

xd3 is the bit-by-bit exclusive-or of *xd1* and *xd2*.

See also: `xor`, `dor`, `dand`.

Source file: <src/lib/math.operators.2-cell.fs>.

dzx7m

```
dzx7m ( a1 a2 -- ) "d-z-x-seven-m"
```

Decompress data, which has been compressed by ZX7, from *a1* and copy the result to *a2*.

`dzx7m` is the port of the ZX7 decompressor, "Mega" version, written by Einar Saukas.

`dzx7m` is the fastest (30% faster than `dzx7s`) but biggest (251 bytes) version of the decompressor. `dzx7s` and `dzx7t` are smaller but slower. See a comparation table in `dzx7s`.

For more information, see ZX7 in World of Spectrum.

Source file: <src/lib/decompressor.zx7.fs>.

dzx7s

```
dzx7s ( a1 a2 -- ) "d-z-x-seven-s"
```

Decompress data, which has been compressed by ZX7, from *a1* and copy the result to *a2*.

`dzx7s` is the port of the ZX7 decompressor, "Standard" version, written by Einar Saukas, Antonio Villena & Metalbrain.

`dzx7s` is the smallest but slowest version of the decompressor. `dzx7t` and `dzx7m` are bigger but faster:

Table 18. Comparation of ZX7 decompressors.

Decompressor
Size in bytes
Relative speed
dzx7s
87
100
dzx7t

Decompressor
97
125
dzx7m
251
130

For more information, see ZX7 in World of Spectrum.

Source file: <src/lib/decompressor.zx7.fs>.

dzx7t

```
dzx7t ( a1 a2 -- ) "d-z-x-seven-t"
```

Decompress data, which has been compressed by ZX7, from *a1* and copy the result to *a2*.

`dzx7t` is the port of the ZX7 decompressor, "Turbo" version, written by Einar Saukas & Urusergi.

`dzx7t` is 25% faster than `dzx7s`, and needs only 10 more bytes (97 bytes in total). `dzx7m` is bigger but faster. See a comparation table in `dzx7s`.

For more information, see ZX7 in World of Spectrum.

Source file: <src/lib/decompressor.zx7.fs>.

e

e

```
e ( -- reg )
```

Return the identifier *reg* of the Z80 `assembler` register "E".

See also: `a`, `b`, `c`, `d`, `h`, `l`, `m`, `ix`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

e

```
e ( n -- )
```

A command of `specforth-editor`: Erase line *n* with blanks.

See also: `b`, `c`, `d`, `f`, `h`, `i`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `x`, `c/l`, `blank`, `update`.

Source file: <src/lib/prog.editor.specforth.fs>.

e-bank_

```
e-bank_ ( -- a ) "e-bank-underscore"
```

Return address *a* of a routine that pages in the bank hold in the E register. This routine is a secondary entry point of `default-bank`.

- Input: E = bank
- Output: A corrupted

See also: `default-bank_`.

Source file: <src/lib/memory.far.fs>.

e>

```
e> ( a -- x ) "e-from"
```

Move *x* from the extra stack *a* defined with `estack` to the data stack.

See also: `>e`, `e@`.

Source file: <src/lib/data.estack.fs>.

e@

```
e@ ( a -- x ) "e-fetch"
```

Copy *x* from the `estack` *a* to the data stack.

See also: `e>`, `>e`.

Source file: <src/lib/data.estack.fs>.

edepth

```
edepth ( a -- n ) "e-depth"
```

Return size *n* in cells of an `estack` *a*.

Source file: <src/lib/data.estack.fs>.

edit-sound

```
edit-sound ( ca -- )
```

Start a simple editor to edit the 14-byte 128K-sound definition stored at *ca*. Instructions are displayed.

Usage example:

```
need train-sound need >body
' train-sound >body edit-sound
```

See also: `sound`, `play`.

Source file: <src/lib/prog.app.edit-sound.fs>.

editor

```
editor ( -- )
```

Replace the first entry in the search order with the word list associated to the block editor.

`editor` is a deferred word (see `defer`). Its action can be `gforth-editor` or `specforth-editor`. When any of these editors is loaded, `editor` is updated accordingly.

Source file: <src/lib/prog.editor.COMMON.fs>.

ei,

```
ei, ( -- ) "e-i-comma"
```

Compile the Z80 `assembler` instruction `EI`.

See also: `di,`, `im1,`, `im2,`, `halt,`.

Source file: <src/lib/assembler.fs>.

either

```
either ( x1 x2 x3 -- f )
```

Return `true` if *x1* equals either *x2* or *x3*; else return `false`.

Origin: IsForth.

See also: `neither`, `ifelse`, `any?`.

Source file: <src/lib/math.operators.1-cell.fs>.

elapsed

```
elapsed ( u1 -- u2 )
```

For the time *u1* in `ticks` return the elapsed time *u2* since then, also in `ticks`.

See also: `timer`, `delapsed`, `ticks>seconds`, `ticks>cs`, `ticks>ms`.

Source file: <src/lib/time.fs>.

else

```
else
  Compilation: ( C: orig1 -- orig2 )
  Run-time:    ( -- )
```

Compilation: Resolve the forward reference *orig1*, usually left by `if`. Put the location of a new unresolved forward reference *orig2* onto the control-flow stack, usually to be resolved by `then`.

Run-time: Continue execution at the location specified by the resolution of *orig2*.

`else` is an `immediate` and `compile-only` word.

Definition:

```
: else \ Compilation: ( C: orig1 -- orig2 )
       \ Run-time:    ( -- )
  ahead cs-swap then ; immediate compile-only
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `ahead`, `cs-swap`.

Source file: <src/kernel.z80s>.

emit

```
emit ( x -- )
```

If *x* is a graphic character in the character set used by the current display mode, display it. If *x* is a control character used by the current display mode, manage it.

`emit` is a deferred word (see `defer`) whose default action is `mode-32-emit`.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `current-mode`, `emit-udg`, `g-emit-udg`.

Source file: <src/kernel.z80s>.

emit-ascii

```
emit-ascii ( c -- )
```

Convert character *c* with `>graphic-ascii-char`, then `emit` it.

See also: `type-ascii`, `fartype-ascii`.

Source file: <src/lib/display.type.fs>.

emit-udg

```
emit-udg ( c|n -- ) "emit-u-d-g"
```

Display the UDG *c*|*n* from the current UDG set, which is pointed by `os-udg`.

NOTE The usual parameter is *c* (0 .. 255), but no check is done: If a 16-bit value *n* is received instead, it will be used to calculate the address of the corresponding character bitmap in the UDG set.

WARNING `emit-udg` gets the cursor position and the current screen address from the OS variables. Therefore, it works only in display modes that use the ROM printing routines and keep those variables updated, like `mode-32` and `mode-32iso`.

See also: `set-udg`, `emit-udga`, `emit`, `mode-32-emit`, `g-emit-udg`, `last-font-char`.

Source file: <src/kernel.z80s>.

emit-udga

```
emit-udga ( ca -- ) "emit-u-d-g-a"
```

Display the UDG defined at *ca*, i.e, the 8 bytes of the UDG are stored at *ca*, in the usual ZX Spectrum font/UDG format: the first byte is the top scan.

WARNING `emit-udga` gets the cursor position and the current screen address from the OS variables. Therefore, it works only in display modes that use the ROM printing routines and keep those variables updated, like `mode-32` and `mode-32iso`.

See also: `emit-udg`, `emit`, `mode-32-emit`.

Source file: <src/kernel.z80s>.

emits

```
emits ( c n -- )
```

If *n* is greater than zero, display *n* characters *c*.

Definition:

```
: emits ( c n -- ) 0 max 0 ?do dup emit loop drop ;
```

Source file: <src/kernel.z80s>.

empty-buffers

```
empty-buffers ( -- )
```

Unassign all block buffers. Do not transfer the contents of any updated block to mass storage.

Definition:

```
: empty-buffers ( -- ) $7FFF disk-buffer ! ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Controlled Reference Words), Forth-94 (BLOCK EXT), Forth-2012 (BLOCK EXT).

See also: `update`, `flush`, `disk-buffer`.

Source file: <src/kernel.z80s>.

empty-fs

```
empty-fs ( -- ) "empty-f-s"
```

Empty the floating-point stack, by storing the content of `fp0` into `fp`.

Source file: <src/lib/math.floating_point.rom.fs>.

empty-heap

```
empty-heap ( -- )
```

Empty the current `heap`, which was created by `allot-heap`, `limit-heap`, `bank-heap` or `farlimit-heap`.

`empty-heap` is a deferred word (see `defer`) whose action can be `charlton-empty-heap` or `gil-empty-heap`, depending on the `heap` implementation used by the application.

Source file: <src/lib/memory.allocate.COMMON.fs>.

empty-stack

```
empty-stack ( -- )
```

Empty the data stack.

Definition:

```
: empty-stack ( -- ) sp0 @ sp! ;
```

See also: `sp0`, `sp!`, `(abort`.

Source file: <src/kernel.z80s>.

empty-stringer

```
empty-stringer ( -- )
```

Empty the `stringer`, by initializing `+stringer` with `/stringer`. The contents of the `stringer` are not modified.

Definition:

```
: empty-stringer ( -- ) /stringer +stringer ! ;
```

See also: `default-stringer`.

Source file: <src/kernel.z80s>.

end-asm

```
end-asm ( -- )
```

Exit the `assembler` mode started by `asm`.

Definition:

```
: end-asm ( -- ) ?csp previous abase @ base ! ;
```

See also: `end-code`, `?csp`, `previous`, `abase`, `base`.

Source file: <src/kernel.z80s>.

end-calc

```
end-calc ( -- )
```

Compile the `end-calc` ROM `calculator` command:

```
db $38 ; exit the ROM calculator
```

See also: `end-calculator`.

Source file: <src/lib/math.calculator.fs>.

end-calculator

```
end-calculator ( -- )
```

Stop compiling ROM `calculator` commands: Restore the search order and compile the following assembly instructions to exit the ROM calculator:

```
db $38 ; ``end&#45;calc`` ROM calculator command
pop bc ; restore the Forth IP
```

See also: `end-calc`.

Source file: <src/lib/math.calculator.fs>.

end-calculator-flag

```
end-calculator-flag ( -- f ) ( F: 1|0 -- )
```

A Z80 `macro` that compiles code to exit the ROM calculator and convert a flag calculated by it (*1*|*0*) to a well-formed flag on the data stack.

`end-calculator-flag` is a common factor of all floating-point logical operators.

See also: `calculator-command`.

Source file: <src/lib/math.floating_point.rom.fs>.

end-class

```
end-class ( class methods vars "name" -- )
```

End the definition of a class.

Source file: <src/lib/objects.mini-oof.fs>.

end-code

```
end-code ( -- )
```

Terminate a code definition started by `code` or `;code`.

Definition:

```
: end-code ( -- ) end-asm reveal ;
```

Origin: Forth-83 (Assembler Extension Word Set).

See also: `end-asm`, `reveal`.

Source file: <src/kernel.z80s>.

end-data

```
end-data ( n orig -- )
```

Finish the definition started by `data`, calculating the number of data items of *n* bytes that were compiled and store it at *orig*.

Source file: <src/lib/data.data.fs>.

end-internal

```
end-internal ( -- a )
```

End internal (private) definitions. Return the current value of the headers pointer, which is the *xtp* (execution token pointer) of the next word defined.

The start of the internal definitions was marked by `internal`. The internal definitions can be unlinked by `unlink-internal` or hidden by `hide-internal`.

Source file: <src/lib/modules.internal.fs>.

end-module

```
end-module ( parent-wid -- )
```

End a `module` definition. All module internal words are no longer accessible. Only words that have been exported with `export` are still available.

Source file: <src/lib/modules.module.fs>.

end-package

```
end-package ( wid0 wid1 -- )
```

End the current package, which was started by `package`.

wid1 is the word list of the current package; *wid0* is the word list in which the current package was created.

Origin: SwiftForth.

See also: `public`, `private`.

Source file: <src/lib/modules.package.fs>.

end-program

```
end-program ( -- )
```

Mark the end of a program that is being loaded by `load-program`.

See also: `loading-program`.

Source file: <src/lib/blocks.fs>.

end-seclusion

```
end-seclusion ( wid1 wid2 -- )
```

End a `seclusion` module.

See also: `-seclusion`, `+seclusion`.

Source file: <src/lib/modules.MISC.fs>.

end-stringtable

```
end-stringtable ( a1 a2 -- )
```

End a named stringtable, consuming *a1* (containing the address of the strings index) and *a2* (the address of the compiled strings), which were left by `begin-stringtable`. Create the strings index by traversing the compiled strings and update its address in *a1*.

See `begin-stringtable` for a usage example.

Source file: <src/lib/data.begin-stringtable.fs>.

end-structure

```
end-structure ( struct-sys +n -- )
```

Terminate definition of a structure started by `begin-structure`.

Origin: Forth-2012 (FACILITY EXT).

Source file: <src/lib/data.begin-structure.fs>.

end-transient

```
end-transient ( -- )
```

End the transient code started by `transient`. `end-transient` must be used after compiling the transient code.

The inner operation is: Restore the old values of `dp`, `np`, `limit` and `farlimit`.

See also: `forget-transient`.

Source file: <src/lib/modules.transient.fs>.

end?ccase

```
end?ccase "end-question-case"
  Compilation: ( C: orig -- )
  Run-time: ( -- )
```

End of a `?ccase` control structure. See `?ccase` for a usage example.

`end?ccase` is an `immediate` and `compile-only` word.

Source file: <src/lib/flow.ccase.fs>.

endcase

```
endcase
  Compilation: ( C: 0 orig#1 ... orig#n -- )
  Run-time:    ( x -- )
```

`endcase` is an `immediate` and `compile-only` word.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `thens`.

Source file: <src/lib/flow.case.fs>.

endccase

```
endccase "end-c-case"
  Compilation: ( C: orig1 orig2 -- )
  Run-time: ( -- )
```

End of a `ccase` control structure. See `ccase` for a usage example.

`endccase` is an `immediate` and `compile-only` word.

Source file: <src/lib/flow.ccase.fs>.

endccase0

```
endccase0 "end-c-case-zero"
  Compilation: ( C: orig -- )
  Run-time: ( -- )
```

End of a `ccase0` control structure. See `ccase0` for a usage example.

`endcase0` is an `immediate` and `compile-only` word.

Source file: <src/lib/flow.ccase.fs>.

endm

```
endm ( -- ) "end-m"
```

Finish the definition of an `assembler` `macro`.

`endm` is an `immediate` word.

See also: `end-asm`, `code`.

Source file: <src/lib/assembler.macro.fs>.

endof

```
endof
  Compilation: ( C: orig1 -- orig2 )
  Run-time:    ( -- )
```

Compilation: Mark the end of an `of` clause (or any of its variants) of the `case` structure. Resolve the forward reference *orig1*, usually left by `of`. Put the location of a new unresolved forward reference *orig2* onto the control-flow stack, usually to be resolved by `endcase`.

Run-time: Continue execution at the location specified by the consumer of *orig2*.

`endof` is equivalent to `else`.

`endof` is an `immediate` and `compile-only` word.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

Source file: <src/lib/flow.case.fs>.

entry:

```
entry: ( x wid "name" -- ) "entry-colon"
```

Create a cell entry *name* in the `associative-list` *wid*, with value *x*.

See also: `centry:`, `2entry:`, `sentry:`, `create-entry`.

Source file: <src/lib/data.associative-list.fs>.

enum

```
enum ( n "name" -- n+1 )
```

Create a constant *name* with value *n* and return *n+1*.

Usage example:

```
0 enum first
  enum second
  enum third
  enum fourth
drop
```

See also: `cenum`, `enumcell`.

Source file: <src/lib/data.MISC.fs>.

enumcell

```
enumcell ( n "name" -- n+cell ) "enum-cell"
```

Create a constant *name* with value *n* and return *n+cell*.

Usage example:

```
0 enumcell first
  enumcell second
  enumcell third
  enumcell fourth
drop
```

See also: `enum`.

Source file: <src/lib/data.MISC.fs>.

environment-wordlist

```
environment-wordlist ( -- wid )
```

A `constant`. *wid* is the identifier of the word list where the environmental queries are defined.

See also: `environment?`.

Source file: <src/lib/environment-question.fs>.

environment?

```
environment? ( ca len -- false | i*x true ) "environment-question"
```

The string *ca len* is the identifier of an environmental query. If the string is not recognized, return a false flag. Otherwise return a true flag and some information about the query.

Table 19. Environmental Query String

String	Value data type	Constant?	Meaning
/COUNTED-STRING	n	yes	maximum size of a counted string, in characters
/HOLD	n	yes	size of the pictured numeric output string buffer, in characters
/PAD	n	yes	size of the scratch area pointed to by PAD, in characters
ADDRESS-UNIT-BITS	n	yes	size of one address unit (one byte), in bits
FLOORED	flag	yes	true if floored division is the default
MAX-CHAR	u	yes	maximum value of any character in the implementation-defined character set
MAX-D	d	yes	largest usable signed double number
MAX-N	n	yes	largest usable signed integer
MAX-U	u	yes	largest usable unsigned integer
MAX-UD	ud	yes	largest usable unsigned double number
RETURN-STACK-CELLS	n	yes	maximum size of the return stack, in cells
STACK-CELLS	n	yes	maximum size of the data stack, in cells

Notes:

1. Forth-2012 designates the Forth-94 practice of using `environment?` to inquire whether a given word set is present as obsolescent. The Forth-94 environmental strings are not supported in Solo Forth.
2. In Solo Forth environment queries are also independent ordinary constants accessible by `need`.

Origin: Forth-2012 (CORE).

See also: `environment-wordlist`. `/counted-string`, `/pad`, `address-unit-bits`, `floored`, `max-char`, `max-d`, `max-n`, `max-u`, `max-ud`, `return-stack-cells`, `stack-cells`.

Source file: <src/lib/environment-question.fs>.

eol?

```
eol? ( c -- f ) "e-o-l-question"
```

If *c* is one of the characters of `newline` return `true`; otherwise return `false`.

Source file: <src/lib/display.control.fs>.

erase

```
erase ( ca len -- )
```

If *len* is greater than zero, clear all bits in each ol *len* consecutive bytes of memory beginning at *ca*.

Origin: fig-Forth, Forth-83 (Controlled Reference Words), Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `fill`, `move`.

Source file: <src/kernel.z80s>.

error

```
error ( n -- )
```

Save the `throw` code *n* into `error#`, and the current block and line into `error-pos`, to be used by `where`. Then perform the action required by `throw` code *n* as follows:

If *n* is minus-one (-1), execute `(abort`.

If *n* is minus-two (-2), perform the function `abort"`, displaying the message associated with the `abort"` that generated the `throw`.

Otherwise, execute `.error-word` and `.throw` to give information about the condition associated with the `throw` code *n*. Subsequently, execute `(abort`.

`error` is a factor of `throw`.

Definition:

```
: error ( n -- )
  dup error# !
  >in @ blk @ error-pos 2!
  dup -1 = if (abort then
  dup -2 = if space abort-message 2@ type (abort then
  .error-word .throw (abort ;
```

See also: `abort-message`.

Source file: <src/kernel.z80s>.

error#

```
error# ( -- a ) "error-number-sign"
```

A `variable`. *a* is the address of a cell containing the number of the last error issued by `error`.

See also: `error-pos`, `where`.

Source file: <src/kernel.z80s>.

error-code-warn

```
error-code-warn ( ca len -- ca len ) "warn-dot-throw"
```

If the contents of the user variable `warnings` is not zero and the word name *ca len* is already defined in the current compilation word list, display a `throw` exception #-257 ("warning: is not unique") without actually throwing an exception.

`error-code-warn` is an alternative action of the deferred word `warn` (see `defer`).

See also: `warnings`, `error-warn`, `message-warn`, `?warn`.

Source file: <src/lib/compilation.fs>.

error-pos

```
error-pos ( -- a )
```

A `2variable`. *a* is the address of a double cell containing the position of the last error issued by `error`, as follows:

- First cell = value of `blk`
- Second cell = value of `>in`

See also: `error#`, `where`.

Source file: <src/kernel.z80s>.

error-warn

```
error-warn ( ca len -- ca len )
```

If the contents of the user variable `warnings` is not zero and the word name *ca len* is already defined in the current compilation word list, `throw` an exception #-257 instead of printing a warning message.

`error-warn` is an alternative action of the deferred word `warn` (see `defer`).

See also: `warnings`, `error-code-warn`, `message-warn`, `?warn`.

Source file: <src/lib/compilation.fs>.

error>line

```
error>line ( -n1 -- n2 ) "error-to-line"
```

Convert error code -*n1* to line *n2* relative to the block that contains the error messages.

See also: `error>ordinal`.

Source file: <src/lib/exception.fs>.

error>ordinal

```
error>ordinal ( -n1 -- +n2 ) "error-to-ordinal"
```

Convert an error code *n1* to its ordinal position +*n2* in the library.

```
-n1 =    -90 ...    -1 \ Standard error codes
        -300 ...  -256 \ Solo Forth error codes
       -1024 ... -1000 \ DOS error codes
+n2 =      1 ...   146
```

See also: `error>line`.

Source file: <src/lib/exception.fs>.

errors-block

```
errors-block ( -- a )
```

A `variable`. *a* is the address of a cell containing the block that holds the error messages.

The variable is initialized during compilation with the first block that contains "Standard error codes" in its first line.

See also: `.throw-message`.

Source file: <src/lib/exception.fs>.

esc-block-chars-wordlist

```
esc-block-chars-wordlist ( -- wid )
```

Identifier of the word list that contains the escaped block characters used by the BASin IDE and other ZX Spectrum tools:

Table 20. Escaped characters defined in `esc-block-chars-wordlist`.

Escaped notation	Default character code
\<space><space>	128
\<space>'	129
\'<space>	130
\"	131
\<space>.	132
\<space>:	133
\'.	134
\':	135
\.<space>	136
\.'	137
\:<space>	138
\:'	139
\..	140
\.:	141
\:.	142
\::	143

In order to make `s\"`, `.\"` and their common factor `parse-esc-string` recognize the escaped block characters, `esc-block-chars-wordlist` must be pushed to the escaped strings search order. Example:

```
need set-esc-order
esc-standard-chars-wordlist
esc-block-chars-wordlist 2 set-esc-order

s\" \::\:.\ '\. \nNew line:\.'\:'\'.\: ..." type
```

The code of the first block character can be modified with the character variable `first-esc-block-char`.

See also: `first-esc-block-char`, `set-esc-order`, `>esc-order`, `esc-standard-chars-wordlist`, `esc-udg-chars-wordlist`, `parse-esc-string`, `s\"`, `.\"`.

Source file: <src/lib/strings.escaped.graphics.fs>.

esc-context

```
esc-context ( -- a )
```

A `variable` that holds the escaped strings search order: *a* is the address of an array of cells, whose maximum length is hold in the `max-esc-order` constant, and whose current length is hold in the `#esc-order` variable. *a* holds the word list at the top of the search order.

See also: `max-esc-order`, `>esc-order`, `get-esc-order`, `set-esc-order`.

Source file: <src/lib/strings.escaped.fs>.

esc-previous

```
esc-previous ( -- )
```

Remove the top word list (the word list that is searched first) from the escaped strings search order.

Source file: <src/lib/strings.escaped.fs>.

esc-standard-chars-wordlist

```
esc-standard-chars-wordlist ( -- wid )
```

Identifier of the word list that contains the words whose names are the standard characters that must be escaped after a backslash in strings parsed by `s\"`, `.\"` and other words.

The execution of the words defined in the word list identified by `esc-standard-chars-wordlist` returns the new character(s) on the stack (the last one at the bottom) and the count. Example of the stack effect of a escaped character that returns two characters:

```
  ( -- c[1] c[0] 2 )
```

Most of the escaped characters are translated to one character, so they are defined as double-cell constants.

Conversion rules:

Table 21. Escaped characters defined in `esc-standard-chars-wordlist`.

Escaped	Name	ASCII characters
\a	BEL (alert)	7
\b	BS (backspace)	8
\e	ESC (escape)	27
\f	FF (form feed)	12
\l	LF (line feed)	10
\m	CR/LF	13, 10
\n	newline	13

Escaped	Name	ASCII characters
\q	double-quote	34
\r	CR (carriage return)	13
\t	HT (horizontal tab)	9
\v	VT (vertical tab)	11
\z	NUL (no character)	0
\"	double-quote	34
\x<hexdigit><hexdigit>		Conversion of the two hexadecimal digits

See also: `parse-esc-string`, `set-esc-order`, `esc-standard-chars-wordlist`, `esc-block-chars-wordlist`, `esc-udg-chars-wordlist`.

Source file: <src/lib/strings.escaped.fs>.

esc-udg-chars-wordlist

```
esc-udg-chars-wordlist ( -- wid )
```

Identifier of the word list that contains the words whose names are the UDG characters ('A'..'U'), in upper case, that must be escaped after a backslash in strings parsed by `s\"`, `.\"` and other words.

The execution of the words defined in the word list identified by `esc-udg-chars-wordlist` returns the correspondent UDG character (144..164) and a 1.

In order to make `s\"`, `.\"` and their common factor `parse-esc-string` recognize the escaped UDG characters, `esc-udg-chars-wordlist` must be pushed on the escaped strings search order. Example:

```
need set-esc-order
esc-standard-chars-wordlist
esc-udg-chars-wordlist 2 set-esc-order

s\" \A\B\C\D\nNew line:\A\B\C\D..." type
```

See also: `set-esc-order`, `>esc-order`, `esc-standard-chars-wordlist`, `esc-block-chars-wordlist`.

Source file: <src/lib/strings.escaped.graphics.fs>.

estack

```
estack ( a -- ) "e-stack"
```

Init extra stack *a*. The extra stack will grow towards high memory and the required memory must be already reserved. No check is done by `estack` or the other words used to manipulate the extra stack.

Usage example:

```
create my-stack 10 cells allot
my-stack estack
100 my-stack >e
my-stack edepth .
my-stack e@ .
my-stack e> .
my-stack edepth .
```

See also: `>e`, `e@`, `e>`, `edepth`, `xstack`.

Source file: <src/lib/data.estack.fs>.

eval

```
eval ( i*x "name" -- j*x )
```

Parse and `evaluate` *name*.

`eval` is a common factor of `[const]`, `[2const]` and `[cconst]`.

See also: `parse-name`.

Source file: <src/lib/compilation.fs>.

evaluate

```
evaluate ( i*x ca len -- j*x )
```

Save the current input source specification. Store minus-one (-1) in `source-id`. Make the string described by *ca len* both the input source and input buffer, set `>in` to zero, and interpret. When the parse area is empty, restore the prior input source specification.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `interpret`, `execute-parsing`.

Source file: <src/lib/parsing.fs>.

even?

```
even? ( n -- f ) "even-question"
```

Is *n* an even number?

`even?` is written in Z80. Its equivalent definition in Forth is the following:

```
: even? ( n -- f ) 1 and 0= ;
```

See also: `odd?`.

Source file: <src/lib/math.operators.1-cell.fs>.

exaf,

```
exaf, ( -- ) "ex-a-f-comma"
```

Compile the Z80 `assembler` instruction `EX AF, AF'`.

See also: `exx,`, `exde,`.

Source file: <src/lib/assembler.fs>.

exchange

```
exchange ( a1 a2 -- )
```

Exchange the cells stored at *a1* and *a2*.

See also: `cexchange`, `!exchange`.

Source file: <src/lib/memory.MISC.fs>.

exde,

```
exde, ( -- ) "ex-de-comma"
```

Compile the Z80 `assembler` instruction `EX DE,HL`.

See also: `exaf,`, `exx,`.

Source file: <src/lib/assembler.fs>.

exec

```
exec ( "name" -- i*x )
```

Parse *name*. If *name* is the name of a word in the current search order, `execute` it; else `throw` an exception #-13 ("undefined word").

See also: `possibly`, `defined`, `name>`, `?throw`, `execute`.

Source file: <src/lib/compilation.fs>.

execute

```
execute ( i*x xt  -- j*x )
```

Execute execution token *xt*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `perform`.

Source file: <src/kernel.z80s>.

execute-hl,

```
execute-hl, ( -- ) "execute-h-l-comma"
```

Compile an `execute` with the *xt* hold in the HL register. `execute-hl,` is used to call Forth words from `code` words.

See also: `call-xt,`, `call`, `call,`, `assembler`.

Source file: <src/lib/assembler.fs>.

execute-parsing

```
execute-parsing ( ca len xt -- )
```

Make *ca len* the current input source (using `string>source`), execute *xt*, and then restore the previous input source.

See also: `evaluate`, `interpret`, `nest-source`.

Origin: Gforth.

Source file: <src/lib/parsing.fs>.

executing?

```
executing? ( -- f ) "executing-question"
```

f is true if `state` is zero, i.e. the Forth system is not in compilation state.

Definition:

```
: executing? ( -- f ) state @ 0= ;
```

See also: `?executing`.

Source file: <src/kernel.z80s>.

executions

```
executions ( xt n -- )
```

Execute *xt* *n* times.

See also: `times`, `dtimes`.

Source file: <src/lib/flow.MISC.fs>.

exit

```
exit ( -- ) ( R: nest-sys -- )
```

Return control to the calling definition, specified by *nest-sys*.

Before executing `exit` within a `loop`, a program shall discard the loop-control parameters by executing `unloop`.

`exit` is compiled by `;`. When words contain and endless loop, the space used by `exit` can be recovered using `no-exit`.

In Solo Forth `exit` can be used in interpretation mode to stop the interpretation of a block, like fig-Forth's `;s`.

Origin: fig-Forth's `;s`, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `?exit`, `0exit`, `-exit`, `+exit`.

Source file: <src/kernel.z80s>.

exitcase

```
exitcase
  Compilation: ( C: orig -- )
  Run-time:    ( -- )
```

Part of a `thiscase` structure.

Compilation: Resolve the forward reference *orig*, which was left by `ifcase`.

Run-time: exit the current definition.

`exitcase` is an `immediate` and `compile-only` word.

See also: `ifcase`, `othercase`.

Source file: <src/lib/flow.thiscase.fs>.

export

```
export ( parent-wid "name" -- parent-wid )
```

Make the word named *name* accessible outside the `module` currently defined. *name* will be still available after `end-module`.

Source file: <src/lib/modules.module.fs>.

exsp,

```
exsp, ( -- ) "ex-s-p-comma"
```

Compile the Z80 `assembler` instruction `EX (SP),HL`.

Source file: <src/lib/assembler.fs>.

extend

```
extend ( -- )
```

Change the `cold` start parameters to extend the system to its current state.

WARNING This word is experimental. See the source code for details.

See also: `system-zone`, `system-size`, `turnkey`.

Source file: <src/lib/tool.turnkey.fs>.

extended-number-point?

```
extended-number-point? ( c -- f )
"extended-number-point-question"
```

Is character *c* an extended number point? Allowed points are: plus sign, comma, hyphen, period, slash and colon, after *Forth Programmer's Handbook*.

`extended-number-point?` is an alternative action for the deferred word `number-point?`, (see `defer`) which is used in `number?`, and whose default action is `standard-number-point?`.

See also: `classic-number-point?`.

Source file: <src/lib/math.number.point.fs>.

exx,

```
exx, ( -- ) "ex-x-comma"
```

Compile the Z80 `assembler` instruction `EXX`.

See also: `exde,`, `exaf,`.

Source file: <src/lib/assembler.fs>.

f

f

```
f ( "ccc<eol>" | -- )
```

A command of `gforth-editor`: Parse *ccc*, `search` it and mark it.

See also: `m`, `l`, `fbuf`.

Source file: <src/lib/prog.editor.gforth.fs>.

f

```
f ( "ccc<eol>" -- )
```

A command of `specforth-editor`: Search forward from the current cursor position until string *ccc* is found. The cursor is left at the end of the string and the cursor line is printed. If the string is not found and error message is given and the cursor repositioned to the top of the block.

See also: `b`, `c`, `d`, `e`, `h`, `i`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `x`, `text`.

Source file: <src/lib/prog.editor.specforth.fs>.

f!

```
f! ( fa -- ) ( F: r -- ) "f-store"
```

Store *r* at *fa*.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `f@`, `f,`, `!`, `2!`, `c!`.

Source file: <src/lib/math.floating_point.rom.fs>.

f,

```
f, ( -- ) ( F: r -- ) "f-comma"
```

Reserve data space for one floating-point number and store *r* in that space.

Origin: Gforth.

See also: `f!`.

Source file: <src/lib/math.floating_point.rom.fs>.

f.

```
f. ( F: r -- )
```

See also: `.`, `d.`, `.fs`.

Source file: <src/lib/math.floating_point.rom.fs>.

f==

```
f== ( -- f ) ( F: r1 r2 -- ) "f-equals-equals"
```

Exact bitwise equality.

Are *r1* and *r2* exactly identical? Flag *f* is true if the bitwise comparison of *r1* and *r2* is succesful.

See also: `f~`.

Source file: <src/lib/math.floating_point.rom.fs>.

f>flag

```
f>flag ( -- f ) ( F: rf -- ) "f-to-flag"
```

Convert a floating-poing flag *rf* (1|0) to an actual flag *f* in the data stack.

Source file: <src/lib/math.floating_point.rom.fs>.

f@

```
f@ ( fa -- ) ( F: -- r )
```

r is the value stored at *fa*.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `f!`, `@`, `2@`, `c@`.

Source file: <src/lib/math.floating_point.rom.fs>.

facos

```
facos ( F: r1 -- r2 )
```

See also: `fasin`, `fatan`, `fcos`.

Source file: <src/lib/math.floating_point.rom.fs>.

fade-display

```
fade-display ( -- )
```

Do a screen fade to black, by decrementing the values of paper and ink in a loop.

See also: `blackout`, `attr-cls`.

Source file: <src/lib/graphics.display.fs>.

falign

```
falign ( -- ) "f-align"
```

If the data space is not float aligned, reserve enough space to make it so.

In Solo Forth, `falign` does nothing: it's an `immediate` `alias` of `noop`.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `faligned`, `sfalign`, `dfalign`, `float`.

Source file: <src/lib/math.floating_point.rom.fs>.

faligned

```
faligned ( a -- fa ) "f-aligned"
```

fa is the first float-aligned address greater than or equal to *a*

In Solo Forth, `faligned` does nothing: it's an `immediate` `alias` of `noop`.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `falign`, `sfaligned`, `dfaligned`, `float`.

Source file: <src/lib/math.floating_point.rom.fs>.

false

```
false ( -- false )
```

Return a false flag, a single-cell value with all bits clear.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `true`, `0`.

Source file: <src/kernel.z80s>.

far

```
far ( a1 -- a2 )
```

Convert a far-memory address *a1* ($0000 .. $FFFF) to its actual equivalent *a2* ($C000 .. $FFFF) and page in the corresponding memory bank, using the configuration at `far-banks`.

`far` is written in Z80. Its equivalent definition in Forth is the following:

```
: far ( a1 -- a2 )
  u>ud /bank um/mod  far-banks + c@ bank  bank-start + ;
```

See also: `far-hl_`, `bank`.

Source file: <src/kernel.z80s>.

far!

```
far! ( x a -- ) "far-store"
```

Store *x* into far-memory address *a*.

`far!` is written in Z80. Its equivalent definition in Forth is the following:

```
: far! ( x a -- )
  >r split r@ 1+ far c! r> farc! ;
```

Faster but larger definition:

```
: far! ( x a -- )
  >r split r@ 1+ far c! r> far c! default-bank ;
```

See also: `far-banks`.

Source file: <src/kernel.z80s>.

far+!

```
far+! ( n|u a -- ) "far-plus-store"
```

Add *n*|*u* to the single-cell number at far-memory address *a*.

See also: `farc+!`, `+!`, `farc!`, `far-banks`.

Source file: <src/lib/memory.far.fs>.

far,

```
far, ( x -- ) "far-comma"
```

Compile *x* in far-memory headers space.

See also: `far-n,`, `,`, `farallot`.

Source file: <src/lib/memory.far.fs>.

far,"

```
far," ( "ccc<quote>" -- ) "far-comma-quote"
```

Parse "ccc" delimited by a double-quote and compile the string in `far` memory.

See also: `fars,`, `parse`, `,"`. `s,`, `far-banks`.

Source file: <src/lib/strings.far.fs>.

far-banks

```
far-banks ( -- ca )
```

ca is the address of an array of four bytes. It holds the four memory banks used as a virtual 64-KiB continuous space, called "far memory". Every byte holds the bank number used for a 16-KiB range of addresses, as follows:

Table 22. Far-memory banks.

Offset	Address range	Bank
+0	$0000 .. $3FFF	1
+1	$4000 .. $7999	3
+2	$8000 .. $BFFF	4
+3	$C000 .. $FFFF	6

See also: `bank`, `banks`, `bank-index`, `far`, `far@`, `farc@`, `far!`, `farc!`, `farcount`, `farplace`, `fartype`, `faruppers`, `farlimit`.

Source file: <src/kernel.z80s>.

far-hl_

```
far-hl_ ( -- a ) "far-h-l-underscore"
```

Address of the `far.hl` routine of the kernel, which converts the far-memory address ($0000..$FFFF) hold in the HL register to its actual equivalent ($C000..$FFFF) and page in the corresponding memory bank.

This is the routine called by `far`. `far-hl_` is used in `code` words.

Input:

- HL = far-memory address ($0000..$FFFF)

Output:

- HL = actual memory address ($C000..$FFFF)
- A DE corrupted

Source file: <src/lib/memory.far.fs>.

far-localized,

```
far-localized, ( x[langs]..x[1] -- )
```

Store a `langs` number of cells, from *x[1]* to *x[langs]* in the far-memory name space, updating `np`.

`far-localized,` is an unused alternative to `localized,`.

Source file: <src/lib/translation.fs>.

far-localized-string

```
far-localized-string ( ca[langs]..ca[1] "name" -- )
```

Create a word *name* that will return a counted string from *ca[langs]..ca[1]*, depending on `lang`.

ca[langs]..ca[1], are the far-memory addresses where the strings have been compiled. *ca[langs]..ca[1]*, are ordered by ISO language code, being TOS the first one.

Note the string returned by *name* is in far memory, where it's compiled. Therefore the application needs `fartype` or `far>stringer` to use it. `far>localized-string` is a variant of `far-localized-string` that returns the strings already copied in the `stringer`.

See also: `far>localized-string`, `localized-string`, `localized-word`, `localized-character`, `langs`, `farcount`.

Source file: <src/lib/translation.fs>.

far-n,

```
far-n, ( x[u]..x[1] u -- ) "far-n-comma"
```

If *u* is not zero, store *u* cells *x[u]..x[1]* into far-memory headers space, being *x[1]* the first one stored and *x[u]* the last one.

See also: `far,`, `n,`, `farallot`.

Source file: <src/lib/memory.far.fs>.

far2!

```
far2! ( d a -- ) "far-two-store"
```

Store *d* into far-memory address *a*.

See also: `far2@`, `far!`, `farc!`, `far-banks`, `2!`.

Source file: <src/lib/memory.far.fs>.

far2@

```
far2@ ( a -- d ) "far-two-fetch"
```

Fetch *d* from far-memory address *a*.

See also: `far2!`, `far2@+`, `far@`, `farc@`, `far-banks`, `2@`.

Source file: <src/lib/memory.far.fs>.

far2@+

```
far2@+ ( a -- a' xd ) "far-two-fetch-plus"
```

Fetch *xd* from *a*. Return *a'*, which is *a* incremented by two cells. This is handy for stepping through double-cell arrays.

See also: `far@+`, `farc@+`, `far2@`, `2@+`. `far-banks`.

Source file: <src/lib/memory.far.fs>.

far2avariable

```
far2avariable ( n "name" -- ) "far-two-a-variable"
```

Create, in far memory, a 1-dimension double-cell variables array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — a)

Return far-memory address *a* of element *n*.

See also: `faravariable`, `farcavariable`, `2avariable`.

Source file: <src/lib/data.array.variable.far.fs>.

far>localized-string

```
far>localized-string ( ca[langs]..ca[1] "name" -- )
```

Create a word *name* that will return a counted string from *ca[langs]..ca[1]*, depending on `lang`, and copied in the `stringer`.

ca[langs]..ca[1], are the far-memory addresses where the strings have been compiled. *ca[langs]..ca[1]*, are ordered by ISO language code, being TOS the first one.

See also: `far-localized-string`, `localized-string`, `localized-word`, `localized-character`, `langs`, `farcount`, `far>stringer`.

Source file: <src/lib/translation.fs>.

far>sconstant

```
far>sconstant ( ca len "name" -- ) "far-to-s-constant"
```

Create a string constant *name* in far memory with value *ca len*.

When *name* is executed, it returns the string *ca len* in the `stringer` as *ca2 len*.

See also: `farsconstant`.

Source file: <src/lib/strings.far.fs>.

far>sconstants

```
far>sconstants ( 0 ca[n]..ca[1] "name" -- n ) "far-to-s-constants"
```

Create a table of string constants *name* in far memory, using counted strings *ca[n]..ca[1]*, being *0* a mark for the last string on the stack, and return the number *n* of compiled strings.

When *name* is executed, it converts the index on the stack (0..*n-1*) to the correspondent string *ca len* in far memory, and return a copy in the `stringer`.

Usage example:

```
0                 \ end of strings
  np@ far," kvar"  \ string 4
  np@ far," tri"   \ string 3
  np@ far," du"    \ string 2
  np@ far," unu"   \ string 1
  np@ far," nul"   \ string 0
far>sconstants digitname  constant digitnames

cr .( There are ) digitnames . .( digit names:)
0 digitname cr type
1 digitname cr type
2 digitname cr type
3 digitname cr type cr
```

See also: `sconstants`, `farsconstants`.

Source file: <src/lib/strings.far.fs>.

far>stringer

```
far>stringer ( ca1 len1 -- ca2 len1 ) "far-to-stringer"
```

Save the string *ca1 len1*, which is in far memory, to the `stringer` and return it as *ca2 len1*.

See also: `>stringer`.

Source file: <src/lib/strings.far.fs>.

far@

```
far@ ( a -- x ) "far-fetch"
```

Fetch *x* from far-memory address *a*.

`far@` is written in Z80. Its equivalent definition in Forth is the following:

```
: far@ ( a -- x )
  dup 1+ far c@ >r farc@ r> join ;
```

Faster but larger definition:

```
: far@ ( a -- x )
  dup 1+ far c@ >r far c@ r> join default-bank ;
```

See also: `far-banks`.

Source file: <src/kernel.z80s>.

far@+

```
far@+ ( a -- a' x ) "far-fetch-plus"
```

Fetch *x* from far-memory address *a*. Return *a'*, which is *a* incremented by one cell. This is handy for stepping through cell arrays.

See also: `farc@+`, `far@+`, `far2@+`, `@+`, `far-banks`.

Source file: <src/lib/memory.far.fs>.

farallot

```
farallot ( n -- ) "far-allot"
```

If *n* is greater than zero, reserve *n* bytes of headers space. If *n* is less than zero, release *n* bytes of headers space. If *n* is zero, leave the headers-space pointer unchanged.

See also: `farfill`, `far-banks`.

Source file: <src/lib/memory.far.fs>.

faravariable

```
faravariable ( n "name" -- ) "far-a-variable"
```

Create, in far memory, a 1-dimension single-cell variables array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — a)

Return far-memory address *a* of element *n*.

See also: `far2avariable`, `farcavariable`, `avariable`.

Source file: <src/lib/data.array.variable.far.fs>.

farc!

```
farc! ( c ca -- ) "far-c-store"
```

Store *c* into far-memory address *ca*.

See also: `far-banks`.

Source file: <src/kernel.z80s>.

farc+!

```
farc+! ( c ca - ) "far-c-plus-store"
```

Add *c* to the char at far-memory address *ca*.

See also: `far+!`, `c+!`, `farc!`, `far-banks`.

Source file: <src/lib/memory.far.fs>.

farc@

```
farc@ ( ca -- c ) "far-c-fetch"
```

Fetch *c* from far-memory address *ca*.

See also: `far-banks`.

Source file: <src/kernel.z80s>.

farc@+

```
farc@+ ( ca -- ca' c ) "far-c-fetch-plus"
```

Fetch the character *c* at far-memory address *ca*. Return *ca'*, which is *ca* incremented by one character. This is handy for stepping through character arrays.

See also: `far@+`, `far-banks`.

Source file: <src/lib/memory.far.fs>.

farcavariable

```
farcavariable ( n "name" -- ) "far-c-a-variable"
```

Create, in far memory, a 1-dimension character variables array *name* with *n* elements and the execution semantics defined below.

name execution:

name (n — ca)

Return far-memory address *ca* of element *n*.

See also: `faravariable`, `far2avariable`, `cavariable`.

Source file: <src/lib/data.array.variable.far.fs>.

farcount

```
farcount ( ca1 -- ca2 len2 )
```

A variant of `count` that works with far-memory addresses.

See also: `far-banks`.

Source file: <src/kernel.z80s>.

fardump

```
fardump ( ca len -- ) "far-dump"
```

Show the contents of *len* bytes from far-memory address *ca*.

See also: `farwdump`, `dump`.

Source file: <src/lib/tool.dump.fs>.

farerase

```
farerase ( ca len -- ) "far-erase"
```

If *len* is greater than zero, clear all bits in each of *len* consecutive address units of `far` memory beginning at *ca*.

See also: `farfill`, `farallot`, `far-n,`, `farc!`, `far-banks`.

Source file: <src/lib/memory.far.fs>.

farfill

```
farfill ( ca len b -- ) "far-fill"
```

If *len* is not zero, store *b* in each of *len* consecutive characters of `far` memory beginning at *a*.

See also: `farerase`, `farallot`, `far-n,`, `farc!`, `far-banks`, `fill`.

Source file: <src/lib/memory.far.fs>.

farlimit

```
farlimit ( -- a )
```

A `variable`. *a* is the address of a cell containing the address above the highest address usable by the name space (the region addressed by `np`). Its default value, which is restored by `cold`, is $0000 on G+DOS and TR-DOS, and $C000 on +3DOS.

`farlimit` can be modified by a program in order to reserve a far-memory zone for special purposes.

Origin: Fig-Forth's `limit` constant.

See also: `farunused`, `limit`, `far-banks`, `fyi`, `greeting`.

Source file: <src/kernel.z80s>.

farlimit-heap

```
farlimit-heap ( n -- a )
```

Create a `heap` of *n* bytes right above `farlimit` and return its address *a*. `farlimit` is moved down *n* bytes, and `heap-bank` is updated with the corresponding bank.

`allocate`, `resize` and `free` page in the corresponding bank at the start and restore the default bank at the end.

WARNING	The heap must be in one memory bank. Therefore, before executing `farlimit-heap`, the application must check that the *n* bytes below `farlimit` belong to one memory bank.

See also: `allot-heap`, `bank-heap`, `limit-heap`, `empty-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

farlowers

```
farlowers ( ca len -- )
```

A variant of `lowers` that works in far memory.

See also: `far-banks`.

Source file: <src/kernel.z80s>.

farplace

```
farplace ( ca1 len1 ca2 -- )
```

Store the string *ca1 len1* (which must be below memory address $C000) as a counted string at far-memory address *ca2*.

See also: `far-banks`, `place`.

Source file: <src/kernel.z80s>.

fars,

```
fars, ( ca len -- ) "fars-comma"
```

Compile a string in `far` memory.

See also: `farplace`, `farallot`, `np@`, `s,`.

Source file: <src/lib/strings.far.fs>.

farsconstant

```
farsconstant ( ca len "name" -- ) "far-s-constant"
```

Create a string constant *name* in far memory with value *ca len*.

When *name* is executed, it returns the string *ca len* in far memory as *ca2 len*.

See also: `far>sconstant`.

Source file: <src/lib/strings.far.fs>.

farsconstants

```
farsconstants ( 0 ca[n]..ca[1] "name" -- ) "far-s-constants"
```

Create a table of string constants *name* in far memory, using counted strings *ca[n]..ca[1]*, being *0* a mark for the last string on the stack, and return the number *n* of compiled strings.

When *name* is executed, it converts the index on the stack (0..*n-1*) to the correspondent string *ca len* in far memory.

Usage example:

```
0                  \ end of strings
  np@ far," kvar"  \ string 4
  np@ far," tri"   \ string 3
  np@ far," du"    \ string 2
  np@ far," unu"   \ string 1
  np@ far," nul"   \ string 0
farsconstants digitname  constant digitnames

cr .( There are ) digitnames . .( digit names:)
0 digitname cr fartype
1 digitname cr fartype
2 digitname cr fartype
3 digitname cr fartype cr
```

See also: `sconstants`, `far>sconstants`.

Source file: <src/lib/strings.far.fs>.

farsconstants,

```
farsconstants, ( 0 ca[n]..ca[1] "name" -- n ) "far-s-constants-comma"
```

Create a table of string constants *name* in far memory, using counted strings *ca[n]..ca[1]*, being *0* a mark for the last string on the stack, and return the number *n* of compiled strings.

When *name* is executed, it returns an address that holds the address of the table in far memory.

`farconstants,` is a common factor of `farsconstants` and `far>sconstants`.

Source file: <src/lib/strings.far.fs>.

farsconstants>

```
farsconstants> ( n a -- ca len ) "far-s-constants-from"
```

Return the far-memory string *ca len* whose address is stored at the *n* cell of the table *a* in data space.

`farsconstants>` is a factor of `farsconstants` and `far>sconstants`.

Source file: <src/lib/strings.far.fs>.

fartype

```
fartype ( ca len -- )
```

If *len* is greater than zero, display the character string *ca len*, which is stored in the far memory.

See also: `far-banks`, `type`, `fartype-ascii`.

Source file: <src/lib/display.type.fs>.

fartype-ascii

```
fartype-ascii ( ca len -- )
```

If *len* is greater than zero, display the string *ca len*, which is stored in far memory, using `emit-ascii` to make sure the characters are graphic ASCII characters.

See also: `fartype`, `type-ascii`.

Source file: <src/lib/display.type.fs>.

farunused

```
farunused ( -- u )
```

Return the amount of far-memory space remaining in the region addressed by `np`, in bytes.

Definition:

```
: farunused ( -- u ) farlimit @ np @ - ;
```

See also: `farlimit`, `unused`, `os-unused`, `fyi`, `greeting`.

Source file: <src/kernel.z80s>.

faruppers

```
faruppers ( ca len -- ) "far-uppers"
```

Convert string *ca len*, which is stored in far memory, to uppercase.

See also: `uppers`, `far-banks`.

Source file: <src/lib/strings.far.fs>.

farwdump

```
farwdump ( a len -- ) "far-w-dump"
```

Show the contents of *len* cells from far-memory address *a*.

See also: `fardump`, `wdump`.

Source file: <src/lib/tool.dump.fs>.

fasin

```
fasin ( F: r1 -- r2 )
```

See also: `facos`, `fatan`, `fsin`.

Source file: <src/lib/math.floating_point.rom.fs>.

fast-get-key?

```
fast-get-key? ( -- f ) "fast-get-key-question"
```

An alternative to `key?`. It works also when the system interrupts are off. Faster variant with absolute jumps.

See also: `get-key?`.

Source file: <src/lib/keyboard.get-key-question.fs>.

fast-gxy>scra_

```
fast-gxy>scra_ ( -- a ) "fast-g-x-y-to-s-c-r-a-underscore"
```

Return address *a* of a a modified copy of the PIXEL-ADD ROM routine ($22AA), to let the range of the y coordinate be 0..191 instead of 0..175.

This code is a bit faster than `slow-gxy>scra_` because the necessary jump to the ROM is saved and a useless `and a` has been removed. But in most cases the speed gain is so small (only 0.01: see `set-pixel-bench`, defined in <src/lib/meta.benchmark.MISC.fs>) that it's not worth the extra space, including the assembler.

When `fast-gxy>scra_` is loaded, it is set as the current action of `gxy>scra_`.

Input registers:

- C = x cordinate (0..255)
- B = y coordinate (0..191)

Output registers:

- HL = address of the pixel byte in the screen bitmap

- A = position of the pixel in the byte address (0..7), note: position 0=bit 7, position 7=bit 0.

See also: `gxy176>scra_`.

Source file: <src/lib/graphics.pixels.fs>.

fast-pixels

```
fast-pixels ( -- n )
```

Return the number *n* of pixels set on the screen. `fast-pixels` is the default action of `pixels`.

See also: `slow-pixels`, `bits`.

Source file: <src/lib/graphics.pixels.fs>.

fast-random

```
fast-random ( n1 -- n2 )
```

Return a random number *n2* from 0 to *n1* minus 1.

See also: `fast-rnd`, `random`.

Source file: <src/lib/random.fs>.

fast-rnd

```
fast-rnd ( -- u ) "fast-r-n-d"
```

Return a random number *u*.

`fast-rnd` generates a sequence of pseudo-random values that has a cycle of 65536 (so it will hit every single number): `f(n+1)=241f(n)+257`.

See also: `fast-random`, `rnd`.

Source file: <src/lib/random.fs>.

fatan

```
fatan ( F: r1 -- r2 )
```

See also: `facos`, `fasin`, `ftan`.

Source file: <src/lib/math.floating_point.rom.fs>.

fbuf

```
fbuf ( -- ca )
```

Return the address *ca* of the 100-byte search buffer used by the `gforth-editor`.

See also: `rbuf`, `ibuf`, `f`.

Source file: <src/lib/prog.editor.gforth.fs>.

fconstant

```
fconstant ( "name" -- ) ( F: r -- ) "f-constant"
```

Create a floating-point constant called *name* with value *r*.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `constant`, `2constant`, `cconstant`, `fvariable`.

Source file: <src/lib/math.floating_point.rom.fs>.

fcos

```
fcos ( F: r1 -- r2 )
```

See also: `fsin`, `ftan`, `facos`.

Source file: <src/lib/math.floating_point.rom.fs>.

fdepth

```
fdepth ( -- +n ) "f-depth"
```

+*n* is the number of values contained on the floating-point stack.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `fp0`, `(fp@` ,`float`, `depth`, `rdepth`.

Source file: <src/lib/math.floating_point.rom.fs>.

fetchhl

```
fetchhl ( -- a ) "fetch-h-l"
```

A `constant`. *a* is the address of a secondary entry point in the code of `@`. The code at *a* fetches the cell pointed by the HL register, pushes it onto the stack and then continues at the address returned by `next`.

See also: `pushhl`.

Source file: <src/kernel.z80s>.

fid

```
fid ( -- fid )
```

Return a usable file identifier *fid*.

See also: `latest-fid`, `find-fid`, `create-fid`.

Source file: <src/lib/dos.gplusdos.fs>.

field:

```
field: ( n1 "name" -- n2 ) "field-colon"
```

Parse *name*. *offset* is the first cell aligned value greater than or equal to *n1*. *n2* = *offset* + 1 cell.

Create a definition for *name* with the execution semantics defined below.

name execution: `( a1 -- a2 )`

Add the *offset* calculated during the compile-time action to *a1* giving the address *a2*.

Origin: Forth-2012 (FACILITY EXT).

See also: `begin-structure`, `+field`.

Source file: <src/lib/data.begin-structure.fs>.

file-exists?

```
file-exists? ( ca len -- f ) "file-exists-question"
```

If the file named in the character string *ca len* is found, *f* is `true`. Otherwise *f* is `false`.

See also: `file-status`.

Source file: <src/lib/dos.gplusdos.fs>.

file-length

```
file-length ( ca1 len1 -- len2 ior )
```

Return the file length of the file named in the character string *ca1 len1*. If the file was successfully found, *ior* is zero and *len2* is the file length. Otherwise *ior* is the I/O result code and *len2* is undefined.

See also: `file-status`.

Source file: <src/lib/dos.gplusdos.fs>.

file-start

```
file-start ( ca1 len1 -- ca2 ior )
```

Return the file start address of the file named in the character string *ca1 len1*. If the file was successfully found, *ior* is zero and *ca2* is the start address. Otherwise *ior* is the I/O result code and *ca2* is undefined.

See also: `file-status`.

Source file: <src/lib/dos.gplusdos.fs>.

file-status

```
file-status ( ca len -- a ior )
```

Return the status of the file identified by the character string *ca len*. If the file exists, *ior* is zero and *a* is the address returned by `ufia`. Otherwise *ior* is the I/O result code ond *a* is undefined.

NOTE Only the 9-byte header in `ufia` is updated, i.e. `hd00`, `hd0b`, `hd0d`, `hd0f` and `hd11`.

Origin: Forth-94 (FILE-EXT), Forth-2012 (FILE-EXT).

See also: `file-exists?`, `file-start`, `file-length`, `file-type`, `find-file`.

Source file: <src/lib/dos.gplusdos.fs>.

file-type

```
file-type ( ca len -- n ior )
```

Return the G+DOS file-type indentifier of the file named in the character string *ca len*. If the file was successfully found, *ior* is zero and *n* is the file-type identifier. Otherwise *ior* is the I/O result code and *n* is undefined.

See also: `file-status`.

Source file: <src/lib/dos.gplusdos.fs>.

file>

```
file> ( ca1 len1 ca2 len2 -- ior ) "file-from"
```

Read the contents of a disk file, whose filename is defined by the string *ca1 len1*, to memory zone *ca2 len2* (i.e. read *len2* bytes and store them starting at address *ca2*), or use the original address and length of the file instead, depending on the following rules:

1. If *len2* is not zero, use *ca2 len2*.
2. If *len2* is zero, use the original file length instead and then check also *ca2*: If *ca2* is zero, use the original file address instead.

Return the I/O result code *ior*.

Example:

The screen memory has been saved to a disk file using the following command:

```
16384 6912 s" pic.scr" >file
```

Therefore, its original address is 16384 and its original size is 6912 bytes.

Now there are several ways to load the file from disk:

Table 23. Usage examples of `file>`.

Example	Result
`s" pic.scr" 16384 6912 file>`	Load the file using its original known values
`s" pic.scr" 16384 6144 file>`	Load only the bitmap to the original known address
`s" pic.scr" 0 0 file>`	Load the file using its original unknown values
`s" pic.scr" 32768 0 file>`	Load the whole file to address 32768
`s" pic.scr" 32768 256 file>`	Load only 256 bytes to address 32768

See also: `>file`, `(file>`.

Source file: <src/lib/dos.gplusdos.fs>.

fill

```
fill ( ca len b -- )
```

If *len* is greater than zero, store *b* in each of *len* consecutive bytes of memory beginning at *ca*.

Origin: fig-Forth, Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `erase`, `farfill`, `move`.

Source file: <src/kernel.z80s>.

find

```
find ( -- )
```

A command of `specforth-editor`: Search for a match to the string at `pad`, from the cursor position until the end of block. If no match found issue an error message and reposition the cursor at the `top` of the block.

See also: `1line`.

Source file: <src/lib/prog.editor.specforth.fs>.

find

```
find ( ca -- ca 0 | xt 1 | xt -1 )
```

Find the definition named in the counted string at *ca*. If the definition is not found, return *ca* and zero. If the definition is found, return its execution token *xt*. If the definition is immediate, also return one (1), otherwise also return minus-one (-1).

Origin: Forth-83 (Required Word Set), Forth-94 (CORE, SEARCH), Forth-2012 (CORE, SEARCH).

See also: `find-name`, `find-name-from`, `find-name-in`.

Source file: <src/lib/word_lists.fs>.

find-fid

```
find-fid ( fid -- fid' )
```

Find an unused file identifier *fid'* starting from *fid*. If no unused file identifier is found, a new one is created.

See also: `free-fid?`, `~fid-link`, `create-fid`.

Source file: <src/lib/dos.gplusdos.fs>.

find-file

```
find-file ( ca len -- a | 0 )
```

If the file named in the character string *ca len* is found, update the contents of `ufia` and return its address *a*. Otherwise return zero.

See also: `file-status`.

Source file: <src/lib/dos.gplusdos.fs>.

find-name

```
find-name ( ca len -- nt | 0 )
```

Find the definition identified by the string *ca len* in the current search order. If the definition is not found after searching all the vocabularies in the search order, return zero. If the definition is found, return its *nt*.

Definition:

```
: find-name ( ca len -- nt | 0 )
  #order @ 0 ?do
    2dup context i cells + @ @ find-name-from ?dup
    ( ca len nt nt | ca len 0 )
    if  nip nip unloop exit  then ( ca len )
  loop  2drop false ;
```

Origin: Gforth.

See also: `find-name-in`, `find-name-from`, `find`.

Source file: <src/kernel.z80s>.

find-name-from

```
find-name-from ( ca len nt1 -- nt2 | 0 )
```

Find the definition named in the string *ca len*, starting at *nt1*. If the definition is found, return its *nt2*, else return zero.

String *ca len* must be below memory address $C000.

See also: `find-name`, `find-name-in`, `find`.

Source file: <src/kernel.z80s>.

find-name-in

```
find-name-in ( ca len wid -- nt | 0 )
```

Find the definition named in the string at *ca len*, in the word list identified by *wid*. If the definition is found, return its *nt*, else return zero.

See also: `search-wordlist`, `find-name-from`, `find-name`, `find`.

Source file: <src/lib/word_lists.fs>.

find-substitution

```
find-substitution ( ca len -- xt f | 0 )
```

Given a string *ca len*, find its substitution. Return *xt* and *f* if found, or just zero if not found.

See also: `replaces`.

Source file: <src/lib/strings.replaces.fs>.

finish-code

```
finish-code ( -- )
```

End the current definition, allow it to be found in the dictionary and enter interpretation state.

`finish-code` is a factor of `;` and `;code`.

Definition:

```
: finish-code ( -- )
  ?csp postpone [ noname? @ noname? off ?exit reveal ;
```

Origin: Gforth.

See also: `reveal`, `noname?`, `?csp`, `[`, `no-exit`.

Source file: <src/kernel.z80s>.

first-drive

```
first-drive ( -- c )
```

A `cconstant`. *c* is the identifier of the first drive available in the DOS.

See also: `max-drives`, `drive`.

Source file: <src/kernel.z80s>.

first-esc-block-char

```
first-esc-block-char ( -- a )
```

A `variable`. *a* is the address of a cell containing the code of the first block graphic. Its default value is 128, like in the ZX Spectrum charset. This variable can be modified in order to make the escaped block characters produce a different range of codes.

See also: `esc-block-chars-wordlist`.

Source file: <src/lib/strings.escaped.graphics.fs>.

first-locatable

```
first-locatable ( -- a )
```

A `variable`. *a* is the address of a cell containing the number of the first block to be searched by `located` and its descendants.

See also: `last-locatable`, `need-from`.

Source file: <src/lib/002.need.fs>.

first-name

```
first-name ( ca1 len1 -- ca2 len2 )
```

Return the first name *ca2 len2* from string *ca1 len1*. A name is a substring separated by spaces.

See also: `/first-name`, `last-name`, `/name`, `-prefix`, `/string`.

Source file: <src/lib/strings.MISC.fs>.

first-stream

```
first-stream ( -- n )
```

n is the number of the first stream.

See also: `last-stream`, `os-strms`, `stream>`, `stream?`.

Source file: <src/lib/os.fs>.

fit-stringer

```
fit-stringer ( len -- )
```

Make sure there's room in the `stringer` for *len* characters.

Definition:

```
: fit-stringer ( len -- )
  dup unused-stringer > if empty-stringer then
      negate +stringer +! ;
```

See also: `unused-stringer`, `empty-stringer`, `+stringer`.

Source file: <src/kernel.z80s>.

flash-mask

```
flash-mask ( -- b )
```

A `cconstant`. *b* is the bitmask of the bit used to indicate the flash status in an attribute byte.

See also: `unflash-mask`, `flashy`, `set-flash`, `attr!`, `bright-mask`, `paper-mask`, `ink-mask`.

Source file: <src/lib/display.attributes.fs>.

flash.

```
flash. ( n -- ) "flash-dot"
```

Set flash *n* by printing the corresponding control characters. If *n* is zero, turn flash off; if *n* is one, turn flash on; if *n* is eight, set transparent flash. Other values of *n* are converted as follows:

- 2, 4 and 6 are converted to 0.
- 3, 5 and 7 are converted to 1.
- Values greater than 8 or less than 0 are converted to 8.

`flash.` is much slower than `set-flash` or `attr!`, but it can handle pseudo-color 8 (transparent), setting the corresponding system variables accordingly.

See also: `bright.`, `(0-1-8-color.`.

Source file: <src/lib/display.attributes.fs>.

flashy

```
flashy ( b1 -- b2 )
```

Convert attribute *b1* to its flashy equivalent *b2*.

`flashy` is written in Z80. Its equivalent definition in Forth is the following:

```
: flashy ( b1 -- b2 ) flash-mask or ;
```

See also: `flash-mask`, `papery`, `brighty`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

flip

```
flip ( x1 -- x2 )
```

Exchange the low and high bytes within *x1*, resulting *x2*.

Origin: eForth.

NOTE `flip` is called `><` in Forth-79 (Word Reference Set) and Forth-83 (Uncontrolled Reference Words), `swab` in LaForth (c. 1980) and `cswap` in other Forth systems.

See also: `split`, `join`.

Source file: <src/kernel.z80s>.

float

```
float ( -- n )
```

n is the size in bytes of a floating-point number.

See also: `floats`, `float+`, `float-`.

Source file: <src/lib/math.floating_point.rom.fs>.

float+

```
float+ ( fa1 -- fa2 ) "float-plus"
```

Add the size in bytes of a floating-point number to *fa1*, giving *fa2*.

See also: `float-`, `float`, `floats`.

Source file: <src/lib/math.floating_point.rom.fs>.

float-

```
float- ( fa1 -- fa2 ) "float-minus"
```

Subtract the size in bytes of a floating-point number from *fa1*, giving *fa2*.

See also: `float+`, `float`, `floats`.

Source file: <src/lib/math.floating_point.rom.fs>.

floats

```
floats ( n1 -- n2 )
```

n2 is the size in bytes of *n1* floating-point numbers.

See also: `float`, `float+`, `float-`.

Source file: <src/lib/math.floating_point.rom.fs>.

floor

```
floor ( F: r1 -- r2 )
```

Round *r1* to an integral value using the "round toward negative infinity" rule, giving *r2*.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `ftrunc`, `fround`.

Source file: <src/lib/math.floating_point.rom.fs>.

floored

```
floored ( -- f )
```

f is `true` if floored division is the default.

See also: `environment?`.

Source file: <src/lib/environment-question.fs>.

flush

```
flush ( -- )
```

Perform the function of `save-buffers`, then unassign all block buffers.

Origin: Forth-83 (Required Word Set), Forth-94 (BLOCK), Forth-2012 (BLOCK).

See also: `empty-buffers`.

Source file: <src/lib/blocks.fs>.

fly-located

```
fly-located ( ca len -- block | 0 )
```

Locate the first block whose header contains the string *ca len* (surrounded by spaces), and return its number. If not found, return zero. The search is case-sensitive. Index all searched blocks on the fly.

See also: `use-fly-index`.

Source file: <src/lib/blocks.indexer.fly.fs>.

fm/mod

```
fm/mod ( d1 n1 -- n2 n3 ) "f-m-slash-mod"
```

Floored division:

```
  d1 = n3*n1+n2
  n1>n2>=0 or 0>=n2>n1
```

Divide *d1* by *n1*, giving the floored quotient *n3* and the remainder *n2*. Input and output stack arguments are signed.

Table 24. Floored Division Example

Dividend	Divisor	Remainder	Quotient
10	7	3	1
-10	7	4	-2
10	-7	-4	-2
-10	-7	-3	1

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `sm/rem`, `m/`.

Source file: <src/lib/math.operators.1-cell.fs>.

for

```
for
  Compilation: ( R: -- dest )
  Run-time:    ( u -- )
```

Start of a `for`..`step` loop, that will iterate *u+1* times, starting with *u* and ending with 0.

The current value of the index can be retrieved with `for-i`.

`for` is an `immediate` and `compile-only` word.

See also: `dfor`, `times`, `?do`, `executions`.

Source file: <src/lib/flow.for.fs>.

for-i

```
for-i ( -- n )
```

Return the current index *n* of a `for` loop.

Source file: <src/lib/flow.for.fs>.

forget-transient

```
forget-transient ( -- )
```

Forget the transient code compiled between `transient` and `end-transient`, by unlinking the header space that was reserved and used for it. `forget-transient` must be used when the transient code is not going to be used any more.

The inner operation is: Restore the old value of `last-wordlist`; store the *nt* of the latest word created before compiling the transient code, into the *lfa* of the first word created after the transient code was finished by `end-transient`.

Source file: <src/lib/modules.transient.fs>.

form

```
form ( -- cols rows )
```

Number of `columns` and `rows` in the terminal in the current display mode (e.g. `mode-32`, `mode-64ao`).

Origin: Gforth.

Source file: <src/lib/display.mode.COMMON.fs>.

form>xy

```
form>xy ( cols rows -- col row ) "form-to-x-y"
```

col row is the new cursor position corresponding to a display mode whose `form` is *cols rows*. *col row* are calculated with the values returned by `xy`, `columns` and `rows` in the current mode.

`form>xy` is a factor of `>form`.

Source file: <src/lib/display.mode.COMMON.fs>.

forth

```
forth ( -- )
```

Transform the search order consisting of *wid#n* .. *wid#2* *wid#1* (where *wid#1* is searched first) into *wid#n* .. *wid#2* *wid#f*, where *wid#f* is the word-list identifier returned by `forth-wordlist`. I.e., replace the top word list of the search order with `forth-wordlist`.

`forth` is the `vocabulary` corresponding to `forth-wordlist`.

Origin: Forth-83 (Required Word Set), Forth-94 (SEARCH EXT), Forth-2012 (SEARCH EXT).

See also: `root`, `wordlist`.

Source file: <src/kernel.z80s>.

forth-wordlist

```
forth-wordlist ( -- wid )
```

Return *wid*, the identifier of the word list that includes all standard words provided by the implementation. `forth-wordlist` is initially the compilation word list and is part of the initial search order.

Origin: Forth-94 (SEARCH), Forth-2012 (SEARCH).

See also: `wordlist`, `set-order`, `root-wordlist`, `assembler-wordlist`.

Source file: <src/kernel.z80s>.

forth2012-block-test

```
forth2012-block-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-block-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-core-test

```
forth2012-core-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-core-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-coreext-test

```
forth2012-coreext-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-coreext-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-coreplus-test

```
forth2012-coreplus-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-coreplus-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-double-test

```
forth2012-double-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-double-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-exception-test

```
forth2012-exception-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-exception-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-facility-test

```
forth2012-facility-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-facility-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-file-test

```
forth2012-file-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-file-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-locals-test

```
forth2012-locals-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-locals-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-memory-test

```
forth2012-memory-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-memory-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-report-errors

```
forth2012-report-errors ( -- )
```

Report the errors found during the latest execution of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-searchorder-test

```
forth2012-searchorder-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-searchorder-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-string-test

```
forth2012-string-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-string-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-test-suite

```
forth2012-test-suite ( -- )
```

An unexistent word. This word is used just for doing `need forth2012-test-suite` in order to run the Forth-2012 Test Suite.

The following partial tests are available: `forth2012-file-test`, `forth2012-block-test`, `forth2012-core-test`, `forth2012-coreext-test`, `forth2012-coreplus-test`, `forth2012-double-test`, `forth2012-exception-test`, `forth2012-facility-test`, `forth2012-locals-test`, `forth2012-memory-test`, `forth2012-searchorder-test`, `forth2012-string-test`, `forth2012-tools-test`, `forth2012-utilities-test`.

See also: `forth2012-report-errors`, `ttester`, `hayes-test`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-tools-test

```
forth2012-tools-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-tools-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

forth2012-utilities-test

```
forth2012-utilities-test ( -- )
```

Do nothing. This word is used just for doing `need forth2012-utilities-test` in order to run only the core test of `forth2012-test-suite`.

Source file: <src/lib/meta.test.forth2012-test-suite.fs>.

fp

```
fp  ( -- a ) "f-p"
```

Return the address *a* of a cell containing the floating-point stack pointer. *a* is the STKEND variable of the OS.

NOTE The floating-point stack (which is the OS calculator stack) grows towards higher memory, and `fp` points to the first free position, therefore above top of stack.

See also: `fp@`, `fp0`.

Source file: <src/lib/math.floating_point.rom.fs>.

fp0

```
fp0  ( -- a ) "f-p-zero"
```

Return address *a* of a cell containing the bottom address of the floating-point stack. *a* is the STKBOT variable of the OS.

NOTE The floating-point stack (which is the OS calculator stack) grows towards higher memory.

See also: `fp`.

Source file: <src/lib/math.floating_point.rom.fs>.

fp@

```
fp@ ( -- fa ) "f-p-fetch"
```

Return the address *fa* of the top of the floating-point stack.

See also: `fp`.

Source file: <src/lib/math.floating_point.rom.fs>.

free

```
free ( a -- ior )
```

Return the contiguous region of data space indicated by *a* to the system for later allocation. *a* shall indicate a region of data space that was previously obtained by `allocate` or `resize`.

If the operation succeeds, *ior* is zero. If the operation fails, *ior* is the I/O result code.

`free` is a deferred word (see `defer`) whose action can be `charlton-free` or `gil-free`, depending on the `heap` implementation used by the application.

Origin: Forth-94 (MEMORY), Forth-2012 (MEMORY).

See also: `allocate`, `resize`, `empty-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

free-buffer

```
free-buffer ( n -- )
```

If the current disk buffer has been updated, write its block to the disk. Assign block number *n* to the disk buffer.

Definition:

```
: free-buffer ( n -- )
  updated?  if    buffer-block write-buffer
            then  disk-buffer ! ;
```

Source file: <src/kernel.z80s>.

free-fid

```
free-fid ( fid -- )
```

Make file identifier *fid* free for reuse. The corresponding file is supposed to be closed already.

See also: `free-fid?`, `create-fid`.

Source file: <src/lib/dos.gplusdos.fs>.

free-fid?

```
free-fid? ( fid -- f )
```

Is file identifier *fid* free to be reused?

See also: `free-fid`, `create-fid`.

Source file: <src/lib/dos.gplusdos.fs>.

free/wtype

```
free/wtype ( ca len -- ca' len' ) "free-slash-w-type"
```

Display in the `current-window` as many characters of string *ca len* as fit in the current line, then remove them from the string, returning the result string *ca' len'*.

`free/wtype` is a factor of `wltype` and `wtype`.

See also: `/wtype`.

Source file: <src/lib/display.window.fs>.

fround

```
fround ( r1 -- r2 ) "f-round"
```

Round *r1* to an integral value using the "round to nearest" rule, giving *r2*.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `ftrunc`, `floor`.

Source file: <src/lib/math.floating_point.rom.fs>.

fsin

```
fsin ( F: r1 -- r2 )
```

See also: `fcos`, `ftan`, `fasin`.

Source file: <src/lib/math.floating_point.rom.fs>.

fstr1

```
fstr1 ( -- ca )
```

Return address *ca* of the file directory number in the current `ufia`.

See also: `dstr1`, `sstr1`, `device`, `nstr1`, `nstr2`, `hd00`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

fta,

```
fta, ( a -- ) "f-t-a-comma"
```

Compile the Z80 `assembler` instruction `LD A,(a)`, i.e. fetch the contents of memory address *a* into register "A".

See also: `sta,`, `ld,`, `ld#,`.

Source file: <src/lib/assembler.fs>.

ftan

```
ftan ( F: r1 -- r2 )
```

See also: `fcos`, `fsin`, `fatan`.

Source file: <src/lib/math.floating_point.rom.fs>.

ftap,

```
ftap, ( repg -- ) "f-t-a-p-comma"
```

Compile the Z80 `assembler` instruction `LD A,(regp)`.

See also: `stap,`.

Source file: <src/lib/assembler.fs>.

fthl,

```
fthl, ( a -- ) "f-t-h-l-comma"
```

Compile the Z80 `assembler` instruction `LD HL,(a)`, i.e. fetch the contents of memory address *a* into register pair "HL".

See also: `sthl,`, `ftp,`.

Source file: <src/lib/assembler.fs>.

ftp,

```
ftp, ( a regp -- ) "f-t-p-comma"
```

Compile the Z80 `assembler` instruction `LD regp,(a)`, i.e. fetch the contents of pair register *regp* from memory address *a*.

NOTE For the "HL" register has a specific word: `fthl,`, which compiles shorten and faster code.

See also: `stp,`.

Source file: <src/lib/assembler.fs>.

ftpx,

```
ftpx, ( disp regpi regp -- ) "f-t-p-x-comma"
```

Compile the Z80 `assembler` instructions required to fetch register pair *regp* from the address pointed by *regpi* plus *disp*.

Example: `16 ix h ftpx,` will compile the Z80 instructions `LD L,(IX+16)` and `LD H,(IX+17)`.

See also: `stpx,`, `ftx,`.

Source file: <src/lib/assembler.fs>.

ftrunc

```
ftrunc ( F: r1 -- r2 ) "f-trunc"
```

Round *r1* to an integral value using the "round toward zero" rule, giving *r2*.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `fround` ,`floor`.

Source file: <src/lib/math.floating_point.rom.fs>.

ftx,

```
ftx, ( disp regpi reg -- ) "f-t-x-comma"
```

Compile the Z80 `assembler` instruction `LD reg,(regpi+disp)`.

See also: `stx,`.

Source file: <src/lib/assembler.fs>.

fvariable

```
fvariable ( "name" -- ) ( F: -- ) "f-constant"
```

Parse *name*. `create` a definition for *name*, which is referred to as a "floating-point variable". `allot` a `float` of data space, the data field of *name*, to hold the contents of the variable. When *name* is later executed, the address of its data field is placed on the data stack.

Origin: Forth-94 (FLOATING), Forth-2012 (FLOATING).

See also: `constant`, `2constant`, `cconstant`, `fconstant`.

Source file: <src/lib/math.floating_point.rom.fs>.

fyi

```
fyi ( -- ) "f-y-i"
```

Display information about the current status of the Forth system.

See also: `#words`, `here`, `last-wordlist`, `limit`, `unused`. `np@`, `latest`, `current-latest`, `farlimit`, `farunused`, `greeting`.

Source file: <src/lib/tool.debug.fyi.fs>.

f~

```
f~ ( -- f ) ( F: r1 r2 r3 -- ) "f-tilde"
```

Medley for comparing *r1* and *r2* for equality:

- *r3*>0: `f~abs`;

- *r3*=0: `f==`;
- *r3*<0: `f~relabs`.

Origin: Forth-94 (FLOATING EXT), Forth-2012 (FLOATING EXT).

See also: `f~rel`.

Source file: <src/lib/math.floating_point.rom.fs>.

f~abs

```
f~abs ( -- f ) ( F: r1 r2 r3 -- ) "f-tilde-abs"
```

Approximate equality with absolute error: `|r1-r2|<r3`.

Flag *f* is true if the absolute value of *r1*-*r2* is less than *r3*.

Origin: Gforth.

See also: `f~rel`, `f~relabs`.

Source file: <src/lib/math.floating_point.rom.fs>.

f~rel

```
f~rel ( -- f ) ( F: r1 r2 r3 -- ) "f-tilde-rel"
```

Approximate equality with relative error: `|r1-r2|<r3*|r1+r2|`.

Flag *f* is true if the absolute value of *r1*-*r2* is less than the value of *r3* times the sum of the absolute values of *r1* and *r2*.

See also: `f~abs`, `f~relabs`.

Source file: <src/lib/math.floating_point.rom.fs>.

f~relabs

```
f~relabs ( -- f ) ( F: r1 r2 r3 -- ) "f-tilde-rel-abs"
```

Approximate equality with relative error: `|r1-r2|<|r3|*|r1+r2|`.

Flag *f* is true if the absolute value of *r1*-*r2* is less than the absolute value of *r3* times the sum of the absolute values of *r1* and *r2*.

See also: `f~rel`, `f~abs`.

Source file: <src/lib/math.floating_point.rom.fs>.

g

g

```
g ( u -- )
```

A command of `gforth-editor`: Go to screen *u*.

See also: `c`, `a`, `n`, `p`, `t`.

Source file: <src/lib/prog.editor.gforth.fs>.

g+dos

```
g+dos ( -- ) "g-plus-dos"
```

An alias of `noop` that is defined only in the G+DOS version of Solo Forth. Its goal is to check the DOS a program is running on, using `defined` or `[defined]`.

`g+dos` is an `immediate` word.

See also: `dos`, `tr-dos`, `+3dos`.

Source file: <src/kernel.z80s>.

g-at-x

```
g-at-x ( gx -- )
```

Set the current graphic x coordinate *gx*, without changing the current graphic y coordinate.

See also: `g-at-xy`, `g-at-y`.

Source file: <src/lib/graphics.coordinates.fs>.

g-at-xy

```
g-at-xy ( gx gy -- ) "g-at-x-y"
```

Set the current graphic coordinates *gx gy*.

See also: `g-xy`, `g-at-y`, `g-at-x`, `g-home`.

Source file: <src/lib/graphics.coordinates.fs>.

g-at-y

```
g-at-y ( gy -- )
```

Set the current graphic y coordinate *gy*, without changing the current graphic x coordinate.

See also: `g-at-xy`, `g-at-x`.

Source file: <src/lib/graphics.coordinates.fs>.

g-cr

```
g-cr ( -- ) "g-c-r"
```

Move the graphic coordinates to the next character row.

See also: `g-at-xy`, `g-emit`.

Source file: <src/lib/display.g-emit.fs>.

g-emit

```
g-emit ( gx gy c -- )
```

Display character *c* (32..255) at the current graphic coordinates. If *c* greater than `last-font-char` from the UDG font, otherwise it is printed from the main font.

The character is printed with overprinting (equivalent to `1 overprint`).

See also: `g-emit-udg`, `(g-emit`, `g-type`.

Source file: <src/lib/display.g-emit.fs>.

g-emit-udg

```
g-emit-udg ( c -- ) "g-emit-u-d-g"
```

Display UDG *c* (0..255) at the current graphic coordinates, from the font pointed by system variable `os-udg`, which contains the address of the first UDG bitmap (0).

The UDG character is printed with overprinting (equivalent to `1 overprint`).

See also: `g-emit`, `g-emit_`.

Source file: <src/lib/display.g-emit.fs>.

g-emit_

```
g-emit_ ( -- a ) "g-emit-underscore"
```

a is the address of a machine code routine that prints an 8x8 bits character at graphic coordinates. Used by `g-emit-udg`.

Input registers:

- DE = address of the first char (0) bitmap in a charset
- A = char code (0..255)
- B = y coordinate
- C = x coordinate

Modifies: AF BC HL IX DE

See also: `g-emit`.

Source file: <src/lib/display.g-emit.fs>.

g-emitted

```
g-emitted ( -- )
```

Update the current graphic coordinates after printing a character at them.

See `g-emit`, `g-cr`, `g-at-xy`.

Source file: <src/lib/display.g-emit.fs>.

g-home

```
g-home ( -- )
```

Set the graphic coordinates to 0, 0.

See also: `g-at-xy`.

Source file: <src/lib/graphics.coordinates.fs>.

g-type

```
g-type ( ca len -- )
```

If *len* is greater than zero, display the character string *ca len* at the current graphic coordinates.

See also: `g-emit`.

Source file: <src/lib/display.g-emit.fs>.

g-x

```
g-x ( -- gx )
```

Return the current graphic x coordinate *gx*.

See also: `g-xy`, `g-y`, `g-at-xy`.

Source file: <src/lib/graphics.coordinates.fs>.

g-xy

```
g-xy ( -- gx gy ) "g-x-y"
```

Return the current graphic coordinates *gx gy*.

See also: `g-x`, `g-y`, `g-at-xy`.

Source file: <src/lib/graphics.coordinates.fs>.

g-y

```
g-y ( -- gy )
```

Return the current graphic y coordinate *gy*.

See also: `g-xy`, `g-x`, `g-at-xy`.

Source file: <src/lib/graphics.coordinates.fs>.

gcd

```
gcd ( n1 n2 -- n3 ) "g-c-d"
```

n3 is the greatest common divisor of *n1* and *n2*.

See also: `/`, `mod`.

Source file: <src/lib/math.operators.1-cell.fs>.

get-block-drives

```
get-block-drives ( -- c#n..c#1 n | 0 )
```

Get the current configuration of block drives, as configured by `set-block-drives`.

See also: `2-block-drives`, `-block-drives`, `#block-drives`, `block-drive!`.

Source file: <src/lib/dos.COMMON.fs>.

get-bright

```
get-bright ( -- f )
```

If bright is active in the current attribute, return `true`, else return `false`.

See also: `set-bright`, `attr@`, `bright.`, `get-paper`, `get-ink`, `get-flash`, `bright-mask`.

Source file: <src/lib/display.attributes.fs>.

get-current

```
get-current ( -- wid )
```

Return *wid*, the identifier of the compilation word list.

Definition:

```
: get-current ( -- wid ) current @ ;
```

Origin: Forth-94 (SEARCH), Forth-2012 (SEARCH).

See also: `current`.

Source file: <src/kernel.z80s>.

get-date

```
get-date ( -- day month year )
```

Get the current date. The default date is 2016-01-01. It can be changed with `set-date`. The date is not updated by the system.

See also: `set-date`, `date`, `time&date`, `.date`.

Source file: <src/lib/time.fs>.

get-drive

```
get-drive ( -- n ior )
```

Get the current default drive *n* (1 or 2), i.e. the drive implied by all file and block operations. The default drive is initially 1. Return also the I/O result code *ior*.

`get-drive` is written in Z80. Its equivalent definition in Forth is the following:

```
: get-drive ( -- n ior ) dos-in $3ACE c@ dos-out false ;
```

See also: `set-drive`, `dos-in`, `dos-out`.

Source file: <src/lib/dos.gplusdos.fs>.

get-esc-order

```
get-esc-order ( -- wid[n]..wid[1] n )
```

Return the number of word lists *n* in the escaped strings search order and the word lists identifiers *wid[n]..wid[1]* identifying these word lists. *wid[1]* identifies the word list that is searched first, and *wid[n]* the word list that is searched last.

See also: `set-esc-order`, `>esc-order`.

Source file: <src/lib/strings.escaped.fs>.

get-flash

```
get-flash ( -- f )
```

If flash is active in the current attribute, return `true`, else return `false`.

See also: `set-flash`, `attr!`, `flash.`, `get-paper`, `get-ink`, `get-bright`, `flash-mask`.

Source file: <src/lib/display.attributes.fs>.

get-font

```
get-font ( -- a )
```

Get address *a* of the current font (characters 32..127), by fetching the system variable `os-chars`. *a* is the bitmap address of character 0.

See also: `set-font`, `default-font`.

Source file: <src/lib/display.fonts.fs>.

get-heap

```
get-heap ( -- a u b )
```

Get the values of the current heap: its address *a* (returned by `heap`), its size *u* (returned by `/heap`) and its bank *b* (stored in `heap-bank`).

`get-heap` and `set-heap` are useful when more than one memory heap are needed by the application.

Source file: <src/lib/memory.allocate.COMMON.fs>.

get-ink

```
get-ink ( -- b )
```

Get the ink color *b* from the current attribute.

See also: `set-ink`, `attr@`, `ink.`, `get-paper`, `get-bright`, `get-flash`, `ink-mask`.

Source file: <src/lib/display.attributes.fs>.

get-inkey

```
get-inkey ( -- 0 | c )
```

Leave the value of the key being pressed. If no key being pressed leave zero.

`get-inkey` reads the keyboard, so it works even when the keyboard is not read by an interrupts routine.

See also: `inkey`, `key`.

Source file: <src/lib/keyboard.get-inkey.fs>.

get-key?

```
get-key? ( -- f ) "get-key-question"
```

An alternative to `key?`. It works also when the system interrupts are off. Variant with relative jumps.

See also: `key?`, `fast-get-key?`.

Source file: <src/lib/keyboard.get-key-question.fs>.

get-mixer

```
get-mixer ( -- b )
```

Get the contents *b* of the mixer register of the AY-3-8912 sound generator.

> Register 7 (Mixer - I/O Enable)
>
> This controls the enable status of the noise and tone mixers for the three channels, and also controls the I/O port used to drive the RS232 and Keypad sockets.
>
> | | |
> |---|---|
> | **Bit 0** | Channel A Tone Enable (0=enabled). |
> | **Bit 1** | Channel B Tone Enable (0=enabled). |
> | **Bit 2** | Channel C Tone Enable (0=enabled). |
> | **Bit 3** | Channel A Noise Enable (0=enabled). |
> | **Bit 4** | Channel B Noise Enable (0=enabled). |
> | **Bit 5** | Channel C Noise Enable (0=enabled). |
> | **Bit 6** | I/O Port Enable (0=input, 1=output). |
> | **Bit 7** | Not used. |

See also: `set-mixer`, `-mixer`, `@sound`.

Source file: <src/lib/sound.128.fs>.

get-order

```
get-order ( -- wid#n .. wid#1 n )
```

Return the number of word lists *n* in the search order and the word lists identifiers *wid#n* .. *wid#1* identifying these word lists. *wid#1* identifies the word list that is searched first, and *wid#n* the word list that is searched last.

```
: get-order ( -- wid#n .. wid#1 n )
  #order @ dup 0 ?do
    dup i - 1- cells context + @ swap
  loop ;
```

Origin: Forth-94 (SEARCH), Forth-2012 (SEARCH).

See also: `set-order`, `>order`, `context`, `#order`.

Source file: <src/kernel.z80s>.

get-paper

```
get-paper ( -- b )
```

Get the paper color *b* from the current attribute.

See also: `set-paper`, `attr@`, `paper.`, `get-ink`, `get-bright`, `get-flash`, `paper-mask`.

Source file: <src/lib/display.attributes.fs>.

get-time

```
get-time ( -- second minute hour )
```

Return the current time.

NOTE The computer doesn't have a real clock. The OS frames counter is used instead, which is increased by the OS interrupts routine every 20th ms. The counter is a 24-bit value, so its maximum is $FFFFFF ticks of 20 ms (335544 seconds or 5592 minutes or 93 hours), then it starts again from zero.

See also: `set-time`, `time&date`, `.time`.

Source file: <src/lib/time.fs>.

get-udg

```
get-udg ( -- a ) "get-u-d-g"
```

Get address *a* of the current UDG set (characters 0..255), by fetching the system variable `os-udg`. *a* is the bitmap address of character 0.

See also: `set-udg`.

Source file: <src/lib/graphics.udg.fs>.

gforth-editor

```
gforth-editor ( -- )
```

A `vocabulary` containing a port of the Gforth block editor. When `gforth-editor` is loaded, it becomes the action of `editor`.

Table 25. Gforth block editor commands

Word	Description
`a ( -- )`	Go to marked position.
`c ( n -- )`	Move cursor by *n* chars.
`d ( -- )`	Delete marked area.
`dl ( -- )`	Delete a line at the cursor position.
`f ( "ccc<eol>" -- )`	Search *ccc* and mark it.
`g ( u -- )`	Go to screen *u*.
`h ( -- )`	Type the line of the marked area, highlighting it.
`i ( "ccc<eol>" -- )`	Insert *ccc*; if *ccc* is empty, instert the contents of the insert buffer.
`il ( -- )`	Insert a line at the cursor position..
`l ( -- )`	List current screen.
`m ( -- )`	Mark current position.
`n ( -- )`	Go to next screen.
`p ( -- )`	Go to previous screen.
`r ( "ccc<eol>" -- )`	Replace marked area.
`s ( u "ccc<eol>" -- )`	Search *ccc* until screen *u*; if *ccc* is empty, use the string of the previous search.
`t ( u "ccc<eol>"-- )`	Go to line *u* and insert *ccc*.
`y ( -- )`	Yank deleted string.

See also: `specforth-editor`.

Source file: <src/lib/prog.editor.gforth.fs>.

gigatype

```
gigatype ( ca len -- )
```

If *len* is greater than zero, display text string *ca len* using the current font, with doubled pixels (16x16 pixels per character) and modifying the characters on the fly after the style stored in `gigatype-style`. The text is combined with the current content of the screen, as if `overprint` were active. The current attribute, set by `attr!` and other words, is used to color the text.

Usage example:

```
: demo ( -- )
  cls
  8 0 ?do
    i gigatype-style c!
    17 0 i 3 * tuck at-xy s" GIGATYPE" gigatype
                    at-xy ." style "   i .
  loop
  key drop home ;
```

See also: `gigatype-title`, `set-font`, `(gigatype`, `type`.

Source file: <src/lib/display.gigatype.fs>.

gigatype-style

```
gigatype-style ( -- ca )
```

ca is the address of a byte containing the font style used by `gigatype` (0..7).

Source file: <src/lib/display.gigatype.fs>.

gigatype-title

```
gigatype-title ( ca len -- )
```

If *len1* is greater than zero, display the character string *ca len* at the center of the current row (the current column is not used), using `gigatype`.

WARNING

`gigatype` prints double-size (16x16 pixels) characters. Therefore, the maximum value of *len1* is 16 characters, but `gigatype-title` does no check. Beside, it calculates the column of the title assuming the current mode is `mode-32` (32 characters per line), which is the default one.

See also: `gigatype-style`, `type-center-field`.

Source file: <src/lib/display.gigatype.fs>.

gil-allocate

```
gil-allocate ( u -- a ior )
```

Allocate *u* bytes of contiguous data space. The data-space pointer is unaffected by this operation. The initial content of the allocated space is undefined.

If the allocation succeeds, *a* is the starting address of the allocated space and *ior* is zero.

If the operation fails, *a* does not represent a valid address and *ior* is the I/O result code #-59, the `throw` code for `allocate`.

`gil-allocate` is the action of `allocate` in the `heap` implementation based on code written by Javier Gil, whose words are defined in `gil-heap-wordlist`.

See also: `gil-free`.

Source file: <src/lib/memory.allocate.gil.fs>.

gil-empty-heap

```
gil-empty-heap ( -- )
```

Empty the current `heap`, which was created by `allot-heap`, `limit-heap`, `bank-heap` or `farlimit-heap`.

`gil-empty-heap` is the action of `empty-heap` in the memory `heap` implementation based on code written by Javier Gil, whose words are defined in `gil-heap-wordlist`.

See also: `gil-allocate`, `gil-free`.

Source file: <src/lib/memory.allocate.gil.fs>.

gil-free

```
gil-free ( a -- ior )
```

Return the contiguous region of data space indicated by *a* to the system for later allocation. *a* shall indicate a region of data space that was previously obtained by `gil-allocate`.

`gil-free` is the action of `free` in the `heap` implementation based on code written by Javier Gil, whose words are defined in `gil-heap-wordlist`.

Source file: <src/lib/memory.allocate.gil.fs>.

gil-heap-wordlist

```
gil-heap-wordlist ( -- wid )
```

wid is the word-list identifier of the word list that holds the words the memory `heap` implementation adapted from code written by Javier Gil (2007-01).

`need gil-heap-wordlist` is used to load the memory heap implementation and configure `allocate`, `free` and `empty-heap` accordingly. This implementation of the memory heap does not provide `resize`.

An alternative, bigger implementation of the memory heap is provided by `charlton-heap-wordlist`.

The actual heap must be created with `allot-heap`, `limit-heap`, `farlimit-heap` or `bank-heap`, which are independent from the heap implemention.

Source file: <src/lib/memory.allocate.gil.fs>.

graphic-ascii-char?

```
graphic-ascii-char? ( c -- f ) "graphic-ascii-char-question"
```

Is *c* a printable ASCII character, i.e. in the range 32..126?

See also: `ascii-char?`, `>graphic-ascii-char`.

Source file: <src/lib/chars.fs>.

greater-of

```
greater-of
  Compilation: ( C: -- of-sys )
  Run-time:    ( x1 x2 -- | x1 )
```

`greater-of` is an `immediate` and `compile-only` word.

Usage example:

```
: test ( x -- )
  case
    10 of         ." ten!"            endof
    15 greater-of ." greater than 15" endof
    ." less than 10 or 11 ... 15"
  endcase ;
```

See also: `case`, `less-of`, `(greater-of`.

Source file: <src/lib/flow.case.fs>.

green

```
green ( -- b )
```

A `cconstant` that returns 4, the value that represents the green color.

See also: `black`, `blue`, `red`, `magenta`, `cyan`, `yellow`, `white`, `contrast`, `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

greeting

```
greeting ( -- )
```

Display the boot message.

See also: `cold`, `fyi`.

Source file: <src/kernel.z80s>.

gx>x

```
gx>x ( gx -- col ) "g-x-to-x"
```

Convert graphic coordinate *gx* to cursor column *col*.

See also: `gy>y`, `x>gx`.

Source file: <src/lib/graphics.pixels.fs>.

gxy176>scra

```
gxy176>scra ( gx gy -- n a ) "g-x-y-176-to-s-c-r-a"
```

Return screen address *a* and pixel position *n* (0..7) of pixel coordinates *gx* (0..255) and *gy* (0..175).

See also: `gxy176>scra_`, `gxy>scra`, `xy>scra`.

Source file: <src/lib/graphics.pixels.fs>.

gxy176>scra_

```
gxy176>scra_ ( -- a ) "g-x-y-176-to-s-c-r-a-underscore"
```

Return address *a* of a routine that uses an alternative entry point to the PIXEL-ADD ROM routine

($22AA), to bypass the error check.

Input registers:

- C = x cordinate (0..255)
- B = y coordinate (0..176)

Output registers:

- HL = address of the pixel byte in the screen bitmap
- A = position of the pixel in the byte address (0..7), note: position 0=bit 7, position 7=bit 0.

See also: `gxy176>scra`, `gxy>scra_`.

Source file: <src/lib/graphics.pixels.fs>.

gxy>attra

```
gxy>attra ( gx gy -- a ) "g-x-y-to-a-t-t-r-a"
```

Convert pixel coordinates *gx gy* to their correspondent attribute address *a*.

Source file: <src/lib/graphics.pixels.fs>.

gxy>scra

```
gxy>scra ( gx gy -- n a ) "g-x-y-to-s-c-r-a"
```

Return screen address *a* and pixel position *n* (0..7) of pixel coordinates *gx* (0..255) and *gy* (0..191).

See also: `gxy>scra_`, `gxy176>scra`, `xy>scra`.

Source file: <src/lib/graphics.pixels.fs>.

gxy>scra_

```
gxy>scra_ ( -- a ) "g-x-y-to-s-c-r-a-underscore"
```

A deferred word (see `defer`) that executes `fast-gxy>scra_` or, by default, `slow-gxy>scra_`: Return address *a* of an alternative to the PIXEL-ADD ROM routine ($22AA), to let the range of the y coordinate be 0..191 instead of 0..175.

See also: `gxy176>scra_`, `xy>scra_`.

Source file: <src/lib/graphics.pixels.fs>.

gy>y

```
gy>y ( gy -- row ) "g-y-to-y"
```

Convert graphic y coordinate *gy* to cursor coordinate *row*.

See also: `gx>x`, `y>gy`.

Source file: <src/lib/graphics.pixels.fs>.

h

h

```
h ( -- reg )
```

Return the identifier *reg* of the Z80 `assembler` register "H", which is interpreted as register pair "HL" by `assembler` words that use register pairs (for example `ldp,`).

See also: `a`, `b`, `c`, `d`, `e`, `l`, `m`, `ix`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

h

```
h ( -- )
```

A command of `gforth-editor`: Type the line of the marked area, highlighting it.

See also: `m`, `a`, `d`, `f`, `r`.

Source file: <src/lib/prog.editor.gforth.fs>.

h

```
h ( n -- )
```

A command of `specforth-editor`: Hold line *n* at `pad` (used by system more often than by user).

See also: `b`, `c`, `d`, `e`, `f`, `i`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `x`.

Source file: <src/lib/prog.editor.specforth.fs>.

halt,

```
halt, ( -- ) "halt-comma"
```

Compile the Z80 `assembler` instruction `HALT`.

See also: `im1,`, `im2,`, `di,`, `ei,`.

Source file: <src/lib/assembler.fs>.

hayes-test

```
hayes-test ( -- )
```

An unexistent word. `hayes-test` is used just for doing `need hayes-test` in order to run the Hayes test, which tests the core words of an ANS Forth system.

The test assumes a two's complement implementation where the range of signed numbers is -2^(n-1) … 2^(n-1)-1 and the range of unsigned numbers is 0 … 2^(n)-1.

Some words are not tested: `key`, `quit`, `abort`, `abort"` `environment?`…

See also: `hayes-tester`, `ttester`, `forth2012-test-suite`.

Source file: <src/lib/meta.test.hayes.fs>.

hayes-tester

```
hayes-tester ( -- )
```

Do nothing. This word is used just for doing `need hayes-tester` in order to load `{`, `->`, and `}`, which are used by `hayes-test`.

Usage example:

```
{ 1 2 3 swap -> 1 3 2 } ok
{ 1 2 3 swap -> 1 2 2 } Incorrect result
Use WHERE to see the error line.
{ 1 2 3 swap -> 1 2 } Wrong number of results:
Expected=3
Actual=2
Use WHERE to see the error line.
```

See also: `ttester`, `forth2012-test-suite`.

Source file: <src/lib/meta.tester.hayes.fs>.

hd00

```
hd00 ( -- ca )
```

Return address *ca* of the file type in the current `ufia`.

See also: `dstr1`, `fstr1`, `sstr1`, `device`, `nstr1`, `nstr2`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

hd0b

```
hd0b ( -- a )
```

Return address *a* of the file length in the current `ufia`.

See also: `dstr1`, `fstr1`, `sstr1`, `device`, `nstr1`, `nstr2`, `hd00`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

hd0d

```
hd0d ( -- a )
```

Return address *a* of the file start address in the current `ufia`.

See also: `dstr1`, `fstr1`, `sstr1`, `device`, `nstr1`, `nstr2`, `hd00`, `hd0b`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

hd0f

```
hd0f ( -- a )
```

Return address *a* of the BASIC length without variables in the current `ufia`.

See also: `dstr1`, `fstr1`, `sstr1`, `device`, `nstr1`, `nstr2`, `hd00`, `hd0b`, `hd0d`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

hd11

```
hd11 ( -- a )
```

Return address *a* of the BASIC autorun line in the current `ufia`.

See also: `dstr1`, `fstr1`, `sstr1`, `device`, `nstr1`, `nstr2`, `hd00`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

header

```
header ( "name" | -- )
```

A deferred word (see `defer`) that creates a dictionary header. Its default action is `input-stream-header`, and it's set by `default-header`. Its alternative temporary action is `nextname-header`.

See also: `header,`.

Source file: <src/kernel.z80s>.

header,

```
header, ( ca len -- ) "header-comma"
```

Create a definition header in the name space using the name *ca len* and hide it by setting its `smudge` bit.

The execution token pointer of the new header points to the data space pointer.

See also: `header`, `warn`.

Source file: <src/kernel.z80s>.

heap

```
heap  ( -- a )
```

Address of the current memory heap, used by `allocate`, `resize` and `free`.

The memory heap can be created by `allot-heap`, `limit-heap`, `bank-heap`, or `farlimit-heap`. Then it must be initialized by `empty-heap`.

See also: `/heap`, `get-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

heap-bank

```
heap-bank  ( -- ca )
```

A `cvariable` *ca* that contains the memory bank used to store the `heap`, when the memory heap was created by `bank-heap` or `farlimit-heap`. If the heap was created by `allot-heap` or `limit-heap`, `heap-bank` contains zero.

See also: `heap-in`, `heap-out`, `get-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

heap-in

```
heap-in  ( -- )
```

If the current `heap` was created by `bank-heap` or `farlimit-heap`, page in its bank, which is stored at `heap-bank`; else do nothing.

`heap-in` is a deferred word (see `defer`) whose default action is `noop`. Its alternative action is `(heap-in`.

See also: `heap-out`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

heap-out

```
heap-out  ( -- )
```

If the current `heap` was created by `bank-heap` or `farlimit-heap`, page in the default memory bank instead; else do nothing.

`heap-out` is a deferred word (see `defer`) whose default action is `noop`. Its alternative action is `default-bank`.

See also: `heap-in`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

here

```
here ( -- a )
```

a is the data-space pointer.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `dp`, `limit`, `unused`, `there`.

Source file: <src/kernel.z80s>.

hex

```
hex ( -- )
```

Set contents of `base` to sixteen.

Origin: fig-Forth, Forth-79 (Reference Word Set), Forth-83 (Controlled Reference Words), Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `decimal`, `binary`.

Source file: <src/kernel.z80s>.

hex.

```
hex. ( n -- ) "hex-dot"
```

Display *n* as an unsigned hexadecimal number, followed by one space.

See also: `dec.`, `bin.`, `u.`, `.`.

Source file: <src/lib/display.numbers.fs>.

hex>

```
hex> ( -- ) "end-hex"
```

End a code zone where hexadecimal radix is the default, by restoring the value of `base` from `base'`. The zone was started by `<hex`.

Source file: <src/lib/display.numbers.fs>.

hidden

```
hidden ( nt -- )
```

Hide the definition *nt* by setting its `smudge` bit.

Definition:

```
: hidden ( nt -- ) smudge-mask swap lex! ;
```

See also: `revealed`, `hide`, `smudge-mask`, `lex!`.

Source file: <src/kernel.z80s>.

hide

```
hide ( -- )
```

Hide the `latest` definition by setting its `smudge` bit.

Definition:

```
: hide ( -- ) latest hidden ;
```

See also: `hidden`, `reveal`.

Source file: <src/kernel.z80s>.

hide-internal

```
hide-internal ( nt xtp -- )
```

Hide all words defined between the latest pair `internal` and `end-internal`, setting the `smudge` bit of their headers.

Usage example:

```
internal

: hello ( -- ) ." hello" ;

end-internal

: salute ( -- ) hello ;

hide-internal

salute  \ ok!
hello   \ error!
```

At least one word must be defined between `end-internal` and `hide-internal`.

The alternative word `unlink-internal` uses a different, simpler method: it unlinks the internal words from the dictionary.

`privatize` uses a similar method, but it has error checking and does not use the stack.

Source file: <src/lib/modules.internal.fs>.

hld

```
hld ( -- a ) "h-l-d"
```

A `user` variable. *a* is the address of a cell containing the address of the latest character of text during numeric output conversion.

Origin: fig-Forth.

See also: `hold`, `<#`, `#>`.

Source file: <src/kernel.z80s>.

hold

```
hold ( c -- )
```

Insert character *c* into a pictured numeric output string. Typically used between `<#` and `#>`.

Definition:

```
: hold ( c -- ) -1 hld +!  hld @ c! ;
```

See also: `holds`.

Source file: <src/kernel.z80s>.

holds

```
holds ( ca len -- )
```

Add string *ca len* to the pictured numeric output string started by `<#`.

Origin: Forth-2012 (CORE EXT).

See also: `hold`.

Source file: <src/lib/display.numbers.fs>.

home

```
home ( -- )
```

Set the cursor position at the top left position (column 0, row 0).

`home` is a deferred word (see `defer`), whose default action is `(home`.

See also: `at-xy`, `home?`.

Source file: <src/kernel.z80s>.

home?

```
home? ( -- f ) "home-question"
```

Is the cursor at home position (column 0, row 0)?

See also: `xy`, `home`.

Source file: <src/lib/display.cursor.fs>.

hook,

```
hook, ( -- ) "hook-comma"
```

Compile the Z80 `assembler` instruction `rst $08`. Therefore `hook,` is equivalent to `$08 rst,`.

See also: `rst,`, `prt,`.

Source file: <src/lib/assembler.fs>.

horizontal-curtain

```
horizontal-curtain ( b -- )
```

Wash the screen with the given color attribute *b* from the top and bottom rows to the middle.

See also: `vertical-curtain`.

Source file: <src/lib/graphics.cls.fs>.

hunt

```
hunt ( ca1 len1 ca2 len2 -- ca3 len3 )
```

Search a string *ca1 len1* for a substring *ca2 len2*. Return the part of *ca1 len1* that starts with the first occurence of *ca2 len2*. Therefore *ca3 len3* = *ca1+n len1-n*.

Origin: Charscan library, by Wil Baden, 2003-02-17, public domain.

See also: `search`, `compare`, `skip`, `scan`.

Source file: <src/lib/strings.MISC.fs>.

hz>bleep

```
hz>bleep ( frequency duration1 -- duration2 pitch ) "hertz-to-bleep;
```

Convert *frequency* (in Hz) and *duration1* (in ms) to the parameters *duration2 pitch* needed by `bleep`.

See also: `dhz>bleep`.

Source file: <src/lib/sound.48.fs>.

i

i

```
i ( -- n|u ) ( R: do-sys -- do-sys )
```

Return a copy *n*|*u* of the current (innermost) `loop` index.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `i'`, `j`, `k`.

Source file: <src/kernel.z80s>.

i

```
i ( "ccc<eol>" -- )
```

A command of `gforth-editor`: `insert` *ccc* or, if it’s empty, the contents of the `ibuf` insert buffer.

See also: `il`, `h`, `l`, `r`.

Source file: <src/lib/prog.editor.gforth.fs>.

i

```
i ( n -- )
```

A command of `specforth-editor`: Insert text from `pad` at line *n*, moving the old line *n* down. Line 15 is lost.

See also: `b`, `c`, `d`, `e`, `f`, `h`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `x`.

Source file: <src/lib/prog.editor.specforth.fs>.

i'

```
i' ( -- n|u ) ( R: loop-sys -- loop-sys ) "i-tick"
```

Return a copy *n*|*u* of the limit of the current (innermost) `loop` index.

Origin: Comus.

See also: `i`, `j'`, `k'`.

Source file: <src/lib/flow.j.fs>.

ibuf

```
ibuf ( -- ca )
```

Return the address *ca* of the 100-byte insert buffer used by the `gforth-editor`.

See also: `rbuf`, `fbuf`, `i`, `il`, `insert`.

Source file: <src/lib/prog.editor.gforth.fs>.

if

```
if
  Compilation: ( C: -- orig )
  Run-time:    ( f -- )
```

Compilation: Compile a conditional `0branch` and put the location *orig* of its unresolved destination on the control-flow stack, to be resolved by `else` or `then`.

Run-time: If *f* is zero, continue execution at the location specified by the resolution of *orig*.

`if` is an `immediate` and `compile-only` word.

Definition:

```
: if \ Compilation: ( C: -- orig )
     \ Run-time:    ( f -- )
  compile 0branch >mark ; immediate compile-only
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `again`, `until`, `ahead`, `0if`, `-if`, `+if`, `andif`, `orif`.

Source file: <src/kernel.z80s>.

if>

```
if> "if-from"
  Compilation: ( -- )
               ( C: -- orig )
```

Part of the `{if` control structure.

Source file: <src/lib/flow.dijkstra.fs>.

ifcase

```
ifcase
  Compilation: ( -- orig )
  Run-time:    ( x f -- x| )
```

Part of a `thiscase` structure that checks *x*.

Compilation: Leave the forward reference *orig*, to be consumed by `exitcase`.

Runtime: If *f* is true, discard *x* and continue execution; else skip the code compiled until the next `exitcase`.

`ifcase` is an `immediate` and `compile-only` word.

See also: `othercase`.

Source file: <src/lib/flow.thiscase.fs>.

ifelse

```
ifelse ( x1 x2 f -- x1 | x2 ) "if-else"
```

If *f* is true return *x1*, otherwise return *x2*.

Source file: <src/lib/math.operators.1-cell.fs>.

if}

```
if} "if-curly-bracket"
  Compilation: ( count -- )
               ( C: orig#...orig#n -- )
```

Terminate a `{if` control structure.

Source file: <src/lib/flow.dijkstra.fs>.

il

```
il ( -- )
```

A command of `gforth-editor`: `insert` the line stored into `pad` at the cursor position.

See also: `i`, `l`.

Source file: <src/lib/prog.editor.gforth.fs>.

im1,

```
im1, ( -- ) "i-m-one-comma"
```

Compile the Z80 `assembler` instruction `IM 1`.

See also: `im2,`, `di,`, `ei,`, `halt,`.

Source file: <src/lib/assembler.fs>.

im2,

```
im2, ( -- ) "i-m-two-comma"
```

Compile the Z80 `assembler` instruction `IM 2`.

See also: `im1,`, `di,`, `ei,`, `halt,`.

Source file: <src/lib/assembler.fs>.

immediate

```
immediate ( -- )
```

Make the most recent definition an immediate word.

Definition:

```
: immediate ( -- ) immediate-mask latest lex! ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `immediate-mask`, `latest`, `lex!`, `immediate?`, `compile-only`.

Source file: <src/kernel.z80s>.

immediate-mask

```
immediate-mask ( -- b )
```

A `cconstant`. *b* is the bitmask of the precedence bit, set by `immediate`.

See also: `compile-only-mask`, `smudge-mask`, `word-length-mask`.

Source file: <src/kernel.z80s>.

immediate?

```
immediate? ( nt -- f ) "immediate-question"
```

f is true if the word *nt* is immediate.

Definition:

```
: immediate? ( nt -- f ) immediate-mask lex? ;
```

See also: `immediate`, `lex?`, `immediate-mask`.

Source file: <src/kernel.z80s>.

in,

```
in, ( b -- ) "in-comma"
```

Compile the Z80 `assembler` instruction `IN A,(b)`.

See also: `out,`, `inbc,`.

Source file: <src/lib/assembler.fs>.

inbc,

```
inbc, ( reg -- ) "in-b-c-comma"
```

Compile the Z80 `assembler` instruction `IN reg,©`.

See also: `outbc,` `in,`.

Source file: <src/lib/assembler.fs>.

inc,

```
inc, ( reg -- ) "inc-comma"
```

Compile the Z80 `assembler` instruction `INC reg`.

See also: `dec,`, `incp,`.

Source file: <src/lib/assembler.fs>.

incp,

```
incp, ( regp -- ) "inc-p-comma"
```

Compile the Z80 `assembler` instruction `INC regp`.

See also: `decp,`, `inc,`.

Source file: <src/lib/assembler.fs>.

incx

```
incx ( -- a ) "inc-x"
```

A `2variable` used by `adraw176` and `aline176`.

See also: `incy`, `x1`, `y1`.

Source file: <src/lib/graphics.lines.fs>.

incx,

```
incx, ( disp regpi --  ) "inc-x-comma"
```

Compile the Z80 `assembler` instruction `INC (regp+disp)`.

See also: `decx,`, `addx,`, `adcx,`.

Source file: <src/lib/assembler.fs>.

incy

```
incy ( -- a ) "ink-y"
```

A `2variable` used by `adraw176` and `aline176`.

See also: `incx`, `x1`, `y1`.

Source file: <src/lib/graphics.lines.fs>.

indented+

```
indented+ ( u -- ) "indented-plus"
```

Add *u* to `#indented`.

Source file: <src/lib/display.ltype.fs>.

index

```
index ( u1 u2 -- )
```

Display the first line of each block over the range from *u1* to *u2*, which conventionally contains a comment with a title.

Origin: fig-Forth, Forth-79 (Reference Word Set), Forth-83 (Uncontrolled Reference Words).

Source file: <src/lib/tool.list.blocks.fs>.

index-block

```
index-block ( block -- )
```

Index block *block*.

See also: `use-fly-index`.

Source file: <src/lib/blocks.indexer.fly.fs>.

index-ilike

```
index-ilike ( u1 u2 "name" -- )
```

Display the first line of each block over the range from *u1* to *u2*, which conventionally contains a comment with a title, as long as the string *name* is included in the line. The string comparison is case-insensitive.

See also: `index`, `index-like`.

Source file: <src/lib/tool.list.blocks.fs>.

index-like

```
index-like ( u1 u2 "name" -- )
```

Display the first line of each block over the range from *u1* to *u2*, which conventionally contains a comment with a title, as long as the string *name* is included in the line. The string comparison is case-sensitive.

See also: `index`, `index-ilike`.

Source file: <src/lib/tool.list.blocks.fs>.

index-name

```
index-name ( ca len -- )
```

Add word *ca len* to the blocks index, if not done before.

The current word list must be `index-wordlist`.

WARNING	The block where *ca len* was found is stored as the execution token of its definition in the index. This way the index uses no data space. Don't put `index-wordlist` in the search order unless you know what you're doing.

Source file: <src/lib/blocks.indexer.COMMON.fs>.

index-wordlist

```
index-wordlist ( -- wid )
```

Word list for the indexed words.

Source file: <src/lib/blocks.indexer.COMMON.fs>.

indexed-block?

```
indexed-block? ( block -- f ) "indexed-block-question"
```

Is block *block* indexed?

See also: `use-fly-index`.

Source file: <src/lib/blocks.indexer.fly.fs>.

indexed-blocks

```
indexed-blocks ( -- ca )
```

Bit array to mark the indexed blocks

See also: `use-fly-index`.

Source file: <src/lib/blocks.indexer.fly.fs>.

init-2val

```
init-2val  ( -- ) "init-two-val"
```

Init the default behaviour of words created by `2val`: Make them return their content.

`init-2val` is a factor of `2val`.

Source file: <src/lib/data.val.fs>.

init-arg-action

```
init-arg-action  ( -- )
```

Set `arg-action` to `arg-default-action`.

`init-arg-action` is a factor of `arguments`.

Source file: <src/lib/locals.arguments.fs>.

init-asm

```
init-asm ( -- )
```

A deferred word (see `defer`) called by `asm`. Its action is set by the assembler labels module in order to init the labels. Its default action is `noop`.

Source file: <src/kernel.z80s>.

init-cval

```
init-cval  ( -- ) "init-c-val"
```

Init the default behaviour of words created by `cval`: Make them return their content.

`init-cval` is a factor of `cval`.

Source file: <src/lib/data.val.fs>.

init-labels

```
init-labels ( -- )
```

Init the `assembler` labels and their references, by allocating space for them in the `stringer` and erasing it. `labels` and `l-refs` are given new values.

Loading `init-labels` makes it the action of `init-asm`, which is called by `asm` and therefore also by `code` and `;code`. Therefore, if the program needs a specific ammount of labels or label references, `max-labels` and `max-l-refs` must be configured before compiling the assembly word.

Source file: <src/lib/assembler.labels.fs>.

init-ufia

```
init-ufia ( -- )
```

Init the contents of `ufia`, erasing them with zeroes, then setting `device` to "d", `dstr1` to "*" (i.e., the default drive) and `sstr1` to 2.

See also: `-ufia`.

Source file: <src/lib/dos.gplusdos.fs>.

init-val

```
init-val  ( -- )
```

Init the default behaviour of words created by `val`: Make them return their content.

`init-val` is a factor of `val`.

Source file: <src/lib/data.val.fs>.

ink-mask

```
ink-mask ( -- b )
```

A `cconstant`. *b* is the bitmask of the bits used to indicate the ink in an attribute byte.

See also: `unink-mask`, `set-ink`, `attr!`, `paper-mask`, `bright-mask`, `flash-mask`.

Source file: <src/lib/display.attributes.fs>.

ink.

```
ink. ( b -- ) "ink-dot"
```

Set ink color to *b* (0..9), by printing the corresponding control characters. If *b* is greater than 9, 9 is used instead.

`ink.` is much slower than `set-ink` or `attr!`, but it can handle pseudo-colors 8 (transparent) and 9 (contrast), setting the corresponding system variables accordingly.

See also: `paper.`, `(0-9-color.`.

Source file: <src/lib/display.attributes.fs>.

inkey

```
inkey ( -- 0 | c )
```

Leave the value of the key being pressed. If no key being pressed, leave 0.

`inkey` works only when an interrupts routine reads the keyboard and updates the related system variables.

See also: `get-inkey`, `key`.

Source file: <src/lib/keyboard.inkey.fs>.

input-buffer

```
input-buffer ( -- a )
```

A `2variable`. *a* is the address of a double cell containing the address and length of the current input buffer.

See also: `source`, `set-source`.

Source file: <src/kernel.z80s>.

input-stream-header

```
input-stream-header ( "name" -- )
```

Create a dictionary header *name*.

Definition:

```
: input-stream-header ( "name" -- )
  parse-name dup 0= #-16 ?throw header, ;
```

See also: `header`, `header,`, `nextname-header`, `parse-name`.

Source file: <src/kernel.z80s>.

insert

```
insert ( ca1 len1 ca2 len2 -- )
```

Insert string *ca1 len1* at the start of string *ca2 len2*.

See also: `delete`, `replace`.

Source file: <src/lib/strings.MISC.fs>.

internal

```
internal ( -- nt )
```

Start internal (private) definitions. Return the *nt* of the latest word created in the compilation word list.

The end of the internal definitions is marked by `end-internal`. Then those definitions can be unlinked by `unlink-internal` or hidden by `hide-internal`.

See also: `isolate`, `module`, `package`, `privatize`, `seclusion`.

Source file: <src/lib/modules.internal.fs>.

interpret

```
interpret ( -- )
```

The text interpreter which sequentially executes or compiles text from the current input `stream source` (terminal or disk) depending on `state`. If the word name cannot be found in the search `order` it is converted to a number by `number?`, according to the current `base`. That also failing, an `error` will happen.

The actions of the text interpreter are determined by the configuration of `interpret-table`.

See also: `evaluate`, `execute-parsing`, `set-source`, `nest-source`.

Source file: <src/kernel.z80s>.

interpret-table

```
interpret-table ( -- a )
```

a is the zero-offset address of the execution table used by `interpret`. The table contains 13 vectors. The behaviour of the Forth text interpreter can be changed by replacing these vectors. The structure and contents of the execution table is the following:

Table 26. Structure of `interpret-table`.

Cell offset	Execution token or zero	Condition
-6	`execute`	Compile an immediate and compile-only word
-5	`compile,`	Compile a compile-only word
-4	`execute`	Compile an immediate word
-3	`compile,`	Compile an ordinary word
-2	`2literal`	Compile a 2-cell number
-1	`xliteral`	Compile a 1-cell number
0	`not-understood`	Not a word nor a number (error)
1	0	Interpret a 1-cell number (do nothing)
2	0	Interpret a 2-cell number (do nothing)
3	`execute`	Interpret an ordinary word
4	`execute`	Interpret an immediate word
5	`compilation-only`	Interpret a compile-only word (error)
6	`compilation-only`	Interpret an immediate and compile-only word (error)

Source file: <src/kernel.z80s>.

inverse

```
inverse ( f -- )
```

If *f* is zero, turn the inverse printing mode off; else turn it on.

See also: `inverse-off`, `inverse-on`, `overprint`.

Source file: <src/lib/display.attributes.fs>.

inverse-cond

```
inverse-cond ( op1 -- op2 )
```

Convert a Z80 `assembler` condition flag *op1* (actually a jump opcode) to its opposite *op2*.

Examples: The opcode returned by `c?` is converted to the opcode returned by `nc?`, `nz?` to `z?`, `po?` to `pe?`, `p?` to `m?; and vice versa.

`inverse-cond` is used by `rif`, `runtil`, `aif` and `auntil`.

Source file: <src/lib/assembler.fs>.

inverse-off

```
inverse-off ( -- )
```

Turn the inverse printing mode off.

See also: `inverse-on`, `inverse`, `overprint-off`.

Source file: <src/lib/display.attributes.fs>.

inverse-on

```
inverse-on ( -- )
```

Turn the inverse printing mode on.

See also: `inverse-off`, `inverse`, `overprint-on`.

Source file: <src/lib/display.attributes.fs>.

inversely

```
inversely ( b1 -- b2 )
```

Convert attribute *b1* to its inversely equivalent *b2*, i.e. *b2* has paper and ink exchanged.

See also: `contrast`, `papery`, `brighty`, `flashy`, `attr>paper`, `attr>ink`.

Source file: <src/lib/display.attributes.fs>.

invert

```
invert ( x1 -- x2 )
```

Invert all bits of *x1* giving its logical inverse *x2*.

See also: `0=`, `negate`.

Source file: <src/kernel.z80s>.

invert-display

```
invert-display ( -- )
```

Invert the pixels of the whole screen.

See also: `wave-display`, `fade-display`.

Source file: <src/lib/graphics.display.fs>.

is

```
is
  Interpretation: ( xt "name" -- )
  Compilation:    ( "name" -- )
  Run-time:       ( xt -- )
```

Interpretation: (xt "name" —)

Set *name*, which was defined by `defer`, to execute *xt*.

Compilation: ("name" —)

Append the run-time semantics given below to the current definition.

Run-time: (xt —)

Set *name*, which was defined by `defer`, to execute *xt*.

WARNING `is` is a state-smart word (see `state`).

Origin: Forth-2012 (CORE EXT).

See also: `[is]`, `<is>`.

Source file: <src/lib/define.deferred.fs>.

isolate

```
isolate ( -- )
```

Create a word list, push it on the search order and set it as the compilation word list.

`isolate` is the simplest way to create a module. Usage example:

```
get-current isolate
  \ Inner words.
set-current
  \ Interface words.
previous
```

See also: `internal`, `module`, `package`, `privatize`, `seclusion`.

Source file: <src/lib/modules.MISC.fs>.

item

```
item ( ca len wid -- i*x )
```

If *ca len* is an item of the `associative-list` *wid*, return its value *i*x*; else `throw` an exception #-13 ("undefined word").

See also: `item?`. `entry:`, `centry:`, `2entry:`, `sentry:`, `items`.

Source file: <src/lib/data.associative-list.fs>.

item?

```
item? ( ca len wid -- false | xt true ) "item-question"
```

Is *ca len* an item of the `associative-list` *wid*? If so return its *xt* and `true`, else return `false`.

See also: `item`. `entry:`, `centry:`, `2entry:`, `sentry:`, `items`.

Source file: <src/lib/data.associative-list.fs>.

items

```
items ( wid -- )
```

List items of the `associative-list` *wid*.

Source file: <src/lib/data.associative-list.fs>.

ix

```
ix ( -- regpi ) "i-x"
```

regpi is the identifier of the Z80 `assembler` register "IX".

See also: `a`, `b`, `c`, `d`, `e`, `h`, `l`, `m`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

iy

```
iy ( -- regpi ) "i-y"
```

regpi is the identifier of the Z80 `assembler` register "IY".

See also: `a`, `b`, `c`, `d`, `e`, `h`, `l`, `m`, `ix`, `sp`.

Source file: <src/lib/assembler.fs>.

j

j

```
j ( -- n|u ) ( R: loop-sys1 loop-sys2 -- loop-sys1 loop-sys2 )
```

Return a copy *n*|*u* of the next-outer `loop` index.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `j'`, `i`, `k`.

Source file: <src/lib/flow.j.fs>.

j'

```
j' ( -- n|u ) ( R: loop-sys1 loop-sys2 -- loop-sys1 loop-sys2 ) "j-tick"
```

Return a copy *n*|*u* of the limit of the next-outer `loop` index.

Origin: Comus.

See also: `j`, `i'`, `k'`.

Source file: <src/lib/flow.j.fs>.

jiffy!

```
jiffy! ( a -- ) "jiffy-store"
```

Set the address *a* of the so called "jiffy call", a Z80 routine to be called by G+DOS after the OS interrupts routine, every 50th of a second.

See also: `jiffy@`, `-jiffy`.

Source file: <src/lib/multitask.gplusdos.fs>.

jiffy@

```
jiffy@ ( -- a ) "jiffy-fetch"
```

Get the address *a* of the so called "jiffy call", a Z80 routine that is called by G+DOS after the OS interrupts routine, every 50th of a second.

See also: `jiffy!`, `-jiffy`.

Source file: <src/lib/multitask.gplusdos.fs>.

join

```
join ( b1 b2 -- x )
```

b1 is the low-order byte of *x*, and *b2* is the high-order byte of *x*.

Origin: IsForth.

See also: `split`, `flip`.

Source file: <src/lib/math.operators.1-cell.fs>.

jp,

```
jp, ( a -- ) "j-p-comma"
```

Compile the Z80 opcode to jump to *a*.

Definition:

```
: jp, ( a -- ) $C3 c, , ;
```

See also: `call,`.

Source file: <src/kernel.z80s>.

jp>jr

```
jp>jr ( op1 -- op2 ) "j-p-greater-than-j-r"
```

Convert a Z80 `assembler` absolute-jump instruction *op1* to its relative-jump equivalent *op2*. Throw error #-273 if the jump condition is invalid.

`jp>jr` is a factor of `?jr,`, `rif` and `runtil`.

Source file: <src/lib/assembler.fs>.

jphl,

```
jphl, ( -- ) "j-p-h-l-comma"
```

Compile the Z80 `assembler` instruction `JP (HL)`.

See also: `jpix,`.

Source file: <src/lib/assembler.fs>.

jpix,

```
jpix, ( -- ) "j-p-i-x-comma"
```

Compile the Z80 `assembler` instruction `JP (IX)`.

See also: `jphl,`.

Source file: <src/lib/assembler.fs>.

jpnext,

```
jpnext, ( -- ) "j-p-next-comma"
```

Compile a Z80 jump to `next`.

See also: `jp,`.

Source file: <src/kernel.z80s>.

jr,

```
jr, ( a -- ) "j-r-comma"
```

Compile the Z80 `assembler` instruction `JR n`, being *n* an offset from the current address to address *a*.

See also: `?jr,`, `djnz,`, `jp,`.

Source file: <src/lib/assembler.fs>.

k

k

```
k ( -- n|u ) ( R: loop-sys1 ... loop-sys3 -- loop-sys1 ... loop-sys3 )
```

Return a copy *n|u* of the second outer `loop` index.

Origin: Forth-83 (Controlled reference words).

See also: `k'`, `i`, `j`.

Source file: <src/lib/flow.j.fs>.

k'

```
k' ( -- n|u ) ( R: loop-sys1 ... loop-sys3 -- loop-sys1 ...  loop-sys3 ) "k-tick"
```

Return a copy *n|u* of the limit of the second outer `loop` index.

Origin: Comus.

See also: `k`, `i'`, `j'`.

Source file: <src/lib/flow.j.fs>.

key

```
key ( -- c )
```

Return character *c* of the key struck, a member of the defined character set. Keyboard events that do not correspond to such characters are discarded until a valid character is received, and those events are subsequently unavailable.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `key?`, `new-key`, `new-key-`, `-keys`.

Source file: <src/kernel.z80s>.

key-caps-lock

```
key-caps-lock ( -- c )
```

c is the caps-lock control character, which is obtained by pressing "Caps Shift + 2" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-right`, `key-down`, `key-up`, `key-delete`, `key-enter`, `key-graphics`, `key-true-video`, `key-inverse-video`.

Source file: <src/lib/keyboard.MISC.fs>.

key-delete

```
key-delete ( -- c )
```

c is the delete control character, which is obtained by pressing "Caps Shift + 0" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-right`, `key-down`, `key-up`, `key-enter`, `key-graphics`, `key-true-video`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-down

```
key-down ( -- c )
```

c is the cursor-down control character, which is obtained by pressing "Caps Shift + 6" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-right`, `key-up`, `key-delete`, `key-enter`, `key-graphics`, `key-true-video`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-edit

```
key-edit ( -- c )
```

c is the edit control character, which is obtained by pressing "Caps Shift + 1" and can be read by `key`.

See also: `key-left`, `key-right`, `key-down`, `key-up`, `key-delete`, `key-enter`, `key-graphics`, `key-true-video`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-enter

```
key-enter ( -- c )
```

c is the enter control character, which is obtained by pressing "Enter" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-right`, `key-down`, `key-up`, `key-delete`, `key-graphics`, `key-true-video`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-graphics

```
key-graphics ( -- c )
```

c is the graphics control character, which is obtained by pressing "Caps Shift + 9" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-right`, `key-down`, `key-up`, `key-delete`, `key-enter`, `key-true-video`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-inverse-video

```
key-inverse-video ( -- c )
```

c is the inverse-video control character, which is obtained by pressing "Caps Shift + 4" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-right`, `key-down`, `key-up`, `key-delete`, `key-enter`, `key-graphics`, `key-`

`true-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-left

```
key-left ( -- c )
```

c is the cursor-left control character, which is obtained by pressing "Caps Shift + 5" and can be read by `key`.

See also: `key-edit`, `key-right`, `key-down`, `key-up`, `key-delete`, `key-enter`, `key-graphics`, `key-true-video`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-right

```
key-right ( -- c )
```

c is the cursor-right control character, which is obtained by pressing "Caps Shift + 8" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-down`, `key-up`, `key-delete`, `key-enter`, `key-graphics`, `key-true-video`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-translation-table

```
key-translation-table ( -- a )
```

A `variable`. *a* is the address of a cell containing the address of the current key translation table, used by `key`.

The table consists of pairs of characters. The first one is the character that has to be translated and the second one is its translation. The table is finished with a zero.

The default table makes it possible to access the following characters with Symbol Shift: '[', ']', '~', '|', '\', '{' and '}'.

Source file: <src/kernel.z80s>.

key-true-video

```
key-true-video ( -- c )
```

c is the true-video control character, which is obtained by pressing "Caps Shift + 3" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-right`, `key-down`, `key-up`, `key-delete`, `key-enter`, `key-graphics`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key-up

```
key-up ( -- c )
```

c is the cursor-up control character, which is obtained by pressing "Caps Shift + 7" and can be read by `key`.

See also: `key-edit`, `key-left`, `key-right`, `key-down`, `key-delete`, `key-enter`, `key-graphics`, `key-true-video`, `key-inverse-video`, `key-caps-lock`.

Source file: <src/lib/keyboard.MISC.fs>.

key?

```
key? ( -- f ) "key-question"
```

If a character is available, return true. Otherwise, return false. If non-character keyboard events are available before the first valid character, they are discarded and are subsequently unavailable. The character is returned by the next execution of `key`.

After `key?` returns with a value of true, subsequent executions of `key?` prior to the execution of `key` also return true, without discarding keyboard events.

Origin: Forth-94 (FACILITY), Forth-2012 (FACILITY).

See also: `-keys`.

Source file: <src/kernel.z80s>.

kk#>kk

```
kk#>kk ( n -- b a ) "k-k-dash-to-k-k"
```

Convert keyboard key number *n* to its data: key bitmask *b* and keyboard row port *a*.

See also: `kk-ports`, `/kk`, `kk@`.

Source file: <src/lib/keyboard.MISC.fs>.

kk,

```
kk, ( b a -- ) "k-k-comma"
```

Compile key definition *b a* (bitmask and port) into table `kk-ports`. The actual definition of `kk,` depends on the value of `/kk`.

See also: `kk@`, `/kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-0

```
kk-0 ( -- b a ) "k-k-0"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "0" with `pressed?`.

See also: `kk-0#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-0#

```
kk-0# ( -- n ) "k-k-0-dash"
```

Return index *n* of the physical key "0" in tables `kk-chars` and `kk-ports`.

See also: `kk-0`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-1

```
kk-1 ( -- b a ) "k-k-1"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "1" with `pressed?`.

See also: `kk-1#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-1#

```
kk-1# ( -- n ) "k-k-1-dash"
```

Return index *n* of the physical key "1" in tables `kk-chars` and `kk-ports`.

See also: `kk-1`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-2

```
kk-2 ( -- b a ) "k-k-2"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "2" with `pressed?`.

See also: `kk-2#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-2#

```
kk-2# ( -- n ) "k-k-2-dash"
```

Return index *n* of the physical key "2" in tables `kk-chars` and `kk-ports`.

See also: `kk-2`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-3

```
kk-3 ( -- b a ) "k-k-3"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "3" with `pressed?`.

See also: `kk-3#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-3#

```
kk-3# ( -- n ) "k-k-3-dash"
```

Return index *n* of the physical key "3" in tables `kk-chars` and `kk-ports`.

See also: `kk-3`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-4

```
kk-4 ( -- b a ) "k-k-4"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "4" with `pressed?`.

See also: `kk-4#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-4#

```
kk-4# ( -- n ) "k-k-4-dash"
```

Return index *n* of the physical key "4" in tables `kk-chars` and `kk-ports`.

See also: `kk-4`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-5

```
kk-5 ( -- b a ) "k-k-5"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "5" with `pressed?`.

See also: `kk-5#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-5#

```
kk-5# ( -- n ) "k-k-5-dash"
```

Return index *n* of the physical key "5" in tables `kk-chars` and `kk-ports`.

See also: `kk-5`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-6

```
kk-6 ( -- b a ) "k-k-6"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "6" with `pressed?`.

See also: `kk-6#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-6#

```
kk-6# ( -- n ) "k-k-6-dash"
```

Return index *n* of the physical key "6" in tables `kk-chars` and `kk-ports`.

See also: `kk-6`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-7

```
kk-7 ( -- b a ) "k-k-7"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "7" with `pressed?`.

See also: `kk-7#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-7#

```
kk-7# ( -- n ) "k-k-7-dash"
```

Return index *n* of the physical key "7" in tables `kk-chars` and `kk-ports`.

See also: `kk-7`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-8

```
kk-8 ( -- b a ) "k-k-8"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "8" with `pressed?`.

See also: `kk-8#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-8#

```
kk-8# ( -- n ) "k-k-8-dash"
```

Return index *n* of the physical key "8" in tables `kk-chars` and `kk-ports`.

See also: `kk-8`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-9

```
kk-9 ( -- b a ) "k-k-9"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "9" with `pressed?`.

See also: `kk-9#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-9#

```
kk-9# ( -- n ) "k-k-9-dash"
```

Return index *n* of the physical key "9" in tables `kk-chars` and `kk-ports`.

See also: `kk-9`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-a

```
kk-a ( -- b a ) "k-k-A"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "A" with `pressed?`.

See also: `kk-a#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-a#

```
kk-a# ( -- n ) "k-k-A-dash"
```

Return index *n* of the physical key "A" in tables `kk-chars` and `kk-ports`.

See also: `kk-a`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-b

```
kk-b ( -- b a ) "k-k-B"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "B" with `pressed?`.

See also: `kk-b#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-b#

```
kk-b# ( -- n ) "k-k-B-dash"
```

Return index *n* of the physical key "B" in tables `kk-chars` and `kk-ports`.

See also: `kk-b`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-c

```
kk-c ( -- b a ) "k-k-C"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "C" with

`pressed?`.

See also: `kk-c#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-c#

```
kk-c# ( -- n ) "k-k-C-dash"
```

Return index *n* of the physical key "C" in tables `kk-chars` and `kk-ports`.

See also: `kk-c`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-chars

```
kk-chars ( -- ca ) "k-k-chars"
```

ca is the address of a 40-byte table that contains the characters used as names of the physical keys (one character per key) and it's organized by keyboard rows, as follows:

Table 27. Keyboard matrix pointed by `kk-chars`.

1	2	3	4	5	
q	w	e	r	t	
a	s	d	f	g	
Caps Shift	z	x	c	v	
0	9	8	7	6	
p	o	i	u	y	
Enter	l	k	j	h	
Space	Symbol Shift	m	n	b	

The first 4 UDG codes displayed after the default configuration of `last-font-char` are used for the keys whose names are not a printable character, as follows:

Table 28. Items of `kk-chars` used as names of special keys.

Byte offset	**UDG code**	**Key**
15	128	Caps Shift
30	129	Enter
35	130	Space
36	131	Symbol Shift

The application should define those UDG with proper icons to represent the corresponding keys.

See also: `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-cs

```
kk-cs ( -- b a ) "k-k-caps-shift"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "Caps Shift" with `pressed?`.

See also: `kk-cs#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-cs#

```
kk-cs# ( -- n ) "k-k-caps-shift-dash"
```

Return index *n* of the physical key "Caps Shift" in tables `kk-chars` and `kk-ports`.

See also: `kk-cs`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-d

```
kk-d ( -- b a ) "k-k-D"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "D" with `pressed?`.

See also: `kk-d#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-d#

```
kk-d# ( -- n ) "k-k-D-dash"
```

Return index *n* of the physical key "D" in tables `kk-chars` and `kk-ports`.

See also: `kk-d`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-e

```
kk-e ( -- b a ) "k-k-E"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "E" with `pressed?`.

See also: `kk-e#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-e#

```
kk-e# ( -- n ) "k-k-E-dash"
```

Return index *n* of the physical key "E" in tables `kk-chars` and `kk-ports`.

See also: `kk-e`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-en

```
kk-en ( -- b a ) "k-k-enter"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "Enter" with `pressed?`.

See also: `kk-en#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-en#

```
kk-en# ( -- n ) "k-k-enter-dash"
```

Return index *n* of the physical key "Enter" in tables `kk-chars` and `kk-ports`.

See also: `kk-en`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-f

```
kk-f ( -- b a ) "k-k-F"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "F" with `pressed?`.

See also: `kk-f#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-f#

```
kk-f# ( -- n ) "k-k-F-dash"
```

Return index *n* of the physical key "F" in tables `kk-chars` and `kk-ports`.

See also: `kk-f`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-g

```
kk-g ( -- b a ) "k-k-G"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "G" with `pressed?`.

See also: `kk-g#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-g#

```
kk-g# ( -- n ) "k-k-G-dash"
```

Return index *n* of the physical key "G" in tables `kk-chars` and `kk-ports`.

See also: `kk-g`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-h

```
kk-h ( -- b a ) "k-k-H"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "H" with `pressed?`.

See also: `kk-h#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-h#

```
kk-h# ( -- n ) "k-k-H-dash"
```

Return index *n* of the physical key "H" in tables `kk-chars` and `kk-ports`.

See also: `kk-h`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-i

```
kk-i ( -- b a ) "k-k-I"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "I" with `pressed?`.

See also: `kk-i#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-i#

```
kk-i# ( -- n ) "k-k-I-dash"
```

Return index *n* of the physical key "I" in tables `kk-chars` and `kk-ports`.

See also: `kk-i`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-j

```
kk-j ( -- b a ) "k-k-J"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "J" with

`pressed?`.

See also: `kk-j#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-j#

```
kk-j# ( -- n ) "k-k-J-dash"
```

Return index *n* of the physical key "J" in tables `kk-chars` and `kk-ports`.

See also: `kk-j`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-k

```
kk-k ( -- b a ) "k-k-K"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "K" with `pressed?`.

See also: `kk-k#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-k#

```
kk-k# ( -- n ) "k-k-K-dash"
```

Return index *n* of the physical key "K" in tables `kk-chars` and `kk-ports`.

See also: `kk-k`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-l

```
kk-l ( -- b a ) "k-k-L"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "L" with `pressed?`.

See also: `kk-l#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-l#

```
kk-l# ( -- n ) "k-k-L-dash"
```

Return index *n* of the physical key "L" in tables `kk-chars` and `kk-ports`.

See also: `kk-l`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-m

```
kk-m ( -- b a ) "k-k-M"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "M" with `pressed?`.

See also: `kk-m#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-m#

```
kk-m# ( -- n ) "k-k-M-dash"
```

Return index *n* of the physical key "M" in tables `kk-chars` and `kk-ports`.

See also: `kk-m`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-n

```
kk-n ( -- b a ) "k-k-N"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "N" with `pressed?`.

See also: `kk-n#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-n#

```
kk-n# ( -- n ) "k-k-N-dash"
```

Return index *n* of the physical key "N" in tables `kk-chars` and `kk-ports`.

See also: `kk-n`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-o

```
kk-o ( -- b a ) "k-k-O"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "O" with `pressed?`.

See also: `kk-o#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-o#

```
kk-o# ( -- n ) "k-k-O-dash"
```

Return index *n* of the physical key "O" in tables `kk-chars` and `kk-ports`.

See also: `kk-o`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-p

```
kk-p ( -- b a ) "k-k-P"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "P" with `pressed?`.

See also: `kk-p#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-p#

```
kk-p# ( -- n ) "k-k-P-dash"
```

Return index *n* of the physical key "P" in tables `kk-chars` and `kk-ports`.

See also: `kk-p`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-ports

```
kk-ports ( -- a ) "k-k-ports"
```

A table that contains the key definitions (bitmak and port) of all keys.

The table contains 40 items, one per physical key, and it’s organized by keyboard rows.

Every item occupies 3 or 4 bytes, depending on the value of `/kk`. The default is 4.

See also: `kk,`, `kk@`, `#kk`, `kk-chars`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-q

```
kk-q ( -- b a ) "k-k-Q"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "Q" with `pressed?`.

See also: `kk-q#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-q#

```
kk-q# ( -- n ) "k-k-Q-dash"
```

Return index *n* of the physical key "Q" in tables `kk-chars` and `kk-ports`.

See also: `kk-q`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-r

```
kk-r ( -- b a ) "k-k-R"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "R" with `pressed?`.

See also: `kk-r#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-r#

```
kk-r# ( -- n ) "k-k-R-dash"
```

Return index *n* of the physical key "R" in tables `kk-chars` and `kk-ports`.

See also: `kk-r`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-s

```
kk-s ( -- b a ) "k-k-S"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "S" with `pressed?`.

See also: `kk-s#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-s#

```
kk-s# ( -- n ) "k-k-S-dash"
```

Return index *n* of the physical key "S" in tables `kk-chars` and `kk-ports`.

See also: `kk-s`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-sp

```
kk-sp ( -- b a ) "k-k-space"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "Space" with

`pressed?`.

See also: `kk-sp#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-sp#

```
kk-sp# ( -- n ) "k-k-space-dash"
```

Return index *n* of the physical key "Space" in tables `kk-chars` and `kk-ports`.

See also: `kk-sp`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-ss

```
kk-ss ( -- b a ) "k-k-symbol-shift"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "Symbol Shift" with `pressed?`.

See also: `kk-ss#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-ss#

```
kk-ss# ( -- n ) "k-k-symbol-shift-dash"
```

Return index *n* of the physical key "Symbol Shift" in tables `kk-chars` and `kk-ports`.

See also: `kk-ss`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-t

```
kk-t ( -- b a ) "k-k-T"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "T" with `pressed?`.

See also: `kk-t#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-t#

```
kk-t# ( -- n ) "k-k-T-dash"
```

Return index *n* of the physical key "T" in tables `kk-chars` and `kk-ports`.

See also: `kk-t`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-u

```
kk-u ( -- b a ) "k-k-U"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "U" with `pressed?`.

See also: `kk-u#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-u#

```
kk-u# ( -- n ) "k-k-U-dash"
```

Return index *n* of the physical key "U" in tables `kk-chars` and `kk-ports`.

See also: `kk-u`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-v

```
kk-v ( -- b a ) "k-k-V"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "V" with `pressed?`.

See also: `kk-v#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-v#

```
kk-v# ( -- n ) "k-k-V-dash"
```

Return index *n* of the physical key "V" in tables `kk-chars` and `kk-ports`.

See also: `kk-v`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-w

```
kk-w ( -- b a ) "k-k-W"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "W" with `pressed?`.

See also: `kk-w#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-w#

```
kk-w# ( -- n ) "k-k-W-dash"
```

Return index *n* of the physical key "W" in tables `kk-chars` and `kk-ports`.

See also: `kk-w`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-x

```
kk-x ( -- b a ) "k-k-X"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "X" with `pressed?`.

See also: `kk-x#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-x#

```
kk-x# ( -- n ) "k-k-X-dash"
```

Return index *n* of the physical key "X" in tables `kk-chars` and `kk-ports`.

See also: `kk-x`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-y

```
kk-y ( -- b a ) "k-k-Y"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "Y" with `pressed?`.

See also: `kk-y#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-y#

```
kk-y# ( -- n ) "k-k-Y-dash"
```

Return index *n* of the physical key "Y" in tables `kk-chars` and `kk-ports`.

See also: `kk-y`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-z

```
kk-z ( -- b a ) "k-k-Z"
```

Return key bitmask *b* and keyboard row port *a* needed for reading the physical key "Z" with `pressed?`.

See also: `kk-z#`, `#kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

kk-z#

```
kk-z# ( -- n ) "k-k-Z-dash"
```

Return index *n* of the physical key "Z" in tables `kk-chars` and `kk-ports`.

See also: `kk-z`, `#kk`, `kk#>kk`.

Source file: <src/lib/keyboard.MISC.fs>.

kk@

```
kk@ ( a1 -- b a2 ) "k-k-fetch"
```

Fetch a key definition *b a2* (bitmask and port) from item *a1* of table `kk-ports`. The actual definition of `kk@` depends on the value of `/kk`.

See also: `kk,`, `/kk`, `kk-ports`.

Source file: <src/lib/keyboard.MISC.fs>.

l

l

```
l ( -- reg )
```

Return the identifier *reg* of the Z80 `assembler` register "L".

See also: `a`, `b`, `c`, `d`, `e`, `h`, `m`, `ix`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

l

```
l ( -- )
```

A command of `gforth-editor`: Go to current screen.

Source file: <src/lib/prog.editor.gforth.fs>.

l

```
l ( -- )
```

A command of `specforth-editor`: List the current block.

See also: `b`, `c`, `d`, `e`, `f`, `h`, `i`, `m`, `n`, `p`, `r`, `s`, `t`, `x`, `scr`, `list`.

Source file: <src/lib/prog.editor.specforth.fs>.

l!

```
l! ( x n -- ) "l-store"
```

If `assembler` label *n* has been defined in the current definition, `throw` exception #-284 (assembly label number already used); else create a new `assembler` label *n* with value *x* and resolve all previous references to it that could have been created by `rl#` or `al#`. Usually *x* is an address.

See also: `l:`, `resolve-refs`.

Source file: <src/lib/assembler.labels.fs>.

l-refs

```
l-refs ( -- a )
```

A `variable`. *a* is the address of a cell containing the address of the label references table, which is allocated in the `stringer` by `init-labels`. The size of the table can be configured with `max-l-refs`.

Each element of the table (0 index) has the following structure:

```
+0 = byte: unused reference:
              all bits are 0
           used reference:
              label number:        bits 0..5
              relative reference?: bit 6 = 1 (mask ``rl&#45;id``)
              absolute reference?: bit 7 = 1 (mask ``al&#45;id``)
+1 = cell: label address
```

Source file: <src/lib/assembler.labels.fs>.

l/scr

```
l/scr ( -- b ) "l-slash-s-c-r"
```

A `cconstant`. *b* is the number of lines per `block` source: 16.

See also: `c/l`.

Source file: <src/kernel.z80s>.

l:

```
l: ( n -- ) "l-colon"
```

If `assembler` label *n* has been defined in the current definition, `throw` exception #-284 (assembly label number already used); else create a new `assembler` label *n* with the value returned by `here` and resolve all previous references to it that could have been created by `rl#` or `al#`.

See also: `l!`, `.l`, `labels`, `l-refs`, `init-labels`.

See also `unresolved` for an alternative method.

Source file: <src/lib/assembler.labels.fs>.

labels

```
labels ( -- a )
```

A `variable`. *a* is the address of a cell containing the address of the labels table, which is allocated in the `stringer` by `init-labels`. The size of the table can be configured with `max-labels`.

Each element of the table (0 index) is one cell, which contains either the address of the corresponding label or zero if the label is undefined.

See also: `/labels`, `l-refs`.

Source file: <src/lib/assembler.labels.fs>.

lang

```
lang ( -- b )
```

A `cconstant` containing the number *b* of the current language, used by translation tools `localized-word`, `localized-string` and `localized-character`.

Its default value is zero. The value must be changed by the application using `c!>`.

See also: `langs`.

Source file: <src/lib/translation.fs>.

langs

```
langs ( -- b )
```

A `cconstant` containing the number *b* of languages used by the application, needed by translation tools `localized-word`, `localized-string` and `localized-character`.

Its default value is zero. The value must be configured by the application using `c!>`, and it should not be changed later.

See also: `lang`.

Source file: <src/lib/translation.fs>.

laser-gun

```
laser-gun ( -- )
```

Laser gun sound for ZX Spectrum 48.

Source file: <src/lib/sound.48.fs>.

last

```
last ( -- a )
```

A `user` variable. *a* is the address of a cell containing the name token of the last word defined.

See also: `latest`, `lastxt`.

Source file: <src/kernel.z80s>.

last-column

```
last-column ( -- col )
```

Last column (x coordinate) in the current screen mode.

See also: `last-row`, `columns`, `column`.

Source file: <src/lib/display.cursor.fs>.

last-font-char

```
last-font-char ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the code of the last character displayed from the current font by the current action of `emit` and by `g-emit`. Higher characters are managed apart, displayed by `emit-udg` (depending on the actual implementation of `emit`, which is a deferred word; see `defer`) or `g-emit-udg`.

At the moment, only `mode-32-emit` and `g-emit` check this value. Eventually, also the alternative modes will use it.

`last-font-char` is a character variable, which must be set with `c!`. Its default value is 127.

See also: `set-font`, `set-udg`.

Source file: <src/kernel.z80s>.

last-locatable

```
last-locatable ( -- a )
```

A `variable`. *a* is the address of a cell containing the number of the last block to be searched by `located` and its descendants. Its default value is the last block of the disk.

See also: `first-locatable`.

Source file: <src/lib/002.need.fs>.

last-name

```
last-name ( ca1 len1 -- ca2 len2 )
```

Get the last name *ca2 len2* from string *ca1 len1*. A name is a substring separated by spaces.

See also: `first-name`, `/name`, `string/`, `-suffix`.

Source file: <src/lib/strings.MISC.fs>.

last-row

```
last-row ( -- row )
```

Last row (y coordinate) in the current screen mode.

See also: `last-column`, `row`, `rows`.

Source file: <src/lib/display.cursor.fs>.

last-stream

```
last-stream ( -- n )
```

n is the number of the last stream.

See also: `first-stream`, `os-strms`, `stream>`, `stream?`.

Source file: <src/lib/os.fs>.

last-tape-filename

```
last-tape-filename ( -- ca )
```

Address of the filename in `last-tape-header`.

See also: `/tape-filename`, `tape-filename`.

Source file: <src/lib/tape.fs>.

last-tape-filetype

```
last-tape-filetype ( -- ca )
```

Address of the file type (one byte) in `last-tape-header`.

See also: `tape-filetype`.

Source file: <src/lib/tape.fs>.

last-tape-header

```
last-tape-header ( -- ca )
```

Address of the second tape header, which is used by the ROM routines while loading. Its structure is the identical to `tape-header`.

It can be used by the application to know the details of the last tape file that was loaded.

See also: `last-tape-filename`, `last-tape-filetype`, `last-tape-start`, `last-tape-length`.

Source file: <src/lib/tape.fs>.

last-tape-length

```
last-tape-length ( -- a )
```

Address of the file length in `last-tape-header`.

See also: `tape-length`.

Source file: <src/lib/tape.fs>.

last-tape-start

```
last-tape-start ( -- a )
```

Address of the file start in `last-tape-header`.

See also: `tape-start`.

Source file: <src/lib/tape.fs>.

last-wordlist

```
last-wordlist ( -- a )
```

A `variable`. *a* is the address of a cell containing the data field address of the latest word list created.

See also: `wordlist`, `latest`.

Source file: <src/kernel.z80s>.

lastblk

```
lastblk ( -- a ) "last-b-l-k"
```

A `user` variable. *a* is the address of a cell containing the `block` number of the block most recently or loaded (e.g. with `load`, `continued` or `load-program`). `lastblk` is updated by `(load` and used by `reload`.

Source file: <src/kernel.z80s>.

lastxt

```
lastxt ( -- a ) "last-x-t"
```

A `user` variable. *a* is the address of a cell containing the execution token of the last word defined.

See also: `last`.

Source file: <src/kernel.z80s>.

latest

```
latest ( -- nt )
```

nt is the name token of the last word defined is the system.

Definition:

```
: latest ( -- nt ) last @ ;
```

Origin: Gforth.

See also: `last`, `current-latest`, `fyi`.

Source file: <src/kernel.z80s>.

latest-fid

```
latest-fid ( -- a )
```

A `variable`. *a* is the address of the latest file identifier created by `create-fid`, or zero if no one was created yet.

Source file: <src/lib/dos.gplusdos.fs>.

latest>wordlist

```
latest>wordlist ( wid -- ) "latest-to-wordlist"
```

Associate the latest name to the word list identified by *wid*.

See also: `wordlist`, `wordlist-name!`, `wordlist>vocabulary`, `wordlists`, `latest`.

Source file: <src/lib/word_lists.fs>.

latestxt

```
latestxt ( -- xt ) "latest-x-t"
```

Leave the execution token of the last word defined.

Origin: Gforth.

Source file: <src/kernel.z80s>.

lb

```
lb ( -- ) "l-b"
```

List bottom half of screen hold in `scr`.

See also: `lt`, `lm`, `list`, `list-lines`.

Source file: <src/lib/tool.list.blocks.fs>.

lcr

```
lcr ( -- ) "l-c-r"
```

If the cursor is neither at the home position nor at the start of a line, move it to the next row. `lcr` is part of the left-justified displaying system.

See also: `lcr?`, `(lcr`, `ltype`.

Source file: <src/lib/display.ltype.fs>.

lcr?

```
lcr? ( -- f ) "l-c-r-question"
```

Is the cursor neither at the home position nor at the start of a line? `lcr?` is part of the left-justified displaying system.

See also: `lcr`, `ltype`.

Source file: <src/lib/display.ltype.fs>.

ld#,

```
ld#, ( 8b reg -- ) "l-d-number-sign-comma"
```

Compile the Z80 `assembler` instruction `LD reg,8b`.

See also: `ld,`, `ldp#,`.

Source file: <src/lib/assembler.fs>.

ld,

```
ld, ( reg1 reg2 -- ) "l-d-comma"
```

Compile the Z80 `assembler` instruction `LD reg2,reg1`.

See also: `ld#,`, `ldp,`.

Source file: <src/lib/assembler.fs>.

ldai,

```
ldai, ( -- ) "l-d-a-i-comma"
```

Compile the Z80 `assembler` instruction `LD A,I`.

See also: `ldia,`, `ldar,`, `ld,`.

Source file: <src/lib/assembler.fs>.

ldar,

```
ldar, ( -- ) "l-d-a-r-comma"
```

Compile the Z80 `assembler` instruction `LD A,R`.

See also: `ldra,`, `ldai,`, `ld,`.

Source file: <src/lib/assembler.fs>.

ldd,

```
ldd, ( -- ) "l-d-d-comma"
```

Compile the Z80 `assembler` instruction `LDD`.

See also: `ldi,`, `lddr,`.

Source file: <src/lib/assembler.fs>.

lddr,

```
lddr, ( -- ) "l-d-d-r-comma"
```

Compile the Z80 `assembler` instruction `LDDR`.

See also: `ldir,`, `ldd,`.

Source file: <src/lib/assembler.fs>.

ldi,

```
ldi, ( -- ) "l-d-i-comma"
```

Compile the Z80 `assembler` instruction `LDI`.

See also: `ldd,`, `ldir,`.

Source file: <src/lib/assembler.fs>.

ldia,

```
ldia, ( -- ) "l-d-i-a-comma"
```

Compile the Z80 `assembler` instruction `LD I,A`.

See also: `ldai,`, `ldra,`, `ld,`.

Source file: <src/lib/assembler.fs>.

ldir,

```
ldir, ( -- ) "l-d-i-r-comma"
```

Compile the Z80 `assembler` instruction `LDIR`.

See also: `lddr,`, `ldi,`.

Source file: <src/lib/assembler.fs>.

ldp#,

```
ldp#, ( 16b regp -- ) "l-d-p-number-sign-comma"
```

Compile the Z80 `assembler` instruction `LD regp,16b`.

See also: `ldp,`, `ld#,`.

Source file: <src/lib/assembler.fs>.

ldp,

```
ldp, ( regp1 regp2 -- ) "l-d-p-comma"
```

Compile the Z80 `assembler` instructions required to load register pair *regp2* with register pair *regp1*.

Example: `b d ldp,` compiles the Z80 instructions `LD D,B` and `LD E,C`.

See also: `ld,`, `subp,`, `tstp,`, `clrp,`.

Source file: <src/lib/assembler.fs>.

ldra,

```
ldra, ( -- ) "l-d-r-a-comma"
```

Compile the Z80 `assembler` instruction `LD R,A`.

See also: `ldar,`, `ldir,`, `ld,`.

Source file: <src/lib/assembler.fs>.

ldsp,

```
ldsp, ( -- ) "l-d-s-p-comma"
```

Compile the Z80 `assembler` instruction `LD SP,HL`.

Source file: <src/lib/assembler.fs>.

leapy-year?

```
leapy-year? ( n -- f ) "leapy-year-question"
```

Is *n* a leapy year?

See also: `set-date`.

Source file: <src/lib/time.fs>.

leave

```
leave ( -- ) ( R: loop-sys -- )
```

Discard the `loop` control parameters for the current nesting level. Continue execution immediately following the innermost syntactically enclosing `loop` or `+loop`.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `?leave`, `0leave`, `unloop`, `do`, `?do`, `+loop`.

Source file: <src/kernel.z80s>.

lemit

```
lemit ( c -- ) "l-emit"
```

Display character *c* as part of the left-justified displaying system.

See also: `ltype`, `lspace`.

Source file: <src/lib/display.ltype.fs>.

lengths

```
lengths ( ca1 len1 ca2 len2 -- ca1 len1 ca2 len2 len1 len2)
```

Duplicate lengths *len1* and *len2* of strings *ca1 len1* and *ca2 len2*. `lengths` is a factor of `s+`.

`lengths` is written in Z80. Its equivalent definition in Forth is the following:

```
: lengths ( ca1 len1 ca2 len2 -- ca1 len1 ca2 len2 len1 len2 )
  2over nip over ;
```

Source file: <src/lib/strings.MISC.fs>.

less-of

```
less-of
  Compilation: ( C: -- of-sys )
  Run-time:    ( x1 x2 -- | x1 )
```

`less-of` is an `immediate` and `compile-only` word.

Usage example:

```
: test ( x -- )
  case
    10      of ." ten!"          endof
    15 less-of ." less than 15" endof
    ." greater than 14"
  endcase ;
```

See also: `case`, `greater-of`, `(less-of`.

Source file: <src/lib/flow.case.fs>.

lex!

```
lex! ( b nt -- ) "lex-store"
```

Set the bits of the mask *b* in the length byte of *nt*.

See also: `lex?`, `immediate`, `compile-only`.

Source file: <src/kernel.z80s>.

lex?

```
lex? ( nt b -- f ) "lex-question"
```

Test the bits at *nt* specified by the bitmask *b*. Return true if the result is non-zero, else return false.

See also: `lex!`, `immediate?`, `compile-only?`.

Source file: <src/kernel.z80s>.

lhome

```
lhome ( -- ) "l-home"
```

Move the cursor used by `ltype` and related words to its home position, at the top left (column 0, row 0).

Source file: <src/lib/display.ltype.fs>.

limit

```
limit ( -- a )
```

A `variable`. *a* is the address of a cell containing the address above the highest address usable by the data space (the data space is the region addressed by `dp`). Its default value is zero, which is right above the highest memory address ($FFFF).

`limit` can be modified by a program in order to reserve a memory zone for special purposes.

Origin: Fig-Forth's `limit` constant.

See also: `unused`, `farlimit`, `fyi`, `greeting`.

Source file: <src/kernel.z80s>.

limit-heap

```
limit-heap ( n -- a )
```

Create a `heap` of *n* bytes right above `limit` and return its address *a*. `limit` is moved down *n* bytes.

See also: `allot-heap`, `bank-heap`, `farlimit-heap`, `empty-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

line

```
line ( n -- a )
```

Part of `specforth-editor`: Leave address *a* of the beginning of line *n* in the current block buffer. The block number is in `scr`. Read the disk block from disk if it is not already in the disk buffer.

See also: `line>string`.

Source file: <src/lib/prog.editor.specforth.fs>.

line>string

```
line>string ( n1 n2 -- ca len ) "line-to-string"
```

Convert the line number *n1* and the screen number *n2* to a string *ca len* in the disk buffer containing the data.

Definition:

```
: line>string ( n1 n2 -- ca len )
  >r c/l b/buf */mod r> + block + c/l ;
```

Origin: fig-Forth's `(line`.

Source file: <src/kernel.z80s>.

lineblock>source

```
lineblock>source ( n u -- ) "line-block-to-source"
```

Set block *u* as the current source, starting from its line *n*.

See also: `block>source`.

Source file: <src/lib/blocks.fs>.

lineload

```
lineload ( n u -- ) "line-load"
```

Begin interpretation at line *n* of block *u*.

Origin: Forth-83 (Uncontrolled Reference Words).

See also: `load`.

Source file: <src/lib/blocks.fs>.

link,

```
link, ( head -- ) "link-comma"
```

Create a new node in data space for the linked list *head*:

Before:

- head → old_node

After:

- head → new_node
- new_node → old_node

See also: `link@`.

Source file: <src/lib/data.MISC.fs>.

link>name

```
link>name ( lfa -- nt ) "link-to-name"
```

Get *nt* from its *lfa*.

See also: `name>link`.

Source file: <src/lib/compilation.fs>.

link@

```
link@ ( node1 -- node2 ) "link-fetch"
```

Fetch the node *node2* from the linked list node *node1*. `link@` is an alias of `@`.

See also: `link,`.

Source file: <src/lib/data.MISC.fs>.

list

```
list ( u -- )
```

Display block *u* and store *u* in `scr`.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Controlled Reference Words), Forth-94 (BLOCK EXT), Forth-2012 (BLOCK EXT).

See also: `scr`, `list-lines`, `lt`, `lm`, `lb`.

Source file: <src/lib/tool.list.blocks.fs>.

list-line

```
list-line ( n u -- )
```

List line *n* from block *u*, without trailing spaces.

See also: `list-lines`, `.line#`, `.line`. `list`, `blk-line`.

Source file: <src/lib/tool.list.blocks.fs>.

list-lines

```
list-lines ( u n1 n2 -- )
```

List lines *n2*..*n3* of block *u* and store *u* in `scr`.

See also: `list`, `scr`, `list-line`.

Source file: <src/lib/tool.list.blocks.fs>.

lit

```
lit ( -- x )
```

Return *x*, which was compiled by `literal` after `lit`.

`lit` is a `compile-only` word.

See also: `clit`, `2lit`.

Source file: <src/kernel.z80s>.

literal

```
literal ( x -- )
```

Compile *x* in the current definition.

`literal` is an `immediate` and `compile-only` word.

Definition:

```
: literal ( x -- ) postpone lit , ; immediate compile-only
```

See also: `lit`, `cliteral`, `2literal`, `xliteral`, `]l`.

Source file: <src/kernel.z80s>.

lm

```
lm ( -- ) "l-m"
```

List middle part of screen hold in `scr`.

See also: `lt`, `lb`, `list`, `list-lines`.

Source file: <src/lib/tool.list.blocks.fs>.

load

```
load ( u -- )
```

Save the current input-source specification. Store *u* in `blk` (thus making block *u* the input source and setting the input buffer to encompass its contents) and `lastblk`, set `>in` to zero, and interpret. When the parse area is exhausted, restore the prior input source specification.

An error is issued if *u* is zero.

Definition:

```
: load ( u -- )
  dup 0= #-259 ?throw nest-source (load unnest-source ;
```

See also: `(load`, `nest-source`, `unnest-source`, `lineload`, `+load`, `thru`, `blk`.

Source file: <src/kernel.z80s>.

load-program

```
load-program ( "name" -- )
```

Load a program, i.e. a set of blocks that are loaded as a whole. The blocks of a program don't have block headers except the first one, which contains *name*. Therefore programs cannot have internal requisites, i.e. they use `need` only to load from the library, which must be before the blocks of the program on the disk or disks.

Programs don't need `-->` or any similar word to control the loading of blocks: The loading starts from the first block of the disk that has *name* in its header (surrounded by spaces), and continues until the last block of the disk or until `end-program` is executed.

See also: `loading-program`, `(load-program`.

Source file: <src/lib/blocks.fs>.

loader

```
loader ( u "name" -- )
```

Define a word *name* which, when executed, will load block *u*.

Origin: Forth-79's `loads` (Reference Word Set), Forth-83's `loads` (Appendix B. Uncontrolled Reference Words).

Source file: <src/lib/blocks.fs>.

loading-program

```
loading-program ( -- a )
```

a is the address of a cell containing a flag: Is a program being loaded by `load-program`? This flag is modified by `load-program` and `end-program`.

Source file: <src/lib/blocks.fs>.

loading?

```
loading? ( -- f ) "loading-question"
```

If a `block` is being loaded, i.e., if the content of `blk` is non-zero, return true; else return false.

See also: `?loading`, `load`.

Source file: <src/kernel.z80s>.

loads

```
loads ( u n -- )
```

Load *n* blocks starting from block *u*.

Origin: MMSFORTH.

Source file: <src/lib/blocks.fs>.

local

```
local ( a -- )
```

Save the value of variable *a*, which will be restored at the end of the current definition.

`local` is a `compile-only` word.

Usage example:

```
variable v
1 v !  v ? \ default value

: test ( -- )
  v local
  v ?  1887 v !  v ? ;

v ? \ default value
```

See also: `2local`, `clocal`, `arguments`, `anon`.

Source file: <src/lib/locals.local.fs>.

localized,

```
localized, ( x[langs]..x[1] -- )
```

Store a `langs` number of cells, from *x[1]* to *x[langs]* in the data space, updating `dp`.

`localized,` is a factor of `localized-word`, `localized-string`, `far-localized-string` and far>localized-string.

See also: `far-localized,`.

Source file: <src/lib/translation.fs>.

localized-character

```
localized-character ( c[langs]..c[1] "name" -- c )
```

Create a word *name* that will return a character from *c[langs]..c[1]*, depending on `lang`. *c[langs]..c[1]* are ordered by ISO language code, being TOS the first one.

See also: `localized-word`, `localized-string`, `langs`.

Source file: <src/lib/translation.fs>.

localized-string

```
localized-string ( ca[langs]..ca[1] "name" -- )
```

Create a word *name* that will return a counted string from *ca[langs]..ca[1]*, depending on `lang`. *ca[langs]..ca[1]*, are the addresses where the strings have been compiled. *ca[langs]..ca[1]*, are ordered by ISO language code, being TOS the first one.

See also: `far-localized-string`, `far>localized-string`, `localized-word`, `localized-character`, `langs`.

Source file: <src/lib/translation.fs>.

localized-word

```
localized-word ( xt[langs]..xt[1] "name" -- )
```

Create a word *name* that will execute an execution token from *xt[langs]..xt[1]*, depending on `lang`. *xt[langs]..xt[1]*, are the execution tokens of the localized versions. *xt[langs]..xt[1]*, are ordered by ISO language code, being TOS the first one.

See also: `localized-string`, `localized-character`, `langs`.

Source file: <src/lib/translation.fs>.

locate

```
locate ( "name" -- block | false )
```

Locate the first block whose header contains *name* (surrounded by spaces), and return its number *block*. If not found, return `false`. The search is case-sensitive.

Only the blocks delimited by `first-locatable` and `last-locatable` are searched.

See also: `located`.

Source file: <src/lib/002.need.fs>.

locate-need

```
locate-need ( "name" -- )
```

If *name* is not found in the current search order, locate the first block where *name* is included is the block header (surrounded by spaces), and load it. If not found, `throw` an exception #-268 ("needed, but not located").

`locate-need` is the default action of the deferred word `need` (see `defer`).

See also: `make-thru-index`.

Source file: <src/lib/002.need.fs>.

locate-needed

```
locate-needed ( ca len -- )
```

If the string *ca len* is not the name of a word found in the current search order, locate the first block where *ca len* is included in the block header (surrounded by spaces), and load it. If not found, `throw` an exception #-268 ("needed, but not located").

`locate-needed` is the default action of the deferred word `needed` (see `defer`).

See also: `make-thru-index`.

Source file: <src/lib/002.need.fs>.

locate-reneed

```
locate-reneed ( "name" -- )
```

Locate the first block whose header contains *name* (surrounded by spaces), and load it. If not found, `throw` an exception #-268 ("needed, but not located").

`locate-reneed` is the default action of the deferred word `reneed` (see `defer`).

See also: `make-thru-index`.

Source file: <src/lib/002.need.fs>.

locate-reneeded

```
locate-reneeded ( ca len -- )
```

Locate the first block whose header contains the string *ca len* (surrounded by spaces), and `load` it. If not found, `throw` an exception #-268 ("needed, but not located").

`locate-reneeded` is the default action of the deferred word `reneeded` (see `defer`).

See also: `make-thru-index`.

Source file: <src/lib/002.need.fs>.

located

```
located ( ca len -- block | 0 )
```

Locate the first block whose header contains the string *ca len* (surrounded by spaces), and return its number. If not found, return zero. The search is case-sensitive.

Only the blocks delimited by `first-locatable` and `last-locatable` are searched`.

`located` is a deferred word (see `defer`) whose default action is `(located`.

See also: `need-from`.

Source file: <src/lib/002.need.fs>.

loop

```
loop
  Compilation: ( do-sys -- )
```

Compile `(loop` and resolve the *do-sys* address left by `do`, or `?do`.

`loop` is an `immediate` and `compile-only` word.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `+loop`, `-do`.

Source file: <src/kernel.z80s>.

lower

```
lower ( c -- c' )
```

Convert *c* to lowercase *c'*.

See also: `lowers`, `upper`.

Source file: <src/kernel.z80s>.

lower_

```
lower_ ( -- a )
```

A `constant`. *a* is the address of a routine that converts to uppercase the ASCII character hold in the A register.

See also: `lower`, `upper_`.

Source file: <src/kernel.z80s>.

lowers

```
lowers ( ca len -- )
```

Convert string *ca len* to lowercase.

See also: `uppers`, `lower`.

Source file: <src/lib/strings.MISC.fs>.

lpage

```
lpage ( -- ) "l-page"
```

Clear the display and init the cursor used by `ltype` and related words.

Source file: <src/lib/display.ltype.fs>.

lshift

```
lshift ( x1 u -- x2 ) "l-shift"
```

Perform a logical left shift of *u* bit-places on *x1*, giving *x2*. Put zeroes into the least significant bits vacated by the shift.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `rshift`, `?shift`, `clshift`.

Source file: <src/lib/math.operators.1-cell.fs>.

lspace

```
lspace ( -- ) "l-space"
```

Display a space as part of the left-justified printing system.

See also: `lemit`, `ltype`.

Source file: <src/lib/display.ltype.fs>.

lt

```
lt ( -- ) "l-t"
```

List top half of screen hold in `scr`.

See also: `lm`, `lb`, `list`, `list-lines`.

Source file: <src/lib/tool.list.blocks.fs>.

ltype

```
ltype ( ca len -- ) "l-type"
```

Display character string *ca len* left-justified from the current cursor position.

See also: `lwidth`.

Source file: <src/lib/display.ltype.fs>.

ltype-indentation

```
ltype-indentation ( u -- ) "l-type-indentation"
```

Display an indentation of *u* spaces and update the corresponding variables of the `ltype` system.

Source file: <src/lib/display.ltype.fs>.

ltyped

```
ltyped ( n -- ) "l-typed"
```

Update `#ltyped` with *n* characters typed by `ltype`.

Source file: <src/lib/display.ltype.fs>.

lwidth

```
lwidth ( -- ca ) "l-width"
```

A byte variable containing the text width in columns used by `ltype` and related words. Its default value is `columns`, ie. the current width of the screen.

Source file: <src/lib/display.ltype.fs>.

m

m

```
m ( -- reg )
```

Return the identifier *reg* of Z80 `assembler` pseudo-register "(HL)", i.e. the byte stored in the memory address pointed by register pair "HL".

See also: `a`, `b`, `c`, `d`, `e`, `h`, `l`, `ix`, `iy`, `sp`.

Source file: <src/lib/assembler.fs>.

m

```
m ( -- )
```

A command of `gforth-editor`: Mark current position.

Source file: <src/lib/prog.editor.gforth.fs>.

m

```
m ( n -- )
```

A command of `specforth-editor`: Move the cursor by *n* characters. The position of the cursor on its line is shown by a "_" (underline).

See also: `b`, `c`, `d`, `e`, `f`, `h`, `i`, `l`, `n`, `p`, `r`, `s`, `t`, `x`, `-move`.

Source file: <src/lib/prog.editor.specforth.fs>.

m*

```
m* ( n1 n2 -- d ) "m-star"
```

Multiply *n1* by *n2*, giving the result *d*.

Definition:

```
: m* ( n1 n2 -- d )
  2dup xor >r
  abs swap abs um*
  r> ?dnegate ;
```

Origin: fig-Forth, Forth-94 (CORE), Forth-2012 (CORE).

See also: `*`, `um*`, `d*`, `?dnegate`.

Source file: <src/kernel.z80s>.

m*/

```
m*/ ( d1 n1 +n2 -- d2 ) "m-star-slash"
```

Multiply *d1* by *n1* producing the triple-cell intermediate result *t*. Divide *t* by *+n2* giving the double-cell quotient *d2*.

Origin: Forth-94 (DOUBLE), Forth-2012 (DOUBLE).

See also: `*/`, `m*`.

Source file: <src/lib/math.operators.2-cell.fs>.

m+

```
m+ ( d1|ud1 n -- d2|ud2 ) "m-plus"
```

Add *n* to *d1*|*ud1*, giving the sum *d2*|*ud2*.

`m+` is written in Z80. An equivalent definition in Forth (1.48 slower, but 4 bytes smaller) is the following:

```
: m+ ( d1|ud1 n -- d2|ud2 ) s>d d+ ;
```

Origin: Forth-94 (DOUBLE) Forth-2012 (DOUBLE).

See also: `+`, `d+`.

Source file: <src/lib/math.operators.2-cell.fs>.

m/

```
m/ ( d n1 -- n2 n3 ) "m-slash"
```

A mixed magnitude math operator which leaves the signed remainder *n2* and signed quotient *n3* from a double number dividend *d* and divisor *n1*.

`m/` is a deferred word (see `defer`) whose default action is `sm/rem`, so it does a symmetric division (the remainder takes its sign from the dividend), as in fig-Forth and Forth-79. It can be set to execute `fm/mod` instead.

`m/` is executed by all other division operators. Therefore setting it to execute either `sm/rem` or `fm/mod` will change the behaviour of all division operators.

Rationale:

The Forth-79 Standard specifies that the signed division operators (`/`, `/mod`, `mod`, `*/mod`, and `*/`) round non-integer quotients towards zero (symmetric division). Forth-83 changed the semantics of these operators to round towards negative infinity (floored division). To resolve this issue, Forth-94 and Forth-2012 permit to supply either floored or symmetric operators, and include a floored division primitive (`fm/mod`), and a symmetric division primitive (`sm/rem`).

Origin: fig-Forth.

Source file: <src/kernel.z80s>.

m?

```
m? ( -- op ) "m-question"
```

Return the opcode *op* of the Z80 `assembler` instruction `jp m`, to be used as condition and consumed by `?ret,`, `?jp,`, `?call,`, `aif`, `awhile` or `auntil`.

See also: `z?`, `nz?`, `c?`, `nc?`, `po?`, `pe?`, `p?`.

Source file: <src/lib/assembler.fs>.

macro

```
macro ( name -- )
```

Start the definition of an `assembler` macro *name*.

Usage example:

```
macro dos-in, ( -- ) DB c, #231 c, endm
  \ Assemble the Z80 instruction `in a,(#231)`, to page in
  \ the Plus D memory.
```

See also: `endm`, `asm`, `code`.

Source file: <src/lib/assembler.macro.fs>.

magenta

```
magenta ( -- b )
```

A `cconstant` that returns 3, the value that represents the magenta color.

See also: `black`, `blue`, `red`, `green`, `cyan`, `yellow`, `white`, `contrast`, `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

make

```
make
  Interpretation: ( "name" -- )
  Compilation:    ( -- )
```

Interpretation: Parse *name*, which is the name of a word created by `doer`, and make it execute the colon definition that follows.

Usage example:

```
doer flashes
flashes \ does nothing
make flashes 8 0 ?do i border loop ;
flashes \ works
```

Compilation: Modify the next inline compiled word of the current definition, which was created by `doer`, and make it execute the rest of the definition after it.

Usage example:

```
doer flashes
flashes \ does nothing
: activate ( -- ) make flashes 8 0 ?do i border loop ;
activate
flashes \ works
```

`make` is an `immediate` word.

See also: `;and`, `undo`.

Source file: <src/lib/flow.doer.fs>.

make-block-chars

```
make-block-chars ( a -- )
```

Make the bit patterns of the 16 ZX Spectrum block characters, originally assigned to character codes 128..143, and store them (128 bytes in total) from address *a*.

`make-block-chars` is provided for easier conversion of BASIC programs that use the original block characters. These characters are part of the ZX Spectrum character set, but they are not included in the ROM font. Instead, their bitmaps are built on the fly by the BASIC ROM printing routine. In Solo Forth there's no such restriction, and characters 0..255 can be redefined by the user.

`make-block-chars` is written in Z80 and uses 18 B of code space, but the word `block-chars` is provided as an alternative.

Source file: <src/lib/graphics.udg.fs>.

make-thru-index

```
make-thru-index ( -- )
```

Create the blocks index and activate it. The current word list and the current search order are preserved.

`make-thru-index` first creates a blocks index, i.e. a word list from the names that are on the index (header) line of every searchable block, ignoring duplicates; second, it executes `use-thru-index` to activate the blocks index, changing the default behaivour of `need` and related words.

The words in the index have a fake execution token, which is the block they belong to. This way, after indexing all the disk blocks only once, `need` will search the word list and load the block of the word found. On the contrary, the default action of `need` is to search all the blocks every time.

The default action of `need` and related words can be restored with `use-no-index`.

Source file: <src/lib/blocks.indexer.thru.fs>.

manual-see

```
manual-see ( -- a )
```

A `variable`. *a* is the address of a cell containing a flag. When the flag is non-zero, the decompilation of colon words done by `see` can be controlled manually with some keys, which are displayed at the start of the process.

See also: `see-usage`.

Source file: <src/lib/tool.see.fs>.

marker

```
marker ( "name" -- )
```

Create a definition *name*. When *name* is executed, it will restore all dictionary allocation and search order pointers to the state they had just prior to the definition of "name". Remove the definition of *name* and all subsequent definitions.

The following data are preserved and restored: the data-space pointer (`here`), the name-space pointer (`np@`), the word lists pointer (`last-wordlist`), the compilation word list (`get-current`), the search order (`order`) and the word lists (`dump-wordlists`).

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `marker,`, `unmarker`, `anew`.

Source file: <src/lib/tool.marker.fs>.

marker,

```
marker, ( -- ) "marker-comma"
```

Save the current state of the system before creating the corresponding `marker` that will restore it with `unmarker`. The data that describes the state of the system is stored at the current data-space pointer (`here`), while the data-space pointer itself is stored in the body of the new `marker`. The saving process is the following:

Store at the current data-space pointer the names pointer (`np@`), the latest definition pointers (`latest` and `latestxt`), the word lists pointer (`last-wordlist`), the current compilation word list (`get-current`), the search order (`order,`) and the word lists (`wordlists,`) at the current data-space pointer.

`marker,` is a factor of `marker`.

Source file: <src/lib/tool.marker.fs>.

mask+attr!

```
mask+attr! ( b1 b2 -- ) "mask-plus-attribute-store"
```

Set *b1* as the current attribute mask and *b2* as the current attribute.

See also: `mask+attr@`, `attr!`, `attr-mask!`

Source file: <src/lib/display.attributes.fs>.

mask+attr-setter

```
mask+attr-setter ( b1 b2 "name" -- ) "mask-plus-attribute-setter"
```

Create a definition *name* that, when executed, will set *b1* as the current attribute mask and *b2* as the current attribute.

See also: `attr-setter`.

Source file: <src/lib/display.attributes.fs>.

mask+attr>perm

```
mask+attr>perm ( -- ) "mask-plus-attribute-to-perm"
```

Make the current attribute and mask permanent.

NOTE	Words that use attributes don't use the OS permanent attribute but the temporary one, which is called "current attribute" in Solo Forth.

Source file: <src/lib/display.attributes.fs>.

mask+attr@

```
mask+attr@ ( -- b1 b2 ) "mask-plus-attribute-fetch"
```

Set *b* as the current attribute mask.

See also: `attr-mask!`, `perm-attr-mask@`.

Source file: <src/lib/display.attributes.fs>.

match

```
match ( ca1 len1 ca2 len2 -- true n3 | false n4 )
```

Part of `specforth-editor`: Match the string *ca2 len2* with all strings contained in the string *ca1 len1*. If found leave *n3* bytes until the end of the matching string, else leave *n4* bytes to end of line.

See also: `-text`.

Source file: <src/lib/prog.editor.specforth.fs>.

max

```
max ( n1 n2 -- n3 )
```

n3 is the greater of *n1* and *n2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `min`, `umax`, `dmax`, `0max`, `>`.

Source file: <src/kernel.z80s>.

max-blocks

```
max-blocks ( -- u )
```

Return the number *u* of maximum blocks available in the system. The number depends on `#block-drives` and `blocks/disk`.

Source file: <src/kernel.z80s>.

max-char

```
max-char  ( -- u )
```

u is the maximum value of any character in the character set.

See also: `address-unit-bits`, `/counted-string`, `environment?`.

Source file: <src/lib/environment-question.fs>.

max-d

```
max-d ( -- d )
```

d is the largest usable signed double.

See also: `max-n`, `max-ud`, `environment?`.

Source file: <src/lib/environment-question.fs>.

max-disk-capacity

```
max-disk-capacity ( -- n )
```

A `constant`. *n* is maximum number of KiB available for files on a disk, i.e. the actual capacity excluding the tracks used for the catalogue.

See also: `tracks/cat`, `tracks/disk`, `b/sector`, `sectors-used`, `drive-unused`.

Source file: <src/lib/dos.gplusdos.fs>.

max-drives

```
max-drives ( -- b )
```

A `cconstant`. *b* is the maximum number of drives available in the DOS.

See also: `first-drive`, `drive`.

Source file: <src/kernel.z80s>.

max-esc-order

```
max-esc-order ( -- n )
```

A `constant` that returns the maximum number of word lists in the escaped strings search order.

Its default value is 4, but the application can define this constant with any other value before loading the words that need it, and it will be kept.

See also: `esc-context`, `#esc-order`, `set-esc-order`, `get-esc-order`, `>order`.

Source file: <src/lib/strings.escaped.fs>.

max-l-refs

```
max-l-refs ( -- ca )
```

ca is the address of a byte containing the maximum number (count) of unresolved `assembler` label references that can be created by `rl#` or `al#`. Its default value is 16. The program can change the value, but the default one should be restored after the code word has been compiled.

`max-l-refs` is used by `init-labels` to allocate the `l-refs` table.

Usage example:

```
need l:
assembler-wordlist >order     max-l-refs c@
                          #20 max-l-refs c! previous

code my-word ( -- )
  \ Z80 code that needs #20 label references
end-code

assembler-wordlist >order max-l-refs c! previous
```

See also: `max-labels`.

Source file: <src/lib/assembler.labels.fs>.

max-labels

```
max-labels ( -- ca )
```

ca is the address of a byte containing the maximum number (count) of `assembler` `labels` that can be defined by `l:`. Its default value is 8, i.e. labels 0..7 can be used. The program can change the value, but the default one should be restored after the code word has been compiled.

`max-labels` is used by `init-labels` to allocate the `labels` table.

Usage example:

```
need assembler need l:
assembler-wordlist >order    max-labels c@
                         #24 max-labels c! previous

code my-word ( -- )
  \ Z80 code that needs #24 labels
end-code

assembler-wordlist >order max-labels c! previous
```

See also: `max-l-refs`.

Source file: <src/lib/assembler.labels.fs>.

max-n

```
max-n ( -- n )
```

n is the largest usable signed integer.

See also: `max-u`, `max-d`, `environment?`.

Source file: <src/lib/environment-question.fs>.

max-order

```
max-order ( -- n )
```

A `constant`. *n* is the maximum number of word lists in the search order.

See also: `context`, `#order`, `set-order`, `get-order`, `>order`.

Source file: <src/kernel.z80s>.

max-u

```
max-u ( -- u )
```

u is the largest usable unsigned integer.

See also: `max-n`, `max-ud`, `environment?`.

Source file: <src/lib/environment-question.fs>.

max-ud

```
max-ud ( -- ud ) "max-u-d"
```

ud is the largest usable unsigned double.

See also: `max-u`, `max-d`, `environment?`.

Source file: <src/lib/environment-question.fs>.

max>top

```
max>top ( n1 n2 -- n1 n2 | n2 n1 )
```

Make sure the top of stack is the greater of *n1* and *n2*.

See also: `min>top`, `pair=`.

Source file: <src/lib/math.operators.1-cell.fs>.

menu

```
menu  ( -- )
```

Activate the current menu, which has been set by `set-menu` and displayed by `.menu`.

See also: `new-menu`, `menu-key-up`, `menu-key-down`, `menu-key-choose`, `options-table`, `actions-table`.

Source file: <src/lib/menu.sinclair.fs>.

menu-banner-attr

```
menu-banner-attr ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the attribute of the current `menu` banner. Its default value is white ink on black paper, with bright.

See also: `menu-body-attr`, `menu-highlight-attr`, `.menu-banner`, `black`, `papery`, `white`, `brighty`.

Source file: <src/lib/menu.sinclair.fs>.

menu-body-attr

```
menu-body-attr ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the attribute of the current `menu` background. Its default value is black ink on white paper, with bright.

See also: `menu-banner-attr`, `menu-highlight-attr`, `.menu-options`, `white`, `papery`, `brighty`.

Source file: <src/lib/menu.sinclair.fs>.

menu-highlight-attr

```
menu-highlight-attr ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the attribute used to highlight the current `menu` option. Its default value is the combination of `cyan`, `papery` and `brighty`, i.e. black ink on cyan bright paper.

See also: `menu-banner-attr`.

Source file: <src/lib/menu.sinclair.fs>.

menu-key-choose

```
menu-key-choose ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the key code used to move the current `menu` selection down. Its default value is 13, i.e. the enter key.

See also: `menu-key-up`, `menu-key-down`.

Source file: <src/lib/menu.sinclair.fs>.

menu-key-down

```
menu-key-down ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the key code used to move the current `menu` selection down. Its default value is character '6'.

See also: `menu-key-up`, `menu-key-choose`.

Source file: <src/lib/menu.sinclair.fs>.

menu-key-up

```
menu-key-up ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the key code used to move the current `menu` selection up. Its default value is character '7'.

See also: `menu-key-down`, `menu-key-choose`.

Source file: <src/lib/menu.sinclair.fs>.

menu-options

```
menu-options ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the current `menu` number of options. `menu-options` is set by `set-menu`.

See also: `menu-width`.

Source file: <src/lib/menu.sinclair.fs>.

menu-rounding

```
menu-rounding ( -- a )
```

A `variable`. *a* is the address of a cell containing a flag. When the flag is non-zero, the top and the bottom `menu` options are connected in a circular manner, i.e. pressing `menu-key-up` at the top option leads to to the botton option, and pressing `menu-key-down` at the bottom option lead to the top.

See also: `menu-key-choose`, `menu-highlight-attr`.

Source file: <src/lib/menu.sinclair.fs>.

menu-title

```
menu-title ( -- a )
```

A `2variable`. *a* is the address of a double cell containing the address and length of a string which is the title of the current `menu`. `menu-title` is set by `set-menu`.

See also: `menu-width`, `menu-xy`, `menu-banner-attr`, `.menu-banner`.

Source file: <src/lib/menu.sinclair.fs>.

menu-width

```
menu-width ( -- ca )
```

A `cvariable`. *ca* is the address of a byte containing the width of the current `menu` in characters. `menu-width` is set by `set-menu`.

See also: `menu-title`, `menu-body-attr`, `menu-banner-attr`, `menu-highlight-attr`.

Source file: <src/lib/menu.sinclair.fs>.

menu-xy

```
menu-xy ( -- a ) "menu-x-y"
```

A `2variable`. *a* is the address of a double cell containing the coordinates (column and row) of the current `menu`. `menu-xy` is set by `set-menu`.

See also: `menu-width`, `.menu-border`, `.menu-banner`.

Source file: <src/lib/menu.sinclair.fs>.

message-warn

```
message-warn ( ca len -- ca len ) "warn-dot-message"
```

If the contents of the user variable `warnings` is not zero and the word name *ca len* is already defined in the current compilation word list, display a warning message.

`message-warn` is an alternative action of the deferred word `warn` (see `defer`).

See also: `warnings`, `error-code-warn`, `error-warn`, `?warn`.

Source file: <src/lib/compilation.fs>.

method

```
method ( m v "name" -- m' `)
```

Define a selector.

Source file: <src/lib/objects.mini-oof.fs>.

middle-octave

```
middle-octave ( -- a )
```

Return the address of a 12-cell table that contains the frequencies in dHz (tenths of Hz) of the middle octave. They are used by `beep>dhz` to calculate the frequency of any note.

Here is a diagram to show the offsets of all the notes in the table, on the piano (extracted from the manual of the ZX Spectrum +3 transcripted by Russell et al.):

```
| | C#| D#| | | F#| G#| A#| |
| | Db| Eb| | | Gb| Ab| Bb| |
| | 1 | 3 | | | 6 | 8 |10 | |
| |___|___| | |___|___|___| |
|   |   |   |   |   |   |   |
| 0 | 2 | 4 | 5 | 7 | 9 |11 |
|___|___|___|___|___|___|___|
  C   D   E   F   G   A   B
```

See also: `beep`, `/octave`, `octave-changer`.

Source file: <src/lib/sound.48.fs>.

min

```
min ( n1 n2 -- n3 )
```

n3 is the lesser of *n1* and *n2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `max`, `umin`, `dmin`, `<`.

Source file: <src/kernel.z80s>.

min>top

```
min>top ( n1 n2 -- n1 n2 | n2 n1 )
```

Make sure the top of stack is the lesser of *n1* and *n2*.

See also: `max>top`, `pair=`.

Source file: <src/lib/math.operators.1-cell.fs>.

mini-64cpl-font

```
mini-64cpl-font ( -- a ) "mini-64-c-p-l-font"
```

a is the address of a 4x8-pixel font compiled in data space (336 bytes used), to be used in `mode-64ao` by setting `mode-64-font` first.

This font is included also in disk 0 as "mini.f64".

See also: `nbot-64cpl-font`, `omn1-64cpl-font`, `omn2-64cpl-font`, `owen-64cpl-font`.

Source file: <src/lib/display.mode.64.COMMON.fs>.

mod

```
mod ( n1 n2 -- n3 )
```

Divide *n1* by *n2*, giving the remainder *n3*.

Definition:

```
: mod ( n1 n2 -- n3 ) /mod drop ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `m/`, `/mod`, `/_mod`, `/-rem`, `gcd`.

Source file: <src/kernel.z80s>.

mode-32

```
mode-32 ( -- )
```

Set the default 32-CPL display mode. Usually this is not needed by the application, except when any other mode has been used, e.g. `mode-32iso`, `mode-42pw` or `mode-64ao`.

When any other mode is loaded, `mode-32` is automatically loaded and made the default display mode (therefore restored by `restore-mode`, which is called by `warm` and `cold`).

See also: `current-mode`, `set-font`, `set-mode-output`, `columns`, `rows`, `mode-32-emit`, `mode-32-xy`, `mode-32-at-xy`, `>form`.

Source file: <src/lib/display.mode.32.fs>.

mode-32-at-xy

```
mode-32-at-xy ( col row -- ) "mode-32-at-x-y"
```

Default action of `at-xy`, in `mode-32`.

WARNING	The system will crash if the coordinates are out of screen. For the sake of speed, no check is done. If needed, the program can use a wrapper word.

Source file: <src/kernel.z80s>.

mode-32-emit

```
mode-32-emit ( c -- )
```

Send character *c* to the current channel, calling the ROM routine at $0010.

`mode-32-emit` is the default action of `emit`.

`mode-32-emit` uses `last-font-char` the following way: characters up to and including `last-font-char` (by default, 0 .. 127) are displayed through the ROM routine, while higher characters (by default, 128 .. 255) are displayed by `emit-udg` from the current UDG set. The following table shows the effect of changing the value of `last-font-char`:

Table 29. Effect of `mode-32-emit` *depending on the value of* `last-font-char`.

Value	Effect
126	Characters 0 .. 126 are displayed by the ROM; characters 127 .. 255 are displayed by `emit-udg`.
127	Characters 0 .. 127 are displayed by the ROM; characters 128 .. 255 are displayed by `emit-udg`.
143	Characters 0 .. 143 are displayed by the ROM (this range includes the block graphics); characters 144 .. 255 are displayed by `emit-udg`.
162	Characters 0 .. 162 are displayed by the ROM (this range includes also the 128-BASIC UDG set 144 .. 162, corresponding to UDG characters 0 .. 18 in Solo Forth); characters 163 .. 255 are displayed by `emit-udg`.
255	Characters 0 .. 255 are displayed by the ROM (this range includes also the 128-BASIC tokens); no character is displayed by `emit-udg`.

When a standard character set is required, without the ROM interpreting characters 128 .. 255 its own way, `mode-32iso-emit` can be used instead.

See also: `current-mode`, `mode-32`, `set-font`, `set-udg`.

Source file: <src/kernel.z80s>.

mode-32-font

```
mode-32-font ( -- a )
```

A `variable`. *a* is the address of a cell containing the address of the font used by `mode-32`. Note the address of the font must be the address of its character 32 (space).

The default value of `mode-32-font` is `rom-font` plus 256 (the address of the space character in the ROM font).

Source file: <src/lib/display.mode.32.fs>.

mode-32-xy

```
mode-32-xy ( -- col row ) "mode-32-x-y"
```

Return the current cursor coordinates *col row* in `mode-32` and `mode-32iso`.

`mode-32-xy` is the action of `xy` when `mode-32` or `mode-32iso` are active, or by default when no alternative display mode has been used (e.g. `mode-64ao`).

Definition:

```
: mode_32-xy ( -- col row )
  24 23689 c@ -
  33 23688 c@ - dup 32 = if  drop 1+ 0  then  swap ;
  \ 23688 = OS variable S_POSX
  \ 23689 = OS variable S_POSY
```

Source file: <src/kernel.z80s>.

mode-32iso

```
mode-32iso ( -- )
```

Activate a 32-CPL display mode, an alternative to the default `mode-32`. The only difference with `mode-32` is `mode-32iso` can use a ISO character set, i.e. it displays characters 32..255 from the current font. See `mode-32iso-emit` for details.

`mode-32iso` is useful when a ISO character set is required (or any character set with more than 128 characters). A similar result could be obtained with `mode-32` and `last-font-char`, by treating the characters greater than 128 as UDG and using `set-udg`. The advantage of `mode-32iso` is the ISO font can be managed (e.g. built, loaded from disk, allocated, etc.) as a whole, using only the font address, and reserving the full UDG set for graphics.

See also: `current-mode`, `set-font`, `set-mode-output`, `columns`, `rows`, `mode-32-xy`, `mode-32-at-xy`, `>form`.

Source file: <src/lib/display.mode.32iso.fs>.

mode-32iso-emit

```
mode-32iso-emit ( c -- )
```

Display character *c* in `mode-32iso`, i.e. using the ROM routines but assuming the current font set by `set-font` contains printable characters 32..255. See the low-level factor `mode-32iso-output_` for details how this is achieved.

`mode-32iso-emit` is not affected by `last-font-char`.

`mode-32iso-emit` is a wrapper word which preserves the Forth IP and calls `mode-32iso-output_`.

Source file: <src/lib/display.mode.32iso.fs>.

mode-32iso-font

```
mode-32iso-font ( -- a )
```

A `variable`. *a* is the address of a cell containing the address of the font used by `mode-32iso`. Note the address of the font must be the address of its character 32 (space).

The default value of `mode-32iso-font` is `rom-font` plus 256 (the address of the space character in the ROM font).

Source file: <src/lib/display.mode.32iso.fs>.

mode-32iso-output_

```
mode-32iso-output_  ( -- a ) "mode-32-iso-output-underscore"
```

a is the address of a Z80 routine, the `mode-32iso` driver, which displays the character in the A register. calling the ROM routine at $09F4, but assuming the current font set by `set-font` contains printable characters 32..255.

In order to force the ROM routine interpret characters 128..255 as ordinary characters (not block graphics, user defined graphics or BASIC tokens, as `mode-32-emit` does), `mode-32iso-emit` modifies *c* if needed and moves the current font address accordingly before calling the ROM. As a result, the ROM routine treats character ranges 128..223 and 224..255 as 32..127 and 32..63 respectively.

`mode-32iso-output_` is called by `mode-32iso-emit`.

`mode-32iso-output_` is activated by `mode-32iso`, i.e. it’s set as the output routine of the current channel.

Source file: <src/lib/display.mode.32iso.fs>.

mode-42pw

```
mode-42pw ( -- ) "mode-42-p-w"
```

Start the 42-CPL display mode based on:

```
PRINT42.ASM
a routine from Your Sinclair #78 (Jun.1992) by P Wardle

Part of the VU-R Browser utility, written by  Jim Grimwood:

http://www.users.globalnet.co.uk/~jg27paw4/pourri/pourri.htm
```

The only control character recognized is #13 (carriage return).

See also: `current-mode`, `set-font`, `set-mode-output`, `columns`, `rows`, `mode-42pw-emit`, `mode-42pw-xy`, `mode-42pw-font`, `>form`, `mode-42pw-output_`, `mode-42rs`.

Source file: <src/lib/display.mode.42pw.fs>.

mode-42pw-emit

```
mode-42pw-emit ( c -- ) "mode-42-p-w-emit"
```

Display character *c* in `mode-42pw`, by calling `mode-64ao-output_`.

`mode-42pw-emit` is configured by `mode-42pw` as the action of `emit`.

Source file: <src/lib/display.mode.42pw.fs>.

mode-42pw-font

```
mode-42pw-font ( -- a ) "mode-42-p-w-font"
```

A `variable`. *a* is the address of a cell containing the address of the font used by `mode-42pw`. The font is a standard ZX Spectrum font (8x8-pixel characters, 32 characters per line), which is converted to 42 characters per line at real time. Note the address of the font must be the address of its character 32 (space).

The default value of `mode-42pw-font` is `rom-font` plus 256 (the address of the space character in the ROM font).

Source file: <src/lib/display.mode.42pw.fs>.

mode-42pw-output_

```
mode-42pw-output_ ( -- a )
```

a is the address of a Z80 routine that displays the character in register A in `mode-42pw`.

The only control character recognized is #13 (move cursor to next line, column 0).

Source file: <src/lib/display.mode.42pw.fs>.

mode-42pw-xy

```
mode-42pw-xy ( -- col row ) "mode-42-p-w-x-y"
```

Return the current cursor coordinates *col row* in `mode-42pw`. `mode-64ao-xy` is the action of `xy` when `mode-42pw` is active.

Source file: <src/lib/display.mode.42pw.fs>.

mode-42rs

```
mode-42rs ( -- ) "mode-42-r-s"
```

Start the 42-CPL display mode based on a routine written by Ricardo Serral Wigge, published on Microhobby, issue 66 (1986-02), page 24:

- http://microhobby.org/numero066.htm
- http://microhobby.speccy.cz/mhf/066/MH066_24.jpg

WARNING `mode-42rs` is under development. See the source code for details.

See also: `current-mode`, `mode-42pw`.

Source file: <src/lib/display.mode.42rs.fs>.

mode-64-font

```
mode-64-font ( -- a )
```

A `variable`. *a* is the address of a cell containing the address of the 4x8-pixel font used by `mode-64ao`. Note the address of the font must be the address of its character 32 (space). The size of a 4x8-pixel font is 336 bytes. The program is responsible for initializing the contents of this variable before executing `mode-64ao`.

NOTE If `mode-64-font` is changed when `mode-64ao` is on, for example to use a new font, `mode-64ao` must be executed again in order to make the change effective.

See also: `mini-64cpl-font`, `nbot-64cpl-font`, `omn1-64cpl-font`, `omn2-64cpl-font`, `owen-64cpl-font`.

Source file: <src/lib/display.mode.64.COMMON.fs>.

mode-64ao

```
mode-64ao ( -- ) "mode-64-a-o"
```

Start the 64-CPL display mode based on:

```
4x8 FONT DRIVER
(c) 2007, 2011 Andrew Owen
optimized by Crisis (to 602 bytes)
http://www.worldofspectrum.org/forums/discussion/14526/redirect/p1
```

The control characters recognized are 8 (left), 13 (carriage return) and 22 (at).

WARNING The "at" control character is followed by column and row, i.e. the order of the coordinates is inverted compared to the Sinclair BASIC convention and `mode-32`. This will be changed in a later version of the code.

See also: `current-mode`, `set-font`, `set-mode-output`, `columns`, `rows`, `mode-64ao-emit`, `mode-64ao-xy`, `mode-64-font`, `>form`, `mode-64ao-output_`, `mode-64es`.

Source file: <src/lib/display.mode.64ao.fs>.

mode-64ao-emit

```
mode-64ao-emit ( c -- ) "mode-64-a-o-emit"
```

Display character *c* in `mode-64ao`, by calling `(mode-64ao-output_`.

`mode-64ao-emit` is configured by `mode-64ao` as the action of `emit`.

Source file: <src/lib/display.mode.64ao.fs>.

mode-64ao-output_

```
mode-64ao-output_ ( -- a ) "mode-64-a-o-output-underscore"
```

a is the address of a Z80 routine, the entry to `mode-64ao` driver, which preserves the Forth IP and then displays the character in the A register by calling `(mode-64ao-output_`.

Source file: <src/lib/display.mode.64ao.fs>.

mode-64ao-xy

```
mode-64ao-xy ( -- col row ) "mode-64-a-o-x-y"
```

Return the current cursor coordinates *col row* in `mode-64ao`. `mode-64ao-xy` is the action of `xy` when `mode-64ao` is active.

Source file: <src/lib/display.mode.64ao.fs>.

mode-64es

```
mode-64es ( -- ) "mode-64-e-s"
```

Start the 64-CPL display mode based on:

```
4x8 FONT DRIVER
(c) 2007, 2011 Andrew Owen
optimized by Crisis (to 602 bytes)
http://www.worldofspectrum.org/forums/discussion/14526/redirect/p1

Version with integrated driver, adapted from 64#4, written
by Einar Saukas:
https://sites.google.com/site/zxgraph/home/einar-saukas/fonts
http://www.worldofspectrum.org/infoseekid.cgi?id=0027130
```

WARNING `mode-64es` is under development. See the source code for details.

See also: `current-mode`, `mode-64ao`.

Source file: <src/lib/display.mode.64es.fs>.

module

```
module ( "name" -- parent-wid )
```

Start the definition of a new module named *name*. `end-module` ends the module and `export` exports a word.

Usage example:

```
module greet

  : hello ( -- ) ." Hello" ;
  : mods ( -- ) ." Modules" ;

  : hi ( -- ) hello ." , " mods ." !" cr ;

export hi

end-module
```

Now only the exported definitions of the module are available.

```
hi       \ displays "Hello, Modules!"
hello    \ error, not found
```

The module name is defined as a constant that holds the word list identifier the module words are defined into. Therefore, to expose the internal words of a module, you can use `name >order`, where *name* is the name of the module.

See also: `internal`, `isolate`, `package`, `privatize`, `seclusion`.

Source file: <src/lib/modules.module.fs>.

more-words?

```
more-words? ( nt|0 -- nt|0 f ) "more-words-question"
```

A common factor of `words` and `words-like`.

Source file: <src/lib/tool.list.words.fs>.

move

```
move ( a1 a2 u -- )
```

If *u* is greater than zero, copy the contents of *u* consecutive bytes at *a1* to the *u* consecutive bytes at *a2*. After `move` completes, the *u* consecutive bytes at *a2* contain exactly what the *u* consecutive bytes at *a1* contained before the move.

See also: `cmove`, `cmove>`.

Origin: Forth-83 (Uncontrolled Reference Words), Forth-94 (STRING), Forth-2012 (STRING).

Source file: <src/kernel.z80s>.

move<far

```
move<far ( a1 a2 len -- ) "move-from-far"
```

If *len* is greater than zero, copy *len* consecutive cells from far-memory address *a1* to main-memory address *a2*.

Source file: <src/lib/memory.far.fs>.

move>far

```
move>far ( a1 a2 len -- ) "move-to-far"
```

If *len* is greater than zero, copy *len* consecutive cells from main-memory address *a1* to far-memory address *a2*.

Source file: <src/lib/memory.far.fs>.

ms

```
ms ( u -- )
```

Wait at least *u* ms (miliseconds).

Origin: Forth-94 (FACILITY EXT), Forth-202 (FACILITY EXT).

See also: `seconds`, `ticks-pause`.

Source file: <src/lib/time.fs>.

ms/tick

```
ms/tick ( -- n ) "ms-slash-tick"
```

Return the duration *n* of one clock tick in miliseconds.

See also: ticsk/second`, `ticks`.

Source file: <src/lib/time.fs>.

ms>ticks

```
ms>ticks ( n1 -- n2 ) "ms-to-ticks"
```

Convert *n1* milisecnods to the corresponding number *n2* of `ticks`.

See also: `ms/tick`.

Source file: <src/lib/time.fs>.

mt*

```
mt*   ( d n -- t ) "m-t-star"
```

t is the signed product of *d* times *n*.

Source file: <src/lib/math.operators.3-cell.fs>.

multiline-(located

```
multiline-(located ( ca len -- block | 0 ) "multiline-paren-located"
```

Locate the first block whose multiline header contains the string *ca len* (surrounded by spaces), and return its number. If not found, return zero. The search is case-sensitive.

Only the blocks delimited by `first-locatable` and `last-locatable` are searched.

`multiline-(located` is the default action of `(located`.

Source file: <src/lib/002.need.fs>.

n

n

```
n ( -- )
```

A command of `gforth-editor`: Go to next screen.

See also: `p`, `c`, `a`, `g`, `t`, `scr`, `top`.

Source file: <src/lib/prog.editor.gforth.fs>.

n

```
n ( -- )
```

A command of `specforth-editor`: Find the next occurrence of the string found by an `f` command.

See also: `b`, `c`, `d`, `e`, `f`, `h`, `i`, `l`, `m`, `p`, `r`, `s`, `t`, `x`, `find`.

Source file: <src/lib/prog.editor.specforth.fs>.

n!

```
n! ( x[u]..x[1] u a -- ) "n-store"
```

If *u* is not zero, store *u* cells at address *a*, being *x[1]* the first cell stored there and *x[u]* the last one.

See also: `nn!`, `!`, `n@`.

Source file: <src/lib/memory.MISC.fs>.

n,

```
n, ( x[u]..x[1] u -- ) "n-comma"
```

If *u* is not zero, store *u* cells *x[u]..x[1]* into data space, being *x[1]* the first one stored and *x[u]* the last one.

See also: `,`, `far-n,`, `nn,`, `n@`, `n!`.

Source file: <src/lib/memory.MISC.fs>.

n>r

```
n>r ( x#1..x#n n -- ) ( R: -- x#1..x#n n ) "n-to-r"
```

Remove *n+1* items from the data stack and store them for later retrieval by `nr>`. The return stack may be used to store the data. Until this data has been retrieved by `nr>`:

- this data will not be overwritten by a subsequent invocation of `n>r` and
- a program may not access data placed on the return stack before the invocation of `n>r`.

Origin: Forth-2012 (TOOLS EXT).

Source file: <src/lib/return_stack.fs>.

n>str

```
n>str ( n -- ca len ) "n-to-s-t-r"
```

Convert *n* to string *ca len*.

See also: `u>str`, `d>str`, `char>string`.

Source file: <src/lib/strings.MISC.fs>.

n@

```
n@ ( a u -- x[u]..x[1] ) "n-fetch"
```

If *u* is not zero, read *u* cells *x[u]..x[1]* from *a*, being *x[1]* the first one stored and *x[u]* the last one.

See also: `nn@`, `@`, `nn!`.

Source file: <src/lib/memory.MISC.fs>.

name-indexed?

```
name-indexed? ( ca len -- false | block true ) "name-indexed-question"
```

Search the index for word *ca len*. If found, return its *block* and `true`, else return `false`.

Source file: <src/lib/blocks.indexer.COMMON.fs>.

name<name

```
name<name ( nt1 -- nt2 ) "name-from-name"
```

Get the previous *nt2* from *nt1*, i.e. *nt2* is the word that was defined before *nt1*.

See also: `name>name`.

Source file: <src/lib/compilation.fs>.

name>

```
name> ( nt -- xt ) "name-to"
```

Definition:

```
: name> ( nt -- xt ) [ 2 cells ] literal - far@ ;
```

See also: `>name`, `name>body`, `name>str`, `name>string`, `name>immediate?`, `name>name`.

Source file: <src/kernel.z80s>.

name>>

```
name>> ( nt -- xtp ) "name-from-from"
```

Convert *nt* into its corresponding *xtp*.

See also: `>>name`, `name>`, `name>body`, `name>name`.

Source file: <src/lib/compilation.fs>.

name>body

```
name>body ( nt -- dfa ) "name-to-body"
```

Get *dfa* from its *nt*.

See also: `body>name`, `>body`, `name>`, `name>>`, `name>name`.

Source file: <src/lib/compilation.fs>.

name>compile

```
name>compile ( nt -- x xt ) "name-to-compile"
```

Compilation token *x xt* represents the compilation semantics of the word *nt*. The returned *xt* has the stack effect (i*x x — j*x). Executing *xt* consumes *x* and performs the compilation semantics of the word represented by *nt*.

Origin: Forth-2012 (TOOLS EXT).

See also: `name>interpret`, `comp'`, `(comp'`, `name>`.

Source file: <src/lib/compilation.fs>.

name>immediate?

```
name>immediate? ( nt -- xt f )
```

f is true if the word *nt* is immediate. *xt* is the corresponding execution token of *nt*.

Definition:

```
: name>immediate? ( nt -- xt f ) dup name> swap immediate? ;
```

See also: `immediate?`, `name>`, `name>body`, `name>str`, `name>string`.

Source file: <src/kernel.z80s>.

name>interpret

```
name>interpret ( nt -- xt | 0 ) "name-to-interpret"
```

Return *xt* that represents the interpretation semantics of the word *nt*. If *nt* has no interpretation semantics, return zero.

Origin: Forth-2012 (TOOLS EXT).

See also: `name>compile`, `'`, `compile-only?`, `name>`.

Source file: <src/lib/compilation.fs>.

name>link

```
name>link ( nt -- lfa ) "name-to-link"
```

Convert *nt* into its corresponding *lfa*.

See also: `link>name`, `name>`, `name>body`, `name>>`, `name>name`.

Source file: <src/lib/compilation.fs>.

name>name

```
name>name ( nt1 -- nt2 ) "name-to-name"
```

Get the next *nt2* from *nt1*, i.e. *nt2* is the word that was defined after *nt1*.

WARNING

`name>name` is not absolutely reliable, because *nt2* is calculated after the name length of *nt1*. If something was compiled in name space or the name-space pointer `np` was altered between the definition identified by *nt1* and the following definition, the result *nt2* will be wrong.

See also: `name<name`, `name>`, `name>body`, `name>>`.

Source file: <src/lib/compilation.fs>.

name>str

```
name>str ( nt -- ca len ) "name-to-s-t-r"
```

Convert the name token *nt* to its name string *ca len* in far memory.

See also: `name>string`, `name>immediate?`, `name>`, `name>body`.

Source file: <src/lib/compilation.fs>.

name>string

```
name>string ( nt -- ca len ) "name-to-string"
```

Convert the name token *nt* to its name string *ca len* in the `stringer`.

See also: `name>str`, `name>immediate?`, `name>`, `name>body`.

Source file: <src/lib/compilation.fs>.

nbot-64cpl-font

```
nbot-64cpl-font ( -- a ) "n-bot-64-c-p-l-font"
```

a is the address of a 4x8-pixel font compiled in data space (336 bytes used), to be used in `mode-64ao` by setting `mode-64-font` first.

This font is included also in disk 0 as "nbot.f64".

See also: `mini-64cpl-font`, `omn1-64cpl-font`, `omn2-64cpl-font`, `owen-64cpl-font`.

Source file: <src/lib/display.mode.64.COMMON.fs>.

nc?

```
nc? ( -- op ) "n-c-question"
```

Return the opcode *op* of the Z80 `assembler` instruction `jp nc`, to be used as condition and consumed by `?ret,`, `?jp,`, `?call,`, `?jr,`, `aif`, `rif`, `awhile`, `rwhile`, `auntil` or `runtil`.

See also: `z?`, `nz?`, `c?`, `po?`, `pe?`, `p?`, `m?`.

Source file: <src/lib/assembler.fs>.

ndrop

```
ndrop ( x1...xn n -- ) "n-drop"
```

Drop *n* cell items from the stack.

See also: `2ndrop`, `drop`, `2drop`.

Source file: <src/lib/data_stack.fs>.

need

```
need ( "name" -- )
```

If *name* is not found in the current search order, locate the first block where *name* is included is the block header (surrounded by spaces), and load it. If not found, `throw` an exception #-268 ("needed, but not located").

`need` is a deferred word (see `defer`) whose default action is `locate-need`.

See also: `make-thru-index`.

Source file: <src/lib/002.need.fs>.

need-from

```
need-from ( "name" -- )
```

Locate the first block whose header contains *name* (surrounded by spaces), and set it the first one `located` will search from. If not found, `throw` an exception #-268 ("needed, but not located").

`need-from` is intended to prevent undesired name clashes during the execution of `need` and related words. *name* is supposed to be a conventional marker.

Usage example:

```
( x )

: x ( -- ) ." Wrong x!" ;

( use-x )

need-from ==data-structures== need x

x

( y ==data-structures== )

: y ." Y data structure; ;

( x )

: x ." X data structure; ;
```

Source file: <src/lib/002.need.fs>.

need-here

```
need-here ( "name" -- )
```

If *name* is not a word found in the current search order, load the current block.

`need-here` is a faster alternative to `need`, when the needed word is in the same block, and conditional compilation is used with `?\`, `?(` or `[if]`.

Source file: <src/lib/002.need.fs>.

needed

```
needed ( ca len -- )
```

If the string *ca len* is not the name of a word found in the current search order, load the first block where *ca len* is included in the block header (surrounded by spaces). If not found, `throw` an exception #-268 ("needed, but not located").

`needed` is a deferred word (see `defer`) whose default action is `locate-needed`.

See also: `make-thru-index`.

Source file: <src/lib/002.need.fs>.

needed-word

```
needed-word ( -- a )
```

A `2variable`. *a* is the address of a double-cell containing the address and length of the string containing the word currently needed by `need` and friends.

Source file: <src/lib/002.need.fs>.

needing

```
needing ( "name" -- f )
```

Parse *name*. If there's no unresolved `need`, `needed`, `reneed` or `reneeded`, return `true`. Otherwise, if *name* is the needed word specified by the last execution of `need` or `needed`, return `true`, else return `false`.

See also: `unneeding`.

Source file: <src/lib/002.need.fs>.

neg,

```
neg, ( -- ) "neg-comma"
```

Compile the Z80 `assembler` instruction `NEG`.

See also: `cpl,`, `scf,`, `ccf,`.

Source file: <src/lib/assembler.fs>.

negate

```
negate ( n1 -- n2 )
```

Negate *n1*, giving its arithmetic inverse *n2*.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012

(CORE).

See also: `?negate`, `0=`, `inverse`, `dnegate`.

Source file: <src/kernel.z80s>.

neither

```
neither ( x1 x2 x3 -- f )
```

Return `true` if *x1* is not equal to either *x2* or *x3*; else return `false`.

Origin: IsForth.

See also: `either`, `ifelse`, `any?`.

Source file: <src/lib/math.operators.1-cell.fs>.

nest-source

```
nest-source ( R: -- source-sys )
```

source-sys describes the current source specification for later use by `unnest-source`.

`nest-source` is a `compile-only` word.

Definition:

```
: nest-source ( R: -- source-sys )
  r>
  input-buffer 2@ 2>r
  source-id >r
  >in @ >r
  blk @ >r
  #tib @ >r
  >r ; compile-only
```

See also: `#tib`, `blk`, `>in`, `(source-id`, `input-buffer`.

Source file: <src/kernel.z80s>.

new

```
new ( class -- o )
```

Create a new incarnation of the class *class*.

Source file: <src/lib/objects.mini-oof.fs>.

new-key

```
new-key ( -- c )
```

Remove all keys from the keyboard buffer, then return character *c* of the key struck, a member of the a member of the defined character set.

See also: `new-key-`, `key`, `xkey`, `-keys`.

Source file: <src/lib/keyboard.MISC.fs>.

new-key-

```
new-key- ( -- ) "new-key-minus"
```

Remove all keys from the keyboard buffer, then wait for a key press and discard it. Finally remove all keys from the keyboard buffer.

See also: `new-key`, `key`, `xkey`, `-keys`.

Source file: <src/lib/keyboard.MISC.fs>.

new-menu

```
new-menu ( a1 a2 ca len col row n1 n2 -- )
```

Set, display an activate a new menu at cursor coordinates *col row*, with *n2* options, *n1* characters width, title *ca len*, actions table *a1* (a cell array of *n2* execution tokens) and option texts table *a2* (a cell array of *n2* addresses of counted strings).

Usage example:

```
need menu need :noname

:noname ( -- ) unnest unnest ;
:noname ( -- ) 2 border ;
:noname ( -- ) 1 border ;
:noname ( -- ) 0 border ;

create actions> , , , ,

here s" EXIT"  s,
here s" Red"   s,
here s" Blue"  s,
here s" Black" s,

create texts> , , , ,

: menu-pars ( -- a1 a2 ca len col row n1 n2 )
  actions> texts> s" Border" 7 7 14 4 ;

menu-pars new-menu
```

See also: `set-menu`, `.menu`, `menu`.

Source file: <src/lib/menu.sinclair.fs>.

new-needed-word

```
new-needed-word ( ca1 len -- ca2 len' )
```

Remove trailing and leading spaces from the word *ca1 len*, which is the parameter of the latest `need` `needed`, `reneed` or `reneeded`, store it in the `stringer` and return it as *ca2 len'* for further processing.

Source file: <src/lib/002.need.fs>.

newline

```
newline ( -- ca len )
```

ca len is a character string containing the character(s) used to mark the start of a new line of text in file operations.

The string is stored at `newline>` as a counted string, which can be configured by the application.

Origin: Gforth.

See also: `'cr'`, `'lf'`.

Source file: <src/lib/display.control.fs>.

newline>

```
newline> ( -- ca ) "new-line-to"
```

ca is the address of a counted string containing the character(s) (maximum 2) used to mark the start of a new line of text in file operations.

The string can be configured by the application. By default it contains only the character `'cr'`.

The string is returned by `newline`.

See also: `'lf'`.

Source file: <src/lib/display.control.fs>.

newton-sqrt

```
newton-sqrt ( n1 -- n2 ) "newton-square-root"
```

Integer square root *n2* of radicand *n1* by Newton's method. `newton-sqrt` is 7..8 times slower than `baden-sqrt`.

Loading `newton-sqrt` makes it the action of `sqrt`.

Source file: <src/lib/math.operators.1-cell.fs>.

next

```
next ( -- a )
```

A `constant`. *a* is the address of the main entry point of the Forth inner interpreter. It is the address Forth words jump to at the end. The code at *a* executes the word whose execution token is in the address pointed by the Forth IP (the Z80 BC register).

In Solo Forth, the Z80 IX register contains *a*, which must be preserved across Forth words.

See also: `pushhl`, `pusha`.

Source file: <src/kernel.z80s>.

nextname

```
nextname ( ca len -- )
```

The next defined word will have the name *ca len*; the defining word will leave the input stream alone. `nextname` works with any defining word.

Origin: Gforth.

See also: `nextname-header`, `nextname-string`.

Source file: <src/lib/define.MISC.fs>.

nextname-header

```
nextname-header ( -- )
```

Create a dictionary header using the name string set by `nextname`. Then restore the default action of `header`.

Origin: Gforth.

See also: `nextname-string`. `default-header`.

Source file: <src/lib/define.MISC.fs>.

nextname-string

```
nextname-string ( -- a )
```

A `2variable`. *a* is the address of a double-cell containing the address and length of a name to be used by the next defining word. This variable is set by `nextname`.

Origin: Gforth.

See also: `nextname-header`.

Source file: <src/lib/define.MISC.fs>.

nip

```
nip ( x1 x2 -- x2 )
```

Drop the first item below the top of stack.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `drop`, `tuck`, `2nip`.

Source file: <src/kernel.z80s>.

nn!

```
nn! ( x[u]..x[1] u a -- ) "n-n-store"
```

Store the count *u* at *a*. If *u* is not zero, store also *u* cells *x[u]..x[1]* at the next cell address, being *x[1]* the first one stored and *x[u]* the last one.

See also: `n!`, `!`, `nn@`.

Source file: <src/lib/memory.MISC.fs>.

nn,

```
nn, ( x[u]..x[1] u -- ) "n-n-comma"
```

Store the count *u* into data space. If *u* is not zero, store also *u* cells *x[u]..x[1]* into data space, being *x[1]* the first one stored and *x[u]* the last one.

See also: `,`, `n,`, `nn!`.

Source file: <src/lib/memory.MISC.fs>.

nn@

```
nn@ ( a -- x[1]..x[u] u | 0 ) "n-n-fetch"
```

Read the count *u* from *a*. If it's zero, return it. If *u* is not zero, read *u* cells *x[u]..x[1]* from the next cell address, being *x[1]* the first cell stored there and *x[u]* the last one.

See also: `n@`, `@`, `nn!`.

Source file: <src/lib/memory.MISC.fs>.

no-exit

```
no-exit  ( -- )
```

Recover the data-space cell used by the `exit` compiled by the `;` of the latest colon definition. `no-exit` can be used after a colon definition that contains and end-less loop, or exits only through an explicit `exit`, `quit` or other means. In such cases the `exit` compiled by `;` can never be reached, so its space is wasted.

Usage examples:

```
: forever ( -- )
  begin ." Forever! " again ; no-exit

: maybe-forever ( -- )
  begin ." Forever? " break-key? until quit ; no-exit
```

The same effect can be achieved by replacing `;` with `[` and `finish-code`:

```
: forever ( -- )
  begin ." Forever!" again [ finish-code

: maybe-forever ( -- )
  begin ." Forever? " break-key? until quit [ finish-code
```

`finish-code` is factor of `;`. It's not an `immediate` word, so `[` is needed to enter interpretation `state`.

Origin: Pygmy Forth's `recover`.

Source file: <src/lib/compilation.fs>.

no-ltyped

```
no-ltyped ( -- ) "no-l-typed"
```

Set `#ltyped` and `#indented` to zero.

See also: `ltyped`.

Source file: <src/lib/display.ltype.fs>.

no-warnings?

```
no-warnings? ( -- f ) "no-warnings-question"
```

Are the warnings deactivated?

See also: `?warn`, `warnings`.

Source file: <src/lib/compilation.fs>.

no?

```
no? ( -- f ) "no-question"
```

Wait for a valid `key` press for a `y/n` question and return `true` if it's the current value of `"n"`, else return `false`.

See also: `yes?`, `y/n?`.

Source file: <src/lib/keyboard.yes-question.fs>.

noname?

```
noname? ( -- a ) "no-name-question"
```

A `variable`. *a* is the address of a cell containing a flag: Was the word being defined created by `:noname`? `noname?` is set by `:noname` and reset by `;`.

Source file: <src/kernel.z80s>.

noop

```
noop ( -- ) "no-op"
```

Do nothing.

See also: `noop_`.

Source file: <src/kernel.z80s>.

noop_

```
noop_ ( -- a ) "no-op-underscore"
```

A `constant`. *a* is the address of a routine that does nothing, except executing a Z80 `ret` to return.

`noop_` is used as the default jump point of `circle-pixel`.

See also: `noop`.

Source file: <src/kernel.z80s>.

nop,

```
nop, ( -- ) "nop-comma"
```

Compile the Z80 `assembler` instruction `NOP`.

Source file: <src/lib/assembler.fs>.

not-block-drive

```
not-block-drive ( -- c )
```

c is a constant identifier used by `set-block-drives`, `-block-drives` and other related words to mark unused elements of `block-drives`.

Source file: <src/lib/dos.COMMON.fs>.

not-redefined?

```
not-redefined? ( ca len -- ca len xt false | ca len true ) "not-redefined-question"
```

Is the word name *ca len* not yet defined in the compilation word list?

See also: `?warn`.

Source file: <src/lib/compilation.fs>.

not-understood

```
not-understood ( -- )
```

`throw` exception code #-256 ("not understood").

`not-understood` is used in `interpret-table`.

See also: `compilation-only`.

Source file: <src/kernel.z80s>.

np

```
np ( -- a ) "n-p"
```

A `constant`. *a* is the address of a cell containing the name-space pointer, which points to the next free address where the next word header will be stored.

Name space is in "far memory": a 64-KiB memory formed by 4 configurable memory banks.

See also: `np0`, `np@`, `dp`, `far-banks`.

Source file: <src/kernel.z80s>.

np!

```
np! ( a -- ) "n-p-store"
```

Store *a* into the name-space pointer `np`.

`np!` is written in Z80. Its equivalent definition in Forth is the following:

```
: np! ( a -- ) np ! ;
```

Source file: <src/lib/memory.far.fs>.

np0

```
np0 ( -- a ) "n-p-zero"
```

A `constant`. *a* is the the bottom (initial) address of the name-space pointer `np`.

Source file: <src/kernel.z80s>.

np@

```
np@ ( -- a ) "n-p-fetch"
```

Fetch the content of the name-space pointer `np`.

`np@` is written in Z80. Its equivalent definition in Forth is the following:

```
: np@ ( -- a ) np @ ;
```

Source file: <src/kernel.z80s>.

nr>

```
nr> ( -- x#1..x#n n ) ( R: x#1..x#n n -- ) "n-r-from"
```

Retrieve the items previously stored by an invocation of `n>r`. *n* is the number of items placed on the data stack.

Origin: Forth-2012 (TOOLS EXT).

Source file: <src/lib/return_stack.fs>.

nstr1

```
nstr1 ( -- ca )
```

Return address *ca* of the directory description in the current `ufia`.

See also: `dstr1`, `fstr1`, `sstr1`, `device`, `nstr2`, `hd00`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

nstr2

```
nstr2 ( -- ca )
```

Return address *ca* of the 10-character file name in the current `ufia`.

See also: `dstr1`, `fstr1`, `sstr1`, `device`, `nstr1`, `hd00`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

nuf?

```
nuf? ( -- f ) "nuf-question"
```

If no key is pressed return `false`. If a key is pressed, discard it and wait for a second key. Then return `true` if it's a carriage return, else return `false`.

Usage example:

```
: listing ( -- )
  begin  ." bla " nuf?  until  ." Aborted" ;
```

See also: `aborted?`.

Source file: <src/lib/keyboard.MISC.fs>.

number

```
number ( ca len -- n | d )
```

Attempt to convert a string *ca len* into a number. If a valid point is found, return *d*; if there is no valid point, return *n*. If conversion fails due to an invalid character, `throw` an exception #-275 ("wrong number").

See also: `number?`, `>number`.

Source file: <src/lib/math.number.conversion.fs>.

number-base

```
number-base ( ca len -- ca' len' n )
```

If the first character of string *ca len* is a radix prefix, return its value *n* and the updated string *ca' len'* (which does not include the radix prefix). Otherwise return *ca len* untouched and the current value of `base` *n*.

Definition:

```
: number-base ( ca len -- ca' len' n )
  dup if
   over c@
   dup '$' = if  drop 1 /string #16  exit  then
   dup '%' = if  drop 1 /string  #2  exit  then
       '#' = if       1 /string #10  exit  then
  then
  base @ ;
```

Source file: <src/kernel.z80s>.

number-point?

```
number-point? ( c -- f ) "number-point-question"
```

f is true if character *c* is a valid point in a number. `number-point?` is a deferred word (see `defer`) used in `number?`. Its default action is `standard-number-point?`, which only allows the period.

See also: `classic-number-point?`, `extended-number-point?`, `dpl`.

Source file: <src/kernel.z80s>.

number?

```
number? ( ca len -- 0 | n 1 | d 2 ) "number-question"
```

Convert a string *ca len* to a number, using the current value of `base`.. Return 0 if the conversion is not possible. If the result is a single number, return *n* and 1. If the result is a double number, return *d* and 2.

`number?` accepts valid point anywhere on the number and updates `dpl` with the position of the last one. If no point is found, `dpl` contains -1.

Characters between single quotes are recognized, after Forth-2012.

Definition:

```
: number? ( ca len -- 0 | n 1 | d 2 )

  dup 0= if  2drop 0 exit  then  \ reject empty strings

  2dup char? if  nip nip 1 exit  then  \ character format

  over c@ number-point?  \ first character is a point?
  if  2drop 0 exit  then  \ is so, reject the string

  base @ >r  number-base base ! ( R: radix )
  skip-sign? >r                 ( R: radix sign )
  0 0 2swap  dpl on

  begin ( d ca len ) >number dup  while

    over c@ number-point? 0=   \ invalid point?
    if  2drop 2drop rdrop r> base ! 0 exit  then

    dup dpl @ =   \ previous character was a point?
    if  2drop 2drop rdrop r> base ! 0 exit  then

    dup 1- dpl !  \ update the position of the last point
    1 /string     \ skip the point

  repeat

  2drop                     \ discard the empty string
  dpl @ 0<                  \ single-cell number?
  if    d>s r> ?negate  1   \ single-cell number
  else  r> ?dnegate  2      \ double-cell number
  then  r> base ! ;         \ restore the radix
```

See also: `>number`, `number-point?`, `skip-sign?`, `dpl`, `number`.

Source file: <src/kernel.z80s>.

nup

```
nup ( x1 x2 -- x1 x1 x2 )
```

This word is defined in Z80. Its equivalent definition in Forth is the following:

```
: nup ( x1 x2 -- x1 x1 x2 ) over swap ;
```

See also: `dup`, `tuck`, `drup`, `dip`.

Source file: <src/lib/data_stack.fs>.

nx

```
nx ( -- ) "n-x"
```

Give next quick index, calculated from `scr`.

See also: `qx`, `px`.

Source file: <src/lib/tool.list.blocks.fs>.

nz?

```
nz? ( -- op ) "n-z-question"
```

Return the opcode *op* of the Z80 `assembler` instruction `jp nz`, to be used as condition and consumed by `?ret,`, `?jp,`, `?call,`, `?jr,`, `aif`, `rif`, `awhile`, `rwhile`, `auntil` or `runtil`.

See also: `z?`, `c?`, `nc?`, `po?`, `pe?`, `p?`, `m?`.

Source file: <src/lib/assembler.fs>.

O

object

```
object ( -- a )
```

The base class of all objets.

Source file: <src/lib/objects.mini-oof.fs>.

ocr

```
ocr ( col row -- c | 0 ) "o-c-r"
```

Try to recognize the character printed at the given cursor coordinates, using the character set whose first printable character is pointed by the variable `ocr-font`. The character variable `ocr-chars` contains the number of characters in the set, and its counterpart `ocr-first` contains the code of its first character. If succesful, return the character number *c* according to the said variables. Otherwise return 0. Inverse characters are not recognized.

NOTE The name `ocr` stands for "Optical Character Recognition".

See also: `udg-ocr`, `ascii-ocr`.

Source file: <src/lib/graphics.ocr.fs>.

ocr-chars

```
ocr-chars ( -- ca ) "o-c-r-chars"
```

A `cvariable`. *ca* is the address of a byte containing the number of characters used by `ocr`, from the address pointed by `ocr-font`. By default it contais 95, the number of printable ASCII characters in the ROM character set.

The configuration of `ocr`, including this variable, can be changed by `ascii-ocr` and `udg-ocr`.

See also: `ocr-first`, `ocr-font`.

Source file: <src/lib/graphics.ocr.fs>.

ocr-first

```
ocr-first ( -- ca ) "o-c-r-first"
```

A `cvariable`. *ca* is the address of a byte containing the code of the first printable character in the character set used by `ocr`, pointed by `ocr-font`. By default it contais `bl`, the code of the space character.

The configuration of `ocr`, including this variable, can be changed by `ascii-ocr` and `udg-ocr`.

See also: `ocr-chars`, `ocr-font`.

Source file: <src/lib/graphics.ocr.fs>.

ocr-font

```
ocr-font ( -- a ) "o-c-r-font"
```

A `variable`. *a* is the address of a cell containing the address of the first printable character in the character set used by `ocr`. By default it contains 0x3D00, the address of the space character in the `rom-font`.

The configuration of `ocr`, including this variable, can be changed by `ascii-ocr` and `udg-ocr`.

See also: `ocr-chars`, `ocr-first`.

Source file: <src/lib/graphics.ocr.fs>.

octave-changer

```
octave-changer ( -- a )
```

a is the address of an execution table that contains the three execution tokens used to calculate the frequency of notes from any octave. *a* is the address of the second execution token (cell offset 0).

See also: `change-octave`, `beep>dhz`, `middle-octave`.

Source file: <src/lib/sound.48.fs>.

odd?

```
odd? ( n -- f ) "odd-question"
```

Is *n* an odd number?

`odd?` is written in Z80. Its equivalent definition in Forth is the following:

```
: odd? ( n -- f ) 1 and 0<> ;
```

See also: `odd?`.

Source file: <src/lib/math.operators.1-cell.fs>.

of

```
of
  Compilation: ( C: -- orig )
  Run-time:    ( x1 x2 -- )
```

`of` is an `immediate` and `compile-only` word.

Compilation: Put *orig* onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *orig* such as `endof`.

Run-time: If *x1* and *x2* are not equal, discard *x2* and continue execution at the location specified by the consumer of *orig*, e.g. following the next `endof`. Otherwise discard *x1 x2* and continue execution in line.

`of` is an `immediate` and `compile-only` word.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `default-of`, `less-of`, `greater-of`, `between-of`, `within-of`, `or-of`, `any-of`.

Source file: <src/lib/flow.case.fs>.

off

```
off ( a -- )
```

Store `false` at *a*.

`off` is written in Z80. Its equivalent definition in Forth is the following:

```
: off ( a -- ) false swap ! ;
```

Origin: Comus.

See also: `on`, `coff`.

Source file: <src/kernel.z80s>.

ok

```
ok ( -- )
```

A deferred word (see `defer`) called by `quit` after interpreting a command. Its default action is the word `.ok`.

Source file: <src/kernel.z80s>.

omn1-64cpl-font

```
omn1-64cpl-font ( -- a ) "omn-1-64-c-p-l-font"
```

a is the address of a 4x8-pixel font compiled in data space (336 bytes used), to be used in `mode-64ao` by setting `mode-64-font` first.

This font is included also in disk 0 as "omn1.f64".

See also: `mini-64cpl-font`, `nbot-64cpl-font`, `omn2-64cpl-font`, `owen-64cpl-font`.

Source file: <src/lib/display.mode.64.COMMON.fs>.

omn2-64cpl-font

```
omn2-64cpl-font ( -- a ) "omn-2-64-c-p-l-font"
```

a is the address of a 4x8-pixel font compiled in data space (336 bytes used), to be used in `mode-64ao` by setting `mode-64-font` first.

This font is included also in disk 0 as "omn2.f64".

See also: `mini-64cpl-font`, `nbot-64cpl-font`, `omn1-64cpl-font`, `owen-64cpl-font`.

Source file: <src/lib/display.mode.64.COMMON.fs>.

on

```
on ( a -- )
```

Store `true` at *a*.

`on` is written in Z80. Its equivalent definition in Forth is the following:

```
: on ( a -- ) true swap ! ;
```

Origin: Comus.

See also: `off`, `con`.

Source file: <src/kernel.z80s>.

only

```
only ( -- )
```

Set the search order to the minimum search order.

Definition:

```
: only ( -- ) -1 set-order ;
```

Origin: Forth-94 (SEARCH EXT), Forth-2012 (SEARCH EXT).

See also: `also`, `set-order`, `previous`, `order`.

Source file: <src/kernel.z80s>.

only-one-pressed

```
only-one-pressed ( -- false | b a true )
```

Return the key identifier *b a* (key bitmask and keyboard row port) of the only key from table `kk-`

`ports` that happens to be pressed, and `true`; if no key is pressed or more than one key is pressed at the same time, return `false`.

See also: `pressed`, `pressed?`.

Source file: <src/lib/keyboard.MISC.fs>.

option

```
option ( x "name" -- )
```

Compile the action *name* of an option *x* in an `options[ … ]options` control structure.

See `options[` for a usage example.

Source file: <src/lib/flow.options-bracket.fs>.

options-table

```
options-table ( -- a )
```

A `variable`. *a* is the address of a cell containing the address of a cell array, which holds the counted strings of the current `menu` options. `options-table` is set by `set-menu`.

See also: `actions-table`.

Source file: <src/lib/menu.sinclair.fs>.

options[

```
options[ "options-left-bracket"
```

Compilation: (— a1 a2 a3)

Start an `options[ … ]options` structure.

The addresses left on the stack will be resolved by `]options`:

- a1 = address of exit point
- a2 = address of the xt of the default option
- a3 = address of number of options

Usage example:

```
: say10      ." dek" ;
: say100     ." cent" ;
: say1000    ." mil" ;
: say-other  ." alia" ;

: say ( n )
  options[
    10 option  say10
   100 option  say100
  1000 option  say1000
       default-option say-other
  ]options ;

10 say  100 say  1000 say  1001 say
```

`options[` is an `immediate` and `compile-only` word.

Source file: <src/lib/flow.options-bracket.fs>.

or

```
or ( x1 x2 -- x3 )
```

x3 is the bit-by-bit inclusive-or of *x1* with *x2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `and`, `xor`, `negate`, `0=`, `dor`.

Source file: <src/kernel.z80s>.

or#,

```
or#, ( b -- ) "or-number-sign-comma"
```

Compile the Z80 `assembler` instruction `OR b`.

See also: `xor#,`, `and#,`, `add#,`.

Source file: <src/lib/assembler.fs>.

or,

```
or, ( reg -- ) "or-comma"
```

Compile the Z80 `assembler` instruction `OR reg`.

See also: `and,`, `xor,`.

Source file: <src/lib/assembler.fs>.

or-of

```
or-of
  Compilation: ( C: -- of-sys )
  Run-time:    ( x1 x2 x3 -- | x1 )
```

A variant of `of`.

Compilation:

Put *of-sys* onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *of-sys*, such as `endof`.

Run-time:

If *x1* is equal to *x2* or *x1* is equal to *x3* discard *x1 x2 x3* and continue execution in line; otherwise discard *x2 x3* and continue execution at the location specified by the consumer of *of-sys*, e.g., following the next `endof`.

`or-of` is an `immediate` and `compile-only` word.

Usage example:

```
: test ( x -- )
  case
    1      of ." one"          endof
    2 3 or-of ." two or three" endof
    4      of ." four"         endof
  endcase ;
```

See also: `case`, `any-of`, `(or-of`.

Source file: <src/lib/flow.case.fs>.

order

```
order ( -- )
```

Display the word lists in the search order in their search order sequence, from first searched to last searched. Also display the word list into which new definitions will be placed.

Origin: Forth-2012 (SEARCH EXT).

See also: `.context`, `.current`, `.wordlist`, `set-order`.

Source file: <src/lib/tool.list.word_lists.fs>.

order,

```
order, ( -- )
```

Compile the current search order by executing `get-order` and `nn,`.

`order,` is a useful factor of `marker`.

See also: `@order`, `wordlists,`.

Source file: <src/lib/tool.marker.fs>.

orif

```
orif "or-if"
  Compilation: ( C: -- orig )
  Run-time:    ( f -- )
```

Short-circuit `or` variant of `if`.

`orif` is an `immediate` and `compile-only` word.

Usage example:

```
: is-alphanum? ( c -- f ) cond  dup is-lower? orif
                                dup is-upper? orif
                                dup is-digit?
                          thens nip ;
```

Compare with the following equivalent definition, where all three conditions are always checked:

```
: is-alphanum? ( c -- f ) dup  is-lower?
                          over is-upper? or
                          swap is-digit? or ;
```

See also: `andif`, `cond`, `thens`.

Source file: <src/lib/flow.MISC.fs>.

orthodraw

```
orthodraw ( gx gy gxinc gyinc len -- )
```

Draw a line formed by *len* pixels, starting from *gx gy* and using *gxinc gyinc* as increments to calculate the coordinates of every next pixel.

The status of `inverse` and `overprint` modes are obeyed; the screen attributes and the system graphic coordinates are updated. That's what makes `orthodraw` much slower than `ortholine`.

See also: `adraw176`, `rdraw176`.

Source file: <src/lib/graphics.lines.fs>.

ortholine

```
ortholine ( gx gy gxinc gyinc len -- )
```

Draw a line formed by *len* pixels, starting from *gx gy* and using *gxinc gyinc* as increments to calculate the coordinates of every next pixel.

The status of `inverse` and `overprint` modes is ignored; the attributes of the screen are not modified; and the system graphic coordinates are not updated. That's what makes `ortholine` almost twice faster than `orthodraw`.

Source file: <src/lib/graphics.lines.fs>.

orx,

```
orx, ( disp regpi --  ) "or-x-comma"
```

Compile the Z80 `assembler` instruction `OR (regpi+disp)`.

See also: `andx,`, `xorx,`, `cpx,`.

Source file: <src/lib/assembler.fs>.

os-attr-p

```
os-attr-p ( -- ca ) "o-s-attribute-p"
```

A `constant` that returns the address *ca* of 1-byte system variable ATTR_P, which holds the current permanent color attribute, as set up by color statements.

See also: `os-attr-t`, `os-mask-p`.

Source file: <src/lib/os.variables.fs>.

os-attr-t

```
os-attr-t ( -- ca ) "o-s-attribute-t"
```

A `constant` that returns the address *ca* of 1-byte system variable ATTR_T, which holds the current temporary color attribute, as set up by color statements.

See also: `os-attr-p`, `os-mask-t`.

Source file: <src/lib/os.variables.fs>.

os-chans

```
os-chans ( -- a ) "o-s-chans"
```

A `constant` that returns the address *a* of the system variable CHANS, which holds the address of the channel data table. Each element of the table has the following structure:

Table 30. Structure of a system channel.

Offset (bytes)	Content
+0	Address of the channel output routine
+2	Address of the channel input routine
+4	Channel identifier character

The default contents of the channel data table are the following:

Table 31. Default system channel data table.

Offset (bytes)	Content
+0	$09F4 (print-out)
+2	$10A8 (key-input)
+4	'K'
+5	$09F4 (print-out)
+7	$15C4 (report-j)
+9	'S'
+10	$08F1 (add-char)
+12	$15C4 (report-j)
+14	'R'
+15	$09F4 (print-out)
+17	$15C4 (report-j)
+19	'P'

NOTE The elements of the channel data table are pointed from `os-strms` by 1-indexed byte offsets, i.e. $0001 points to the first element of the channel data table, channel 'K'.

See also: `.os-chans`.

Source file: <src/lib/os.variables.fs>.

os-chars

```
os-chars ( -- a ) "o-s-chars"
```

A `constant` that returns the address of system variable CHARS, which holds the bitmap address of character 0 of the current font (actual characters 32..127). By default this system variables holds ROM address 15360 ($3C00).

See also: `set-font`, `get-font`, `rom-font`, `os-udg`.

Source file: <src/lib/os.variables.fs>.

os-coords

```
os-coords ( -- a ) "o-s-coords"
```

A `constant` that returns the address *a* of 2-byte system variable COORDS which holds the graphic coordinates of the last point plotted.

See also: `set-pixel`, `plot`, `os-coordx`, `os-coordy`.

Source file: <src/lib/os.variables.fs>.

os-coordx

```
os-coordx ( -- ca ) "o-s-coord-x"
```

A `constant` that returns the address *ca* of 1-byte system variable COORDX which holds the graphic x coordinate of the last point plotted.

See also: `set-pixel`, `plot`, `os-coords`, `os-coordy`.

Source file: <src/lib/os.variables.fs>.

os-coordy

```
os-coordy ( -- ca ) "o-s-coord-y"
```

A `constant` that returns the address *ca* of 1-byte system variable COORDY which holds the graphic y coordinate of the last point plotted.

See also: `set-pixel`, `plot`, `os-coords`, `os-coordx`.

Source file: <src/lib/os.variables.fs>.

os-flags2

```
os-flags2 ( -- ca ) "o-s-flags-two"
```

A `constant` that returns the address *ca* of 1-byte system variable FLAGS2, which holds several flags.

See also: `capslock`.

Source file: <src/lib/os.variables.fs>.

os-frames

```
os-frames ( -- a ) "o-s-frames"
```

A `constant` that returns the address *a* of the 24-bit system variable FRAMES (least significant byte first), containing the counter of frames, which is incremented every 20 ms by the interrupt routine of the OS. This counter is returned by `ticks` and used by its related words.

See also: `set-ticks`, `reset-ticks`, `ticks-pause`, `?ticks-pause`.

Source file: <src/lib/os.variables.fs>.

os-mask-p

```
os-mask-p ( -- ca ) "o-s-mask-p"
```

A `constant` that returns the address *ca* of 1-byte system variable MASK_P, which holds the permanent color attribute mask, used for transparent colors, etc. Any bit that is 1 shows that the corresponding attribute bit is taken not from `os-attr-p` but from what is already on the screen.

See also: `os-attr-p`, `os-mask-t`.

Source file: <src/lib/os.variables.fs>.

os-mask-t

```
os-mask-t ( -- ca ) "o-s-mask-t"
```

A `constant` that returns the address *ca* of 1-byte system variable MASK_T, which holds the temporary color attribute mask, used for transparent colors, etc. Any bit that is 1 shows that the corresponding attribute bit is taken not from `os-attr-t` but from what is already on the screen.

See also: `os-attr-t`, `os-mask-p`.

Source file: <src/lib/os.variables.fs>.

os-p-flag

```
os-p-flag ( -- ca ) "o-s-p-flag"
```

A `constant` that returns the address *ca* of 1-byte system variable P_FLAG, which holds some flags related to printing.

Source file: <src/lib/os.variables.fs>.

os-prog

```
os-prog ( -- a ) "o-s-prog"
```

A `constant` that returns the address *a* of 2-byte system variable PROG which holds the address of the BASIC program.

See also: `os-stkend`, `os-ramtop`, `os-chans`.

Source file: <src/lib/os.variables.fs>.

os-ramtop

```
os-ramtop ( -- a ) "o-s-ram-top"
```

A `constant` that returns the address *a* of 2-byte system variable RAMTOP which holds the address of the last byte of BASIC system area.

See also: `os-stkend`, `os-prog`, `os-chans`.

Source file: <src/lib/os.variables.fs>.

os-seed

```
os-seed ( -- a ) "o-s-seed"
```

A `constant` that returns the address *a* of system variable SEED, which holds the seed of the BASIC random number generator.

Source file: <src/lib/os.variables.fs>.

os-sp

```
os-sp ( -- a ) "os-s-p"
```

A `variable`. *a* is the address of a cell containing a copy of the OS stack pointer, which is saved when the Forth system is entered from BASIC, and then restored by `(bye` before returning to BASIC.

Source file: <src/kernel.z80s>.

os-stkend

```
os-stkend ( -- a ) "o-s-stack-end"
```

A `constant` that returns the address *a* of 2-byte system variable STKEND which holds the address of the start of spare space of BASIC system area.

See also: `os-prog`, `os-chans`.

Source file: <src/lib/os.variables.fs>.

os-strms

```
os-strms ( -- a ) "o-s-streams"
```

A `constant` that returns the address *a* of a 38-byte (19-cell) system variable STRMS which holds one cell per stream, containing the address of the channel attached to it, as follows:

Table 32. Structure of the system streams table.

Offset (cells)	Stream	Content
+0	-3	$0001 (offset to channel 'K')
+1	-2	$0006 (offset to channel 'S')
+2	-1	$000B (offset to channel 'R')
+3	0	$0001 (offset to channel 'K')
+4	1	$0001 (offset to channel 'K')
+5	2	$0006 (offset to channel 'S')
+6	3	$0010 (offset to channel 'P')
+7..+18	4..15	$0000..$0000 (not attached)

NOTE The contents are 1-index offsets from the address `os-chans`. When the content of a stream cell is zero, the stream is not attached to a channel.

See also: `.os-strms`.

Source file: <src/lib/os.variables.fs>.

os-udg

```
os-udg ( -- a ) "o-s-u-d-g"
```

A `constant` that returns the address *a* of system variable UDG, which holds the address of the first character bitmap of the current User Defined Graphics set (characters 128..255 or 0..255, depending on the words used to access them).

See also: `set-udg`, `get-udg`, `os-chars`.

Source file: <src/lib/os.variables.fs>.

os-unused

```
os-unused ( -- u ) "o-s-unused"
```

u is the amount of unused space by the OS and the BASIC interpreter.

See also: `unused`, `farunused`.

Source file: <src/lib/os.fs>.

othercase

```
othercase ( x -- )
```

Mark the default option of a `thiscase` structure that checked *x*.

See also: `ifcase`, `exitcase`.

Source file: <src/lib/flow.thiscase.fs>.

othercase>

```
othercase> ( orig counter "name" -- ) "other-case-from"
```

Compile the default option of a `cases:` to be the word *name*. This must be the last option of the structure and is mandatory. When no default action is required, `othercase> noop` can be used.

See `cases:` for an usage example.

Source file: <src/lib/flow.cases-colon.fs>.

out,

```
out, ( b -- ) "out-comma"
```

Compile the Z80 `assembler` instruction `OUT (b),A`.

See also: `in,`, `outbc,`.

Source file: <src/lib/assembler.fs>.

outbc,

```
outbc, ( reg -- ) "out-b-c-comma"
```

Compile the Z80 `assembler` instruction `OUT ©,reg`.

See also: `inbc,`, `out,`.

Source file: <src/lib/assembler.fs>.

outlet-autochars

```
outlet-autochars ( a -- )
```

Create a modified, bolder copy of the ZX Spectrum ROM font and store it at *a*. 768 bytes will be used from *a*. Then activate the new font by modifing the contents of `os-chars`.

The code of `outlet-autochars` has been adapted from the Autochars routine used by the Outlet magazine, published in its issue #1 (1987-09).

Usage example:

```
create outlet-font  768 allot
need outlet-autochars
outlet-font outlet-autochars
```

See also: `set-font`, `rom-font`.

Source file: <src/lib/display.fonts.fs>.

over

```
over ( x1 x2 -- x1 x2 x1 )
```

Place a copy of *x1* on top of the stack.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `dup`, `swap`, `2over`.

Source file: <src/kernel.z80s>.

overprint

```
overprint ( f -- )
```

If *f* is zero, turn the overprint mode off; else turn it on.

See also: `overprint-on`, `overprint-off`, `inverse`.

Source file: <src/lib/display.attributes.fs>.

overprint-off

```
overprint-off ( -- )
```

Turn the overprint mode off.

See also: `overprint-on`, `overprint`, `inverse-off`.

Source file: <src/lib/display.attributes.fs>.

overprint-on

```
overprint-on ( -- )
```

Turn the overprint mode on.

See also: `overprint-off`, `overprint`, `inverse-on`.

Source file: <src/lib/display.attributes.fs>.

owen-64cpl-font

```
owen-64cpl-font ( -- a ) "owen-64-c-p-l-font"
```

a is the address of a 4x8-pixel font compiled in data space (336 bytes used), to be used in `mode-64ao` by setting `mode-64-font` first.

This font is included also in disk 0 as "owen.f64".

See also: `mini-64cpl-font`, `nbot-64cpl-font`, `omn1-64cpl-font`, `omn2-64cpl-font`.

Source file: <src/lib/display.mode.64.COMMON.fs>.

p

p

```
p ( -- )
```

A command of `gforth-editor`: Go to previous screen.

See also: `n`, `c`, `a`, `g`, `t`, `scr`, `top`.

Source file: <src/lib/prog.editor.gforth.fs>.

p

```
p ( n "ccc<eol>" -- )
```

A command of `specforth-editor`: Put *ccc* on line *n*.

See also: `b`, `c`, `d`, `e`, `f`, `h`, `i`, `l`, `m`, `n`, `r`, `s`, `t`, `x`, `text`.

Source file: <src/lib/prog.editor.specforth.fs>.

p?

```
p? ( -- op ) "p-question"
```

Return the opcode *op* of the Z80 `assembler` instruction `jp p`, to be used as condition and consumed by `?ret,`, `?jp,`, `?call,`, `aif`, `awhile` or `auntil`.

See also: `z?`, `nz?`, `c?`, `nc?`, `po?`, `pe?`, `m?`.

Source file: <src/lib/assembler.fs>.

package

```
package ( "name" -- wid0 wid1 )
```

If the package *name* has been previously defined, open it. Otherwise create it.

wid1 is the word list of the package *name*; *wid0* is the word list in which the package *name* was created.

`end-package` ends the package; `public` start public definitions and `private` starts private definitions.

Syntax:

```
package package-name
... private definitions here ...
public
... public definitions here ...
private
... more private definitions maybe ...
end-package
```

In the above, private definitions are placed in the `package-name` word list. Public definitions are placed in whatever word list was current before `package package-name`. If a package called `package-name` already exists prior to the above, then it is reused, rather than redefined.

Usage example:

```
package example

defer text
: ex1 ( -- ca len ) s" This is an example" ;
' ex1 ' text defer!

public

: .example ( -- ) text cr type ;

private

: ex2 ( -- ca len ) s" This is an example (cont.)" ;

end-package
```

At this point, `.example` is a new word in whatever the current wordlist was, and `text`, `ex1` and `ex2` are all words in the `example` word list. `example` itself is created in the current wordlist if it didn't already exist. (if `example` exists and **isn't** a package, this is an unchecked error which will probably be revealed when `public` runs.)

If this code is in a library, code including the library can then run `.example` freely.

If there's some need to reopen the package, this is easily done:

```
package example

:noname ( -- ca len ) s" This is yet another example" ; '
text defer!

end-package

.example
```

Use case: loading a package using library with a prelude:

Suppose that you need to load a package with some alien definitions, you can put them in a package with the same name before loading the code, and this will only affect that package:

```
package some-package

\ This package's code relies on ``place`` appending a nul byte.

: place ( ca1 len1 ca2 -- ) 2dup + 0 swap c!  place ;

end-package

include some-lib.fs
```

Origin: SwiftForth.

See also: `internal`, `isolate`, `module`, `privatize`, `seclusion`.

Source file: <src/lib/modules.package.fs>.

pad

```
pad ( -- ca )
```

ca is the address of a transient region that can be used to hold data for intermediate processing. It's a fixed offset (`/hold` bytes) above `here`.

Definition:

```
: pad ( -- ca ) here /hold + ;
```

`pad` is specifically intended as a programmer convenience. No standard words use it.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `/pad`.

Source file: <src/kernel.z80s>.

padding-spaces

```
padding-spaces ( len1 len2 -- )
```

If *len2* minus *len1* is a positive number, display that number of spaces; else do nothing.

See also: `type-left-field`.

Source file: <src/lib/display.type.fs>.

page

```
page ( -- )
```

Move to another page for output. On a terminal, `page` clears the screen and resets the cursor position to the upper left corner. On a printer, `page` performs a form feed.

Origin: Forth-79 (Reference Word Set), Forth-83 (Uncontrolled Reference Words), Forth-94 (FACILITY), Forth-2012 (FACILITY).

See also: `cls`.

Source file: <src/kernel.z80s>.

pair=

```
pair= ( x1 x2 x3 x4 -- f )
```

f is true if and only if *x1 x2* is the same pair as *x3 x4*, i.e. both components of the each pair are in the other pair, no matter the order.

See also: `str<>`, `min>top`, `max>top`.

Source file: <src/lib/math.operators.1-cell.fs>.

paper-mask

```
paper-mask ( -- b )
```

A `cconstant`. *b* is the bitmask of the bits used to indicate the paper in an attribute byte.

See also: `unpaper-mask`, `papery`, `set-paper`, `attr!`, `ink-mask`, `bright-mask`, `flash-mask`.

Source file: <src/lib/display.attributes.fs>.

paper.

```
paper. ( b -- ) "paper-dot"
```

Set paper color to *b* (0..9), by printing the corresponding control characters. If *b* is greater than 9, 9 is used instead.

`paper.` is much slower than `set-paper` or `attr!`, but it can handle pseudo-colors 8 (transparent) and 9 (contrast), setting the corresponding system variables accordingly.

See also: `ink.`, `(0-9-color.`.

Source file: <src/lib/display.attributes.fs>.

papery

```
papery ( b1 -- b2 )
```

Convert paper color *b1* to its equivalent attribute *b2*.

`papery` is an alias of `8*`, which is written in Z80.

See also: `brighty`, `flashy`, `attr>paper`, `contrast`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

parse

```
parse ( char "ccc<char>" -- ca len )
```

Parse *ccc* delimited by the delimiter *char*. *ca* is the address (within the input buffer) and *len* is the length of the parsed string. If the parse area was empty, the resulting string has a zero length.

Definition:

```
: parse ( char "ccc<char>" -- ca len )
  stream 2dup 2>r rot scan
  dup if  char-  then
  2r> rot - parsed
  tuck - ;
```

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `stream`, `scan`, `parsed`, `parse-name`, `parse-char`, `parse-string`.

Source file: <src/kernel.z80s>.

parse-all

```
parse-all ( "ccc" -- ca len )
```

Parse the rest of the source.

`parse-all` is a useful factor of `text`, which is part of `specforth-editor`.

Source file: <src/lib/parsing.fs>.

parse-char

```
parse-char ( "c" -- c )
```

Parse the next character in the input stream and return its code *c*.

See also: `parse-name`, `parse`, `parsed`, `stream`.

Source file: <src/lib/parsing.fs>.

parse-esc-char>chars

```
parse-esc-char>chars ( "c" -- c[n-1]..c[0] n ) "parse-esc-char-to-chars"
```

Parse and translate a escaped char 'c' to a number of chars *c[n-1]..c[0] and their count _n*.

The translation is done by searching the name of the escaped char in the current search order, which has been set by calling `set-esc-order` in `parse-esc-string`.

Source file: <src/lib/strings.escaped.fs>.

parse-esc-string

```
parse-esc-string ( "ccc<quote>"  -- ca len )
```

Parse a text string delimited by a double quote, translating some configurable characters that are escaped with a backslash. Return the translated string *ca len* in the `stringer`.

The characters that must be escaped depend on the search order set by `set-esc-order`. By default, the escaped characters are those described in Forth-2012’s `s\"`.

`parse-esc-string` is a common factor of `s\"` and `.\"`.

See also: `(parse-esc-string`.

Source file: <src/lib/strings.escaped.fs>.

parse-name

```
parse-name ( "name" -- ca len )
```

Parse *name* and return it as string *ca len* within the input buffer. If the parse area is empty or contains only white space, the *len* is zero.

Definition:

```
: parse-name ( "name" -- ca len )
  stream               ( ca0 len0 )
  dup >r   -leading    ( ca1 len1 ) ( R: len0 )
  over >r  bl scan     ( ca2 len2 ) ( R: len0 ca1 )
  dup if  char-  then  \ skip trailing delimiter
  r> r> rot -  parsed  \ update ``&#62;in``
  tuck -               ( ca len )
  2dup parsed-name 2! ;
```

Origin: Forth-2012 (CORE EXT).

See also: `parse`, `parse-name-thru`, `parse-char`, `word`, `parse-string`, `stream`, `scan`, `parsed`, `parsed-name`, `>in`, `-leading`.

Source file: <src/kernel.z80s>.

parse-name-thru

```
parse-name-thru  ( "name" -- ca len )
```

Parse *name* and return it as string *ca len* within the input buffer. If the parse area is empty, use `refill` to fill it from the input source. If the input source is exhausted, `throw` an exception #-289 ("input source exhausted").

See also: `parse-name`, `parse`.

Source file: <src/lib/parsing.fs>.

parse-string

```
parse-string
  Compilation:    ( c "ccc<char>" -- )
  Interpretation: ( c "ccc<char>" -- ca len )
  Run-time:       ( -- ca len )
```

Parse *ccc* delimited by character *c*. If interpreting, copy the parsed string to the `stringer` and return it as *ca len*. If compiling, compile the parsed string and return it at run-time as *ca len*.

WARNING `parse-string` is a state-smart word (see `state`).

Definition:

```
: parse-string \ Compilation:    ( c "ccc<char>" -- )
               \ Interpretation: ( c "ccc<char>" -- ca len )
               \ Run-time:       ( -- ca len )
  parse compiling? if postpone sliteral exit then >stringer ;
```

See also: `parse-name`, `compiling?`, `sliteral`, `>stringer`, `parse-char`, `parse`.

Source file: <src/kernel.z80s>.

parsed

```
parsed ( len -- )
```

Add the given *len* to `>in`.

Definition:

```
: parsed ( len -- ) >in +! ;
```

See also: `parse`.

Source file: <src/kernel.z80s>.

parsed-name

```
parsed-name ( -- a )
```

A `variable`. *a* is the address of a double cell containing the address and length of the most recently name parsed by `parse-name`. It is displayed by `.error-word`.

As a special case, `parsed-name` is set also by `?located`.

Source file: <src/kernel.z80s>.

past?

```
past? ( u -- f ) "past-question"
```

Return true if the `ticks` clock has passed *u*.

Usage example: The following word will execute the hypothetical word `test` for *u* clock `ticks`:

```
: try ( u -- ) ticks + begin test dup past? until drop ;
```

Origin: lina.

See also: `dpast?`, `elapsed`, `timer`.

Source file: <src/lib/time.fs>.

pe?

```
pe? ( -- op ) "p-e-question"
```

Return the opcode *op* of the Z80 `assembler` instruction `jp pe`, to be used as condition and consumed by `?ret,`, `?jp,`, `?call,`, `aif`, `awhile` or `auntil`.

See also: `z?`, `nz?`, `c?`, `nc?`, `po?`, `p?`, `m?`.

Source file: <src/lib/assembler.fs>.

perform

```
perform ( a  -- )
```

If the cell stored at *a* is zero, do nothing. Otherwise execute it as an execution token.

`perform` is written in Z80. Its equivalent definition in Forth is the following:

```
: perform ( a  -- ) @ ?dup if execute then ;
```

NOTE `perform` is called `@execute` in other Forth systems.

See also: `execute`, `+perform`.

Source file: <src/kernel.z80s>.

perm-attr!

```
perm-attr! ( b -- ) "perm-attribute-store"
```

Set *b* as the permanent attribute.

NOTE Words that use attributes don’t use the OS permanent attribute but the temporary one, which is called "current attribute" in Solo Forth.

See also: `perm-attr@`, `attr!`.

Source file: <src/lib/display.attributes.fs>.

perm-attr-mask!

```
perm-attr-mask! ( b -- ) "perm-attribute-mask-store"
```

Set *b* as the permanent attribute mask.

NOTE Words that use attributes don’t use the OS permanent attribute but the temporary one, which is called "current attribute" in Solo Forth.

See also: `perm-attr-mask@`, `attr-mask!`.

Source file: <src/lib/display.attributes.fs>.

perm-attr-mask@

```
perm-attr-mask@ ( -- b ) "perm-attribute-mask-fetch"
```

Get the permanent attribute mask *b*.

NOTE Words that use attributes don’t use the OS permanent attribute but the temporary one, which is called "current attribute" in Solo Forth.

See also: `perm-attr-mask!`, `attr-mask@`.

Source file: <src/lib/display.attributes.fs>.

perm-attr@

```
perm-attr@ ( -- b ) "perm-attribute-fetch"
```

Get the permanent attribute *b*.

NOTE Words that use attributes don’t use the OS permanent attribute but the temporary one, which is called "current attribute" in Solo Forth.

See also: `perm-attr!`, `attr@`.

Source file: <src/lib/display.attributes.fs>.

pick

```
pick ( x#u...x#1 x#0 u -- x#u...x#1 x#0 x#u )
```

Remove *u* copy the *x#u* to the top of the stack. `0 pick` is equivalent to `dup` and `1 pick` is equivalent to `over`.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `unpick`, `roll`, `rot`.

Source file: <src/lib/data_stack.fs>.

pixel-pan-right

```
pixel-pan-right ( -- )
```

Pan the whole screen one pixel to the right. `pixel-pan-right` is a wrapper that calls `(pixel-pan-right` saving the BC register.

See `(pixel-pan-right`, `pixels-pan-right`, `pixel-scroll-up`.

Source file: <src/lib/graphics.scroll.fs>.

pixel-scroll-up

```
pixel-scroll-up ( -- )
```

Scroll the whole screen one pixel up. `pixel-scroll-up` is a wrapper that calls `(pixel-scroll-up` saving the BC register.

See also: `pixel-pan-right`, `pixels-scroll-up`.

Source file: <src/lib/graphics.scroll.fs>.

pixels

```
pixels ( -- u )
```

Return the number *u* of pixels that are set on the screen. `pixels` is a deferred word (see `defer`) set by `fast-pixels` or `slow-pixels`.

See also: `bits`.

Source file: <src/lib/graphics.pixels.fs>.

pixels-pan-right

```
pixels-pan-right ( u -- )
```

Pan the whole screen *u* pixels to the right.

See `pixel-pan-right`, `pixels-scroll-up`.

Source file: <src/lib/graphics.scroll.fs>.

pixels-scroll-up

```
pixels-scroll-up ( u -- )
```

Scroll the whole screen *u* pixels up.

See also: `pixel-scroll-up`, `pixels-pan-right`.

Source file: <src/lib/graphics.scroll.fs>.

place

```
place ( ca1 len1 ca2 -- )
```

Store the string *ca1 len1* as a counted string at *ca2*. The source and destination strings are permitted to overlap.

`place` is written in Z80. Its equivalent definition in Forth is the following:

```
: place ( ca1 len1 ca2 -- ) 2dup c! char+ smove ;
```

See also: `+place`, `smove`.

Source file: <src/kernel.z80s>.

play

```
play ( ca -- )
```

Play a 14-byte sound definition stored at *ca*.

See also: `sound,`, `sound`, `!sound`, `edit-sound`.

Source file: <src/lib/sound.128.fs>.

plot

```
plot ( gx gy -- )
```

Set a pixel, changing its attribute on the screen and the current graphic coordinates. *gx* is 0..255; *gy* is 0..191.

See also: `set-pixel`, `plot176`, `xy>gxy`.

Source file: <src/lib/graphics.pixels.fs>.

plot176

```
plot176 ( gx gy -- ) "plot-176"
```

Set a pixel, changing its attribute on the screen and the current graphic coordinates, using only the top 176 pixel rows of the screen (the lower 16 pixel rows are not used). *gx* is 0..255; *gy* is 0..175.

`plot176` is equivalent to Sinclair BASIC's `PLOT` command.

WARNING If parameters are out of range, the ROM will throw a BASIC error, and the system will crash.

See also: `set-pixel176`, `plot`, `xy>gxy176`.

Source file: <src/lib/graphics.pixels.fs>.

po?

```
po? ( -- op ) "p-o-question"
```

Return the opcode *op* of the Z80 `assembler` instruction `jp op`, to be used as condition and consumed by `?ret,`, `?jp,`, `?call,`, `aif`, `awhile` or `auntil`.

See also: `z?`, `nz?`, `c?`, `nc?`, `pe?`, `p?`, `m?`.

Source file: <src/lib/assembler.fs>.

polarity

```
polarity ( n -- -1|0|1 )
```

If *n* is zero, return zero. If *n* is negative, return negative one. If *n* is positive, return positive one.

`polarity` is written in Z80. These are example implementations in Forth:

```
: polarity ( n -- -1|0|1 ) dup 0= ?exit 0< ?dup ?exit 1 ;

: polarity ( n -- -1|0|1 ) dup 0= ?exit 0< 2* 1+ ;

: polarity ( n -- -1|0|1 ) -1 max 1 min ;
```

See also: `<=>`, `negate`, `within`, `between`.

Source file: <src/lib/math.operators.1-cell.fs>.

pop,

```
pop, ( regp -- ) "pop-comma"
```

Compile the Z80 `assembler` instruction `PUSH regp`.

See also: `pop,`, `ret,`, `sp`.

Source file: <src/lib/assembler.fs>.

positional-case:

```
positional-case: ( "name" -- ) "positional-case-colon"
```

Create a positional case word *name*. At runtime, *name* will execute the n-th word compiled in its definition, depending upon the value on the stack. No range checking.

Usage example:

```
: say0 ( -- ) ." nul" ;
: say1 ( -- ) ." unu" ;
: say2 ( -- ) ." du" ;

positional-case: say ( n -- ) say0 say1 say2 ;

0 say cr 1 say cr 2 say cr
```

Source file: <src/lib/flow.positional-case-colon.fs>.

possibly

```
possibly ( "name" -- )
```

Parse *name*. If *name* is the name of a word in the current search order, execute it; else do nothing.

See also: `exec`, `defined`, `name>`, `execute`, `anew`.

Source file: <src/lib/compilation.fs>.

postpone

```
postpone ( "name" -- )
```

Skip leading space delimiters. Parse *name* delimited by a space. Find *name*. Append the compilation semantics of *name* to the current definition.

`postpone` is an `immediate` word.

Definition:

```
: postpone ( "name" -- )
  defined dup ?defined
  name>immediate? 0= if compile compile then compile, ;
  immediate
```

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `[compile]`, `compile`, `defined`, `?defined`, `name>immediate?`, `compile,`, `0if`.

Source file: <src/kernel.z80s>.

prefix?

```
prefix? ( ca1 len1 ca2 len2 -- f ) "prefix-question"
```

Is string *ca2 len2* the prefix of string *ca1 len1*?

See also: `suffix?`, `-prefix`.

Source file: <src/lib/strings.MISC.fs>.

pressed

```
pressed ( -- false | b a true )
```

Return the key identifier *b a* (key bitmask and keyboard row port) of the first key from table `kk-ports` that happens to be pressed, and `true`; if no key is pressed, return `false`.

See also: `only-one-pressed`, `pressed?`.

Source file: <src/lib/keyboard.MISC.fs>.

pressed?

```
pressed? ( b a -- f ) "pressed-question"
```

Is a keyboard key *b a* pressed? *b* is the key bitmask and *a* is the keyboard row port.

See also: `pressed`, `only-one-pressed`.

Source file: <src/lib/keyboard.MISC.fs>.

previous

```
previous ( -- )
```

Remove the top word list (the word list that is searched first) from the search order.

Definition:

```
: previous ( -- ) get-order nip 1- set-order ;
```

Origin: Forth-94 (SEARCH EXT), Forth-2012 (SEARCH EXT).

See also: `>order`, `get-order`, `set-order`.

Source file: <src/kernel.z80s>.

previous-mode

```
previous-mode ( -- a )
```

A `variable`. *a* is the address of a cell containing the execution token of the word that activates the screen mode that was active before executing `bye` (e.g. `mode-32`, `mode-32iso`, `mode-64ao`). `previous-mode` is updated by `bye`, and used by `warm` to restore the `current-mode`.

Source file: <src/kernel.z80s>.

printer

```
printer ( -- )
```

Select the printer as output.

See also: `terminal`, `printing`.

Source file: <src/lib/display.control.fs>.

printing

```
printing ( -- a )
```

A `variable`. *a* is the address of a cell containing the printer flag. `printing` is set by `printer`, reset by `terminal` and checked by `page`. `printing` should not be changed directly by the program.

Source file: <src/kernel.z80s>.

private

```
private ( wid0 wid1 -- wid0 wid1 )
```

Mark subsequent definitions invisible outside the current package. This is the default condition following the usage of `package`.

wid1 is the word list of the current package; *wid0* is the word list in which the current package was created.

Origin: SwiftForth.

See also: `end-package`, `public`.

Source file: <src/lib/modules.package.fs>.

private{

```
private{ ( -- ) "private-curly-bracket"
```

Start private definitions. See `privatize` for a usage example.

Source file: <src/lib/modules.privatize.fs>.

privatize

```
privatize ( -- )
```

Hide all words defined between the latest valid pair of `private{` and `}private`.

Usage example:

```
private{

\ Everything between ``private{`` and ``}private``
\ will become private.

: foo ;
: moo ;

}private

: goo   foo moo ;  \ can use ``foo`` and ``moo``
privatize          \ hide ``foo`` and ``moo``
' foo              \ will fail
```

See also: `internal`, `isolate`, `module`, `package`, `seclusion`.

Source file: <src/lib/modules.privatize.fs>.

prt,

```
prt, ( -- ) "p-r-t-comma"
```

Compile the Z80 `assembler` instruction `rst $16`. Therefore `prt,` is equivalent to `$16 rst,`.

See also: `rst,`, `hook,`.

Source file: <src/lib/assembler.fs>.

public

```
public ( wid0 wid1 -- wid0 wid1 )
```

Mark subsequent definitions available outside the current package defined with `package`.

wid1 is the word list of the current package; *wid0* is the word list in which the current package was created.

Origin: SwiftForth.

See also: `end-package`, `private`.

Source file: <src/lib/modules.package.fs>.

push,

```
push, ( regp -- ) "push-comma"
```

Compile the Z80 `assembler` instruction `PUSH regp`.

See also: `push,`, `ret,`, `sp`.

Source file: <src/lib/assembler.fs>.

pusha

```
pusha ( -- a ) "push-a"
```

A `constant`. *a* is the address of a secondary entry point of the Forth inner interpreter. The code at *a* pushes the A register onto the stack and then continues at the address returned by `next`.

`pusha` is useful for exiting from a `code` word using an absolute conditional jump, or to save the bytes needed to prepare an 8-bit register to be pushed on the stack.

See also: `pushhl`, `pushhlde`.

Source file: <src/kernel.z80s>.

pushhl

```
pushhl ( -- a ) "push-h-l"
```

A `constant`. *a* is the address of a secondary entry point of the Forth inner interpreter. The code at *a* pushes the HL register onto the stack and then continues at the address returned by `next`.

`pushhl` is useful for exiting from a `code` word using an absolute conditional jump.

See also: `pusha`, `pushhlde`.

Source file: <src/kernel.z80s>.

pushhlde

```
pushhlde ( -- a ) "push-h-l-d-e"
```

a is the address of a secondary entry point of the Forth inner interpreter. The code at *a* pushes registers DE and HL onto the stack and then continues at the address returned by `next`.

NOTE	DE is pushed first, so HL is left on top of the stack. This is equivalent to pushing the double number formed by both registers, being HL the high part and DE the lower part.

`pushhlde` is useful for exiting from a `code` word using an absolute conditional jump.

See also: `pusha`, `pushhl`.

Source file: <src/lib/assembler.MISC.fs>.

px

```
px ( -- ) "p-x"
```

Give previous quick index, calculated from `scr`.

See also: `qx`, `nx`.

Source file: <src/lib/tool.list.blocks.fs>.

q

query

```
query ( -- )
```

Make the user input device the input source. Receive input into the terminal input buffer, replacing any previous contents. Make the result, whose address is returned by `tib`, the input buffer. Set `>in` to zero.

The function of `query` may be performed with `accept` and `evaluate`.

Definition:

```
: query ( -- )
  tib /tib 2dup blank accept #tib ! space terminal>source ;
```

Origin: fig-forth, Forth-79 (Required Word Set), Forth-83 (Controlled Reference Words), Forth-94 (CORE EXT, obsolescent).

Source file: <src/kernel.z80s>.

quit

```
quit ( -- )
```

Empty the return stack, make the terminal the current source and enter interpretation `state`. Then repeat the following:

- Accept a line from the input source into the input buffer, set `>in` to zero and `interpret`.
- Display the system prompt, if in interpretation `state`.

Definition:

```
: quit ( -- )
  rp0 @ rp! postpone [
  begin
    cr query interpret
    interpreting? if ok then
  again ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `[`, `query`, `interpret`, `ok`, `abort`.

Source file: <src/kernel.z80s>.

qx

```
qx ( -- ) "q-x"
```

Give a quick index. The number and width of the columns depend on the current screen mode. The current block, stored in `scr`, is highlighted.

Origin: Gforth's `blocked` editor.

See also: `nx`, `px`.

Source file: <src/lib/tool.list.blocks.fs>.

qx-bounds

```
qx-bounds ( -- u1 u2 ) "q-x-bounds"
```

Blocks to be included in the quick index, from block *u2* to block *u1-1*. They depend on `scr`.

See also: `qx`.

Source file: <src/lib/tool.list.blocks.fs>.

qx-columns

```
qx-columns ( -- n ) "q-x-columns"
```

n is the number of columns (2..4) of the quick index. It depends on the columns (32, 42, 64...) of the current screen mode.

See also: `qx`, `/qx-column`.

Source file: <src/lib/tool.list.blocks.fs>.

r

r

```
r ( "ccc<eol>" -- )
```

A command of `gforth-editor`: `replace` marked area with *ccc*.

See also: `d`, `m`, `a`, `d`, `f`, `h`, `i`.

Source file: <src/lib/prog.editor.gforth.fs>.

r

```
r ( n -- )
```

A command of `specforth-editor`: Replace line *n* with the text in `pad`.

See also: `b`, `c`, `d`, `e`, `f`, `h`, `i`, `l`, `m`, `n`, `p`, `s`, `t`, `x`, `-move`.

Source file: <src/lib/prog.editor.specforth.fs>.

r#

```
r# ( -- a ) "r-slash"
```

A `variable`. *a* is the address of a cell containing the location of the editing cursor, an offset from the top of the current block. Its default value is zero.

`r#` is used by `specforth-editor` and `gforth-editor`.

Origin: fig-Forth's user variable `r#`.

See also: `top`.

Source file: <src/lib/prog.editor.COMMON.fs>.

r'@

```
r'@ ( -- x1 ) ( R: x1 x2 -- x1 x2 ) "r-tick-fetch"
```

Fetch *x1* from the return stack.

See also: `r@`.

Source file: <src/lib/return_stack.fs>.

r/o

```
r/o ( -- fam ) "r-o"
```

Return the "read only" file access method *fam*.

See also: `w/o`, `r/w`, `bin`.

Origin: Forth-94 (FILE), Forth-2012 (FILE).

Source file: <src/lib/dos.gplusdos.fs>.

r/w

```
r/w ( -- fam ) "r-w"
```

Return the "read/write" file access method *fam*.

WARNING `r/w` is not supported on G+DOS.

See also: `r/o`, `w/o`, `bin`.

Origin: Forth-94 (FILE), Forth-2012 (FILE).

Source file: <src/lib/dos.gplusdos.fs>.

r>

```
r> ( -- x ) ( R: x -- ) "r-from"
```

Move *x* from the return stack to the data stack.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `>r`, `r@`, `2r>`.

Source file: <src/kernel.z80s>.

r>xy

```
r>xy ( -- ) ( R: col row -- ) "r-to-x-y"
```

Restore the current cursor coordinates from the return stack.

See also: `xy>r`, `restore-mode`.

Source file: <src/lib/display.cursor.fs>.

r@

```
r@ ( -- x ) ( R: x -- x ) "r-fetch"
```

Copy *x* from the return stack to the data stack.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `>r`, `r>`, `rdrop`, `r'@`.

Source file: <src/kernel.z80s>.

ragain

```
ragain ( dest cs-id -- ) "r-again"
```

Compile a Z80 `assembler` unconditional relative-jump instruction to address *dest*, as part of a relative-address control-flow structure `rbegin` .. `ragain`, identified by *cs-id*.

See also: `aagain`, `(runtil`.

Source file: <src/lib/assembler.fs>.

rahead

```
rahead ( -- orig ) "r-ahead"
```

Compile a Z80 `assembler` forward relative jump. Leave its unresolved address *orig*, to be resolved by `>rresolve`.

Source file: <src/lib/assembler.fs>.

ram

```
ram ( -- n )
```

A `constant`. *n* is the total RAM size in kibibytes.

NOTE On G+DOS, the RAM size includes the 8 KiB of the Plus D interface.

See also: `banks`.

Source file: <src/kernel.z80s>.

random

```
random ( n1 -- n2 )
```

Return a random number *n2* from 0 to *n1* minus 1.

See also: `rnd`, `random-within`, `fast-random`.

Source file: <src/lib/random.fs>.

random-between

```
random-between ( n1 n2 -- n3 )
```

Return a random number *n3* from *n1* (min) to *n2* (max).

See also: `random-within`, `random`, `between`.

Source file: <src/lib/random.fs>.

random-within

```
random-within ( n1 n2 -- n3 )
```

Return a random number *n3* from *n1* (min) to *n2-1* (max).

See also: `random-between`, `random`, `within`.

Source file: <src/lib/random.fs>.

randomize

```
randomize ( n -- )
```

Set the seed used by `fast-rnd` and `fast-random` to *n*.

See also: `randomize0`.

Source file: <src/lib/random.fs>.

randomize0

```
randomize0 ( -- ) "randomize-zero"
```

Set the seed used by `fast-rnd` and `fast-random` to *n*; if *n* is zero use the system frames counter instead.

See also: `randomize`.

Source file: <src/lib/random.fs>.

rbegin

```
rbegin ( -- dest cs-id ) "r-begin"
```

Mark the start of an `assembler` sequence for repetitive execution, leaving *dest* to be resolved by the corresponding `runtil`, `ragain` or `rrepeat`. Also, leave the control-flow structure identifier_cs-id_ to be checked by the corresponding same word.

`rbegin` is part of the `assembler` relative-address control-flow structures `rbegin` .. `ragain`, `rbegin` .. `runtil` and `rbegin` .. `rwhile` .. `rrepeat`.

See also: `abegin`.

Source file: <src/lib/assembler.fs>.

rbuf

```
rbuf ( -- ca )
```

Return the address *ca* of the 100-byte replace buffer used by the `gforth-editor`.

See also: `ibuf`, `fbuf`, `r`.

Source file: <src/lib/prog.editor.gforth.fs>.

rdepth

```
rdepth ( -- +n ) "r-depth"
```

+*n* is the number of single-cell values contained in the return stack.

See also: `rp0`, `rp`, `depth`, `fdepth`.

Source file: <src/lib/return_stack.fs>.

rdraw176

```
rdraw176 ( gx gy -- ) "r-draw-176"
```

Draw a line relative *gx gy* to the current coordinates, using only the top 176 pixel rows of the screen (the lower 16 pixel rows are not used). *gx* is 0..255; *gy* is 0..175.

`rdraw176` is equivalent to Sinclair BASIC’s `DRAW` command.

See also: `adraw176`.

Source file: <src/lib/graphics.lines.fs>.

rdrop

```
rdrop ( R: x -- ) "r-drop"
```

Remove *x* from the return stack.

See also: `r@`, `drop`.

Origin: Comus.

Source file: <src/kernel.z80s>.

read-block

```
read-block ( u -- )
```

Read disk block *u* to the buffer.

Definition:

```
: read-block ( u -- ) read-mode transfer-block ;
```

See also: `read-mode`, `transfer-block`, `write-block`, `block`.

Source file: <src/kernel.z80s>.

read-mode

```
read-mode ( -- )
```

Set the read mode for `transfer-sector` and `transfer-block`.

See also: `write-mode`.

Source file: <src/kernel.gplusdos.z80s>.

realias

```
realias ( xt "name" -- )
```

Set the alias *name* to execute *xt*.

See `alias`, `alias!`.

Source file: <src/lib/define.alias.fs>.

recurse

```
recurse ( -- )
```

Append the execution semantics of the current definition to the current definition.

`recurse` is an `immediate` and `compile-only` word.

Origin: Forth-83 (Controlled Reference Words), Forth-94 (CORE), Forth-2012 (CORE).

Source file: <src/lib/flow.MISC.fs>.

red

```
red ( -- b )
```

A `cconstant` that returns 2, the value that represents the red color.

See also: `black`, `blue`, `magenta`, `green`, `cyan`, `yellow`, `white`, `contrast`, `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

refill

```
refill ( -- f )
```

Definition:

```
: refill ( -- f )
  loading? if  blk @ 1+ dup block>source block? exit  then
  false source-id ?exit 0= query ;
```

Origin: Forth-94 (CORE EXT, BLOCK EXT); Forth-2012 (CORE EXT, BLOCK EXT).

Source file: <src/kernel.z80s>.

reload

```
reload ( -- )
```

Load the most recently loaded block.

See also: `load`, `lastblk`.

Source file: <src/lib/blocks.fs>.

relse

```
relse ( orig1 cs-id -- orig2 cs-id ) "r-else"
```

Check the Z80 `assembler` control-flow structure identifier *cs_id*, and resolve the forward reference *orig1*, both left by `rif`; then compile a Z80 `assembler` unconditional relative-address jump, putting its unresolved forward reference *orig2* and control-flow structure identifier *cs-id* on the stack, to be resolved by `rthen`.

`relse` is part of the `assembler` relative-address control-flow structure `rif` .. `relse` .. `rthen`.

See also: `aelse`, `?pairs`, `(rif`.

Source file: <src/lib/assembler.fs>.

rename-file

```
rename-file ( ca1 len1 ca2 len2 -- ior )
```

Rename the file named by the character string *ca1 len1* to the name in the character string *ca2 len2* and return I/O result code *ior*.

Origin: Forth-94 (FILE EXT), Forth-2012 (FILE EXT).

Source file: <src/lib/dos.gplusdos.fs>.

reneed

```
reneed ( "name" -- )
```

Load the first block whose header contains *name* (surrounded by spaces).

`reneed` is a deferred word (see `defer`) whose default action is `locate-reneed`.

See also: `make-thru-index`.

Source file: <src/lib/002.need.fs>.

reneeded

```
reneeded ( ca len -- )
```

Load the first block whose header contains the string *ca len* (surrounded by spaces). If not found, `throw` an exception #-268 ("needed, but not located").

`reneeded` is a deferred word (see `defer`) whose default action is `locate-reneeded`.

See also: `make-thru-index`.

Source file: <src/lib/002.need.fs>.

repeat

```
repeat
  Compilation: ( C: orig dest -- )
  Run-time:    ( -- )
```

Compilation: Compile an unconditional `branch` to the backward reference *dest*, usually left by `begin`. Resolve the forward reference *orig*, usually left by `while`.

Run-time: Continue execution at the location specified by *dest*.

`repeat` is an `immediate` and `compile-only` word.

Definition:

```
: repeat \ Compilation: ( C: orig dest -- )
         \ Run-time:    ( -- )
  postpone again postpone then ; immediate compile-only
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `again`, `then`, `until`.

Source file: <src/kernel.z80s>.

replace

```
replace ( ca1 len1 ca2 len2 -- )
```

Replace the contents of zone *ca2 len2* with string *ca1 len1*. If *len1* is greater than *len2*, only *len2* bytes are replaced.

See also: `insert`, `delete`, `replaces`.

Source file: <src/lib/strings.MISC.fs>.

replaces

```
replaces ( ca1 len1 ca2 len2 -- )
```

Set the string *ca1 len1* as the text to substitute for the substitution named by *ca2 len2*. If the substitution does not exist it is created. The program may then reuse the buffer *ca1 len1* without affecting the definition of the substitution.

The name of a substitution should not contain the "%" delimiter character.

`replaces` allots data space and creates a definition.

Origin: Forth-2012 (STRING EXT).

See also: `substitute`, `unescape`, `substitution`, `find-substitution`, `substitute-wordlist`, `replace`.

Source file: <src/lib/strings.replaces.fs>.

res,

```
res, ( reg b -- ) "res-comma"
```

Compile the Z80 `assembler` instruction `RES b,reg`.

See also: `bit,`, `set,`, `sub#,`.

Source file: <src/lib/assembler.fs>.

reserve

```
reserve ( n -- a )
```

Reserve *n* bytes of data space, erase the zone and return its address *a*.

See also: `buffer:`, `allot`, `allotted`, `here`, `erase`.

Source file: <src/lib/memory.MISC.fs>.

reset-bit

```
reset-bit ( b1 n -- b2 )
```

Reset bit *n* of *b1*, returning the result *b2*.

See also: `bit?`, `set-bit`, `bit>mask`.

Source file: <src/lib/memory.MISC.fs>.

reset-default-mode

```
reset-default-mode ( -- )
```

Set `default-mode` to its default action `noop`. `reset-default-mode` is executed by `cold`.

Source file: <src/kernel.z80s>.

reset-dticks

```
reset-dticks ( -- ) "reset-d-ticks"
```

Reset the system clock to zero ticks.

See also: `reset-ticks`, `dticks`, `set-dticks`, `ticks/second`, `bench{`.

Source file: <src/lib/time.fs>.

reset-pixel

```
reset-pixel ( gx gy -- )
```

Reset a pixel without changing its attribute on the screen or the current graphic coordinates. *gx* is 0..255; *gy* is 0..191.

See also: `set-pixel`, `toggle-pixel`, `reset-pixel176`.

Source file: <src/lib/graphics.pixels.fs>.

reset-pixel176

```
reset-pixel176 ( gx gy -- ) "reset-pixel-176"
```

Reset a pixel without its attribute on the screen or the current graphic coordinates, and using only the top 176 pixel rows of the screen (the lower 16 pixel rows are not used). *gx* is 0..255; *gy* is 0..175.

See also: `set-pixel176`, `toggle-pixel176`, `reset-pixel`, `set-pixel`, `toggle-pixel`, `plot`, `plot176`.

Source file: <src/lib/graphics.pixels.fs>.

reset-ticks

```
reset-ticks ( -- )
```

Reset the low 16 bits of the OS clock to zero ticks.

See also: `ticks`, `set-dticks`, `ticks/second`, `bench{`.

Source file: <src/lib/time.fs>.

reset-time

```
reset-time ( -- )
```

Reset the current time to 00:00:00.

See also: `get-time`.

Source file: <src/lib/time.fs>.

resize

```
resize ( a1 -- a2 ior )
```

Change the allocation of the contiguous data space starting at the address *a1*, previously allocated by `allocate` or `resize`, to *u* bytes. *u* may be either larger or smaller than the current size of the region. The data-space pointer is unaffected by this operation.

If the operation succeeds, *a2* is the starting address of *u* bytes of allocated memory and *ior* is zero. *a2* may be, but need not be, the same as *a1*. If they are not the same, the values contained in the region at *a1* are copied to *a2*, up to the minimum size of either of the two regions. If they are the same, the values contained in the region are preserved to the minimum of *u* or the original size. If *a2* is not the same as *a1*, the region of memory at *a1* is returned to the system according to the operation of `free`.

If the operation fails, *a2* equals *a1*, the region of memory at *a1* is unaffected, and ior is the I/O result code.

`resize` is a deferred word (see `defer`) whose action can be `charlton-resize`, depending on the `heap` implementation used by the application.

Origin: Forth-94 (MEMORY), Forth-2012 (MEMORY).

See also: `allocate`, `free`, `empty-heap`.

Source file: <src/lib/memory.allocate.COMMON.fs>.

resolve-al#

```
resolve-al# ( orig b -- ) "resolve-a-l-number-sign"
```

Resolve an absolute reference at *orig* to label *b*.

See also: `resolve-rl#`, `(resolve-ref`, `>l`.

Source file: <src/lib/assembler.labels.fs>.

resolve-refs

```
resolve-refs ( n -- )
```

Resolve all references to `assembler` label *n*, which was defined by `l:`.

`resolve-refs` is a factor of `l!`.

Source file: <src/lib/assembler.labels.fs>.

resolve-rl#

```
resolve-rl# ( orig b -- ) "resolve-r-l-number-sign"
```

Resolve a relative reference at *orig* to `assembler` label *b*.

See also: `resolve-al#`, `(resolve-ref`, `>l`.

Source file: <src/lib/assembler.labels.fs>.

restore-mode

```
restore-mode ( -- )
```

Restore the screen mode that was saved in `previous-mode` by `save-mode`.

`restore-mode` is executed by `warm`.

Definition:

```
: restore-mode ( -- ) previous-mode perform ;
```

See also: `current-mode`, `perform`.

Source file: <src/kernel.z80s>.

results

```
results ( +n -- )
```

Define the number +*n* of local variables to leave on the stack as results. Used with locals created by `arguments`.

`results` is a `compile-only` word.

Source file: <src/lib/locals.arguments.fs>.

resx,

```
resx, ( disp regpi b --  ) "res-x-comma"
```

Compile the Z80 `assembler` instruction `RES b,(regpi+disp)`.

See also: `bitx,`, `setx,`, `subx,`, `sbcx,`, `andx,`, `xorx,`, `orx,`, `decx,`.

Source file: <src/lib/assembler.fs>.

ret,

```
ret, ( -- ) "ret-comma"
```

Compile the Z80 `assembler` instruction `RET`.

See also: `?ret,`, `call,`, `pop,`.

Source file: <src/lib/assembler.fs>.

retry

```
retry ( -- )
```

Do an unconditional branch to the start of the word.

`retry` is an `immediate` and `compile-only` word.

See also: `?retry`.

Source file: <src/lib/flow.MISC.fs>.

return-stack-cells

```
return-stack-cells ( -- n )
```

n is the maximum size of the return stack, in cells.

See also: `stack-cells`, `environment?`.

Source file: <src/lib/environment-question.fs>.

reveal

```
reveal ( -- )
```

Reveal the `latest` definition by resetting its `smudge` bit.

Definition:

```
: reveal ( -- ) latest revealed ;
```

See also: `revealed`, `hide`.

Source file: <src/kernel.z80s>.

revealed

```
revealed ( nt -- )
```

Reveal the definition *nt* by resetting its `smudge` bit.

See also: `reveal`, `hidden`.

Source file: <src/kernel.z80s>.

rif

```
rif ( op -- orig cs-id ) "r-if"
```

Compile a Z80 `assembler` conditional relative-jump instruction *op*, which was put on the stack by `z?`, `nz?`, `c?` or `nc?`. Return the address *orig* to be resolved by `relse` or `rthen` and the control-structure identifier *cs-id*.

`rif` is part of the `assembler` relative-address control-flow structure `rif` .. `relse` .. `rthen`.

See also: `aif`, `rbegin`, `jp>jr`, `inverse-cond`.

Source file: <src/lib/assembler.fs>.

rl#

```
rl# ( n -- a ) "r-l-number-sign"
```

Create a relative reference to `assembler` label number *n*, defined by `l:`. If label *n* is already defined, *a* is its value. Otherwise *a* is a temporary address to be consumed by the relative jump instruction, and the actual address will be resolved when the label is defined by `l:`.

Usage example:

```
code my-code ( -- )
  #2 rl# jr, \ a relative jump to label #2
  nop,
  #2 l:      \ definition of label #2
  ret,
end-code
```

WARNING `rl#` is used before the Z80 command, while its counterpart `al#` is used after the Z80 command.

Source file: <src/lib/assembler.labels.fs>.

rl,

```
rl, ( reg -- ) "r-l-comma"
```

Compile the Z80 `assembler` instruction `RL reg`.

See also: `rr,`, `rla,`, `rlc,`, `rlca,`.

Source file: <src/lib/assembler.fs>.

rl-id

```
rl-id ( -- b ) "r-l-i-d"
```

b is the identifier of relative references created by `rl#`. `rl-id` is used as a bitmask added to the `assembler` label number stored in `l-refs`.

See also: `al-id`.

Source file: <src/lib/assembler.labels.fs>.

rla,

```
rla, ( -- ) "r-l-a-comma"
```

Compile the Z80 `assembler` instruction `RLA`.

See also: `rra,`, `rl,`, `rlc,`, `rlca,`, `rld,`.

Source file: <src/lib/assembler.fs>.

rlc,

```
rlc, ( reg -- ) "r-l-c-comma"
```

Compile the Z80 `assembler` instruction `RLC reg`.

See also: `rrc,`, `rlca,`, `rl,`, `rla,`.

Source file: <src/lib/assembler.fs>.

rlca,

```
rlca, ( -- ) "r-l-c-a-comma"
```

Compile the Z80 `assembler` instruction `RLCA`.

See also: `rrca,`, `rlc,`, `rl,`, `rla,`, `rld,`.

Source file: <src/lib/assembler.fs>.

rlcx,

```
rlcx, ( disp regpi --  ) "r-l-c-x-comma"
```

Compile the Z80 `assembler` instruction `RLC (regpi+disp)`.

See also: `rrcx,`, `rlx,`, `rrx,`, `slax,`, `srax,`, `sllx,`, `srlx,`, `bitx,`, `resx,`, `setx,`.

Source file: <src/lib/assembler.fs>.

rld,

```
rld, ( -- ) "r-l-d-comma"
```

Compile the Z80 `assembler` instruction `RLD`.

See also: `rla,`, `rlca,`, `rra,`.

Source file: <src/lib/assembler.fs>.

rlx,

```
rlx, ( disp regpi --  ) "r-l-x-comma"
```

Compile the Z80 `assembler` instruction `RL (regpi+disp)`.

See also: `rlcx,`, `rrcx,`, `rrx,`, `slax,`, `srax,`, `sllx,`, `srlx,`, `bitx,`, `resx,`, `setx,`.

Source file: <src/lib/assembler.fs>.

rnd

```
rnd ( -- x ) "r-n-d"
```

Generate a single-cell random number *x*.

See also: `random`, `random-within`, `fast-rnd`.

Source file: <src/lib/random.fs>.

roll

```
roll ( x#u x#u-1...x#0 u -- x#u-1...x#0 x#u )
```

See also: `pick`, `rot`.

Source file: <src/lib/data_stack.fs>.

rom-font

```
rom-font ( -- a )
```

A `constant`. *a* is the address of the ROM font, which is 15360 ($3C00), the bitmap address of character 0, 256 bytes below the bitmap of the space (character 32), which is the first printable character. *a* is the default value of `os-chars`.

See also: `default-font`, `set-font`, `get-font`, `outlet-autochars`.

Source file: <src/lib/display.fonts.fs>.

root

```
root ( -- )
```

Transform the search order consisting of *wid#n .. wid#2 wid#1* (where *wid#1* is searched first) into *wid#n .. wid#2 wid#r*, where *wid#r* is the word-list identifier returned by `root-wordlist`. I.e., replace the top word list of the search order with `root-wordlist`.

`root` is the `vocabulary` corresponding to `root-wordlist`.

See also: `forth`, `wordlist`.

Source file: <src/kernel.z80s>.

root-wordlist

```
root-wordlist ( -- wid )
```

Return *wid*, the identifier of the word list that includes the words defined in the minimum search order. The words defined in the word list identified by `root-wordlist` are aliases of the definitions in `forth-wordlist`.

See also: `only`, `wordlist`, `set-order`, `assembler-wordlist`, `alias`.

Source file: <src/kernel.z80s>.

rot

```
rot ( x1 x2 x3 -- x2 x3 x1 )
```

Rotate the top three stack entries.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `-rot`, `over`, `tuck`, `swap`, `roll`, `pick`.

Source file: <src/kernel.z80s>.

row

```
row ( -- row )
```

Current row (y coordinate).

See also: `column`, `last-row`, `rows`.

Source file: <src/lib/display.cursor.fs>.

rows

```
rows ( -- n )
```

Return the number of rows in the current screen mode. The default value is 24.

See also: `columns`, `last-row`, `row`.

Source file: <src/lib/display.mode.COMMON.fs>.

rp

```
rp ( -- a ) "r-p"
```

A `constant`. *a* is the address of the return stack pointer.

See also: `rp@`, `rp!`.

Source file: <src/kernel.z80s>.

rp!

```
rp! ( a -- ) "r-p-store"
```

Store *a* into the return stack pointer.

`rp!` is written in Z80. Its equivalent definition in Forth is the following:

```
: rp! ( a -- ) rp ! ;
```

See also: `rp`, `rp@`.

Source file: <src/kernel.z80s>.

rp0

```
rp0 ( -- a ) "r-p-zero"
```

A `user` variable. *a* is the address of a cell containing the address of the bottom of the return stack.

Origin: fig-Forth's `r0`.

Source file: <src/kernel.z80s>.

rp@

```
rp@ ( -- a ) "r-p-fetch"
```

Fetch the content of the return stack pointer.

`rp@` is written in Z80. Its equivalent definition in Forth is the following:

```
: rp@ ( -- a ) rp @ ;
```

See also: `rp`, `rp!`.

Source file: <src/kernel.z80s>.

rr,

```
rr, ( reg -- ) "r-r-comma"
```

Compile the Z80 `assembler` instruction `RR reg`.

See also: `rl,`, `rra,`, `rrc,`, `rrca,`.

Source file: <src/lib/assembler.fs>.

rra,

```
rra, ( -- ) "r-r-a-comma"
```

Compile the Z80 `assembler` instruction `RRA`.

See also: `rla,`, `rr,` `rrc,`, `rrca,`.

Source file: <src/lib/assembler.fs>.

rrc,

```
rrc, ( reg -- ) "r-r-c-comma"
```

Compile the Z80 `assembler` instruction `RRC reg`.

See also: `rlc,`, `rr,`, `rra,`, `rrca,`.

Source file: <src/lib/assembler.fs>.

rrca,

```
rrca, ( -- ) "r-r-c-a-comma"
```

Compile the Z80 `assembler` instruction `RRCA`.

See also: `rlca,`, `rrc,`, `rr,`, `rra,`.

Source file: <src/lib/assembler.fs>.

rrcx,

```
rrcx, ( disp regpi --  ) "r-r-c-x-comma"
```

Compile the Z80 `assembler` instruction `RRC (regpi+disp)`.

See also: `rlcx,`, `rlx,`, `rrx,`, `slax,`, `srax,`, `sllx,`, `srlx,`, `bitx,`, `resx,`, `setx,`.

Source file: <src/lib/assembler.fs>.

rrepeat

```
rrepeat ( dest cs-id1 orig cs-id2 --) "r-repeat"
```

Compile a Z80 `assembler` unconditional relative-jump instruction to address *dest*, left by `rbegin`, and check its control-flow identifier *cs-id1*. Resolve the forward reference *orig*, usually left by `rwhile`, and check its control-flow structure *cs-id2*.

`rrepeat` is part of the `assembler` relative-address control-flow structure `rbegin` .. `rwhile` .. `rrepeat`.

See also: `arepeat`, `ragain`.

Source file: <src/lib/assembler.fs>.

rresolve

```
rresolve ( orig dest -- ) "r-resolve"
```

Resolve a Z80 `assembler` relative branch.

See also: `<rresolve`, `>rresolve`, `?rel`.

Source file: <src/lib/assembler.fs>.

rrx,

```
rrx, ( disp regpi --  ) "r-r-x-comma"
```

Compile the Z80 `assembler` instruction `RR (regpi+disp)`.

See also: `rlcx,`, `rrcx,`, `rlx,`, `slax,`, `srax,`, `sllx,`, `srlx,`, `bitx,`, `resx,`, `setx,`.

Source file: <src/lib/assembler.fs>.

rshift

```
rshift ( x1 u -- x2 ) "r-shift"
```

Perform a logical right shift of *u* bit-places on *x1*, giving *x2*. Put zeroes into the most significant bits vacated by the shift.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `lshift`, `?shift`.

Source file: <src/lib/math.operators.1-cell.fs>.

rst,

```
rst, ( b -- ) "r-s-t-comma"
```

Compile the Z80 `assembler` instruction `RST b`.

Source file: <src/lib/assembler.fs>.

rstep

```
rstep ( dest cs-id -- ) "r-step"
```

`rstep` is part of the `assembler` relative-address control-flow structure `rbegin` .. `rstep`.

See also: `(runtil`.

Source file: <src/lib/assembler.fs>.

rthen

```
rthen ( orig cs-id -- ) "r-then"
```

Check the control-flow structure identifier *cs-id*. Then resolve the address *orig* left by `rif` or `relse`

`rthen` is part of the `assembler` relative-address control-flow structure `rif` .. `relse` .. `rthen`.

See also: `athen`, `>rresolve`.

Source file: <src/lib/assembler.fs>.

ruler

```
ruler ( c len -- ca len )
```

Return a string *ca len* of characters *c*, in the `stringer`.

See also: `chars>string`, `char>string`, `s+`.

Source file: <src/lib/strings.MISC.fs>.

run:

```
run: ( a n "ccc<semicolon>" -- a ) "run-colon"
```

Add a clause to a `[switch` structure whose head is *a*. The key value of the clause is *n* and its associated behavior is one or more previously defined words, ending with `;`.

Origin: SwiftForth.

See also: `switch]`.

Source file: <src/lib/flow.bracket-switch.fs>.

runs

```
runs ( a n "name" -- )
```

Add a clause to a `[switch` structure whose head is *a*. The key value of the clause is *n* and its associated behavior is the previously defined *name*.

Origin: SwiftForth.

See also: `[switch`, `switch]`.

Source file: <src/lib/flow.bracket-switch.fs>.

runtil

```
runtil ( dest cs-id op -- ) "r-until"
```

Compile a Z80 `assembler` conditional relative-jump instruction *op* to address *dest*, as part of a relative-address control-flow structure `rbegin` .. `runtil`, identified by *cs-id*.

See also: `auntil`, `(runtil`, `jp>jr`, `inverse-cond`.

Source file: <src/lib/assembler.fs>.

rwhile

```
rwhile ( op -- orig cs-id ) "r-while"
```

Compile a Z80 `assembler` relative-jump instruction *op*, which was put on the stack by `z?`, `nz?`, `c?` or `nc?`. Put the location of a forward reference *orig* onto the stack, to be resolved by `rrepeat`, and the control-structure identifier *cs-id*.

`rwhile` is part of the `assembler` relative-address control-flow structures `rbegin` .. `rwhile` .. `rrepeat`.

See also: `awhile`.

Source file: <src/lib/assembler.fs>.

S

s

```
s ( u "ccc<eol>" | u -- )
```

A command of `gforth-editor`: Search for *ccc* until screen *u*. If *ccc* is empty, use the string of the previous search.

See also: `f`, `c`, `a`, `g`, `n`, `p`, `t`.

Source file: <src/lib/prog.editor.gforth.fs>. ==== s

```
s ( n -- )
```

A command of `specforth-editor`: Spread at line *n*. Line *n* and following lines are are moved down one line. Line *n* becomes blank. Line 15 is lost.

See also: `b`, `c`, `d`, `e`, `f`, `h`, `i`, `l`, `m`, `n`, `p`, `r`, `t`, `x`.

Source file: <src/lib/prog.editor.specforth.fs>.

s"

```
s" "s-quote"
  Compilation:    ( "ccc<quote>" -- )
  Interpretation: ( "ccc<quote>" -- ca len )
  Run-time:       ( -- ca len )
```

Parse *ccc* delimited by a double quote. If interpreting, copy the parsed string to the `stringer` and return it as *ca len*. If compiling, compile the parsed string and return it at run-time as *ca len*.

`s"` is an `immediate` word.

Definition:

```
: s" \ Compilation:    ( "ccc<quote>" -- )
     \ Interpretation: ( "ccc<quote>" -- ca len )
     \ Run-time:       ( -- ca len )
  '"' parse-string ; immediate
```

Origin: Forth-94 (CORE, FILE), Forth-2012 (CORE, FILE).

See also: `parse-string`, `s\"`, `s'`, `s""`, `."`, `,"`.

Source file: <src/kernel.z80s>.

s""

```
s"" ( -- ca len ) "s-quote-quote"
```

Return an empty string in the `stringer`.

See also: `s"`, `s\"`, `s'`.

Source file: <src/lib/strings.MISC.fs>.

s'

```
s' "s-tick"
  Compilation: ( "ccc<char>" -- )
  Run-time:    ( -- ca len )
```

Identical to the standard word `s"`, but using single quote as delimiter. A simple alternative to `s\"` when only double quotes are needed in a string.

`s'` is an `immediate` word.

Source file: <src/lib/strings.MISC.fs>.

s+

```
s+ ( ca1 len1 ca2 len2 -- ca3 len3 ) "s-plus"
```

Append the string *ca2 len2* to the end of string *ca1 len1* returning the string *ca3 len3* in the `stringer`.

See also: `/string`, `string/`, `lengths`.

Source file: <src/lib/strings.MISC.fs>.

s,

```
s, ( ca len -- ) "s-comma"
```

Compile the string *ca len*.

Definition:

```
: s, ( ca len -- ) tuck here place char+ allot ;
```

See also: `c,`, `here`, `cmove`, `allot`, `count`, `fars,`.

Source file: <src/kernel.z80s>.

s>d

```
s>d ( n -- d ) "s-to-d"
```

Sign extend a single number *n* to form a double number *d*.

Definition:

```
: s>d ( n -- d )
  dup 0< ;
```

Origin: fig-Forth’s `s->d`, Forth-94 (CORE), Forth-2012 (CORE).

See also: `d>s`, `u>ud`.

Source file: <src/kernel.z80s>.

s\"

```
s\"
  Compilation:    ( "ccc<quote>" -- )
  Interpretation: ( "ccc<quote>" -- ca len )
  Run-time:       ( -- ca len )
"s-backslash-quote"
```

NOTE When `s\"` is loaded, `esc-standard-chars-wordlist` is set as the only word list by `set-esc-order`. That is the standard behaviour. Alternative escaped chars can be configured with `esc-block-chars-wordlist` and `esc-udg-chars-wordlist`.

`s\"` is an `immediate` word.

Origin: Forth-2012 (CORE EXT, FILE EXT).

See also: `parse-esc-string`, `set-esc-order`, `.\"`.

Source file: <src/lib/strings.escaped.fs>.

save-buffers

```
save-buffers ( -- )
```

If the disk buffer has been modified, transfer its contents to disk and mark it as unmodified.

Definition:

```
: save-buffers ( -- )
  updated? 0exit
  buffer-block dup write-block disk-buffer ! ;
```

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (BLOCK), Forth-2012 (BLOCK).

See also: `empty-buffers`, `flush`, `block`, `buffer`, `buffer-block`, `disk-buffer`.

Source file: <src/kernel.z80s>.

save-mode

```
save-mode ( -- )
```

Store the contents of `current-mode` into `previous-mode`.

`save-mode` is executed by `bye` before setting the default screen mode (e.g. `mode-32`, `mode-32iso`, `mode-`

`64ao`).

Definition:

```
: save-mode ( -- ) current-mode @ previous-mode ! ;
```

See also: `restore-mode`.

Source file: <src/kernel.z80s>.

sbc#,

```
sbc#, ( b -- ) "s-b-c-number-sign-comma"
```

Compile the Z80 `assembler` instruction `SBC A,b`.

Source file: <src/lib/assembler.fs>.

sbc,

```
sbc, ( reg -- ) "s-b-c-comma"
```

Compile the Z80 `assembler` instruction `SBC reg`.

See also: `sub,`, `adc,`, `add,`, `subp,`.

Source file: <src/lib/assembler.fs>.

sbcp,

```
sbcp, ( regp -- ) "s-b-c-p-comma"
```

Compile the Z80 `assembler` instruction `SBC HL,regp`.

See also: `subp,`, `sbc,`.

Source file: <src/lib/assembler.fs>.

sbcx,

```
sbcx, ( disp regpi --  ) "s-b-c-x-comma"
```

Compile the Z80 `assembler` instruction `SBC (regpi+disp)`.

See also: `subx,`, `adcx,`.

Source file: <src/lib/assembler.fs>.

scan

```
scan ( ca1 len1 c -- ca2 len2 )
```

Scan the string *ca1 len1* for the first occurence of character *c*. Leave match address *ca2* and length remaining *len2*. If no match occurred then *len2* is zero and *ca2* is *ca1*+*len1*.

Source file: <src/kernel.z80s>.

scf,

```
scf, ( -- ) "s-c-f-comma"
```

Compile the Z80 `assembler` instruction `SCF`.

See also: `cpl,`, `ccf,`, `neg,`, `set,`, `and,`.

Source file: <src/lib/assembler.fs>.

sconstant

```
sconstant ( ca len "name" -- ) "s-constant"
```

Create a character string constant *name* with value *ca len*. The character string is stored into data space. When *name* is later executed, it returns the corresponding *ca2 len*, being *ca2* the address where the original string was stored by `sconstant`.

See also: `sconstants`.

Source file: <src/lib/strings.MISC.fs>.

sconstants

```
sconstants ( 0 ca[n]..ca[1] "name" -- n ) "s-constants"
```

Create a table of string constants *name*, using counted strings *ca[n]..ca[1]*, being *0* a mark for the last string on the stack, and return the number *n* of compiled strings.

When *name* is executed, it converts the index on the stack (0..*n-1*) to the correspondent string *ca len*.

Usage example:

```
0               \ end of strings
  here ," kvar" \ string 4
  here ," tri"  \ string 3
  here ," du"   \ string 2
  here ," unu"  \ string 1
  here ," nul"  \ string 0
sconstants digitname
  constant digitnames

cr .( There are ) digitnames . .( digit names:)
0 digitname cr type
1 digitname cr type
2 digitname cr type
3 digitname cr type cr
```

See also: `sconstant`, `,"`, `begin-stringtable`.

Source file: <src/lib/strings.MISC.fs>.

scr

```
scr ( -- a ) "s-c-r"
```

A `user` variable. *a* is the address of a cell containing the number of the `block` most recently listed by `list`. `scr` is used by the block editors.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Controlled Reference Words), Forth-94 (BLOCK EXT), Forth-2012 (BLOCK EXT).

See also: `editor`.

Source file: <src/kernel.z80s>.

scra>attra

```
scra>attra ( a1 -- a2 ) "s-c-r-a-to-a-t-t-r-a"
```

Convert screen bitmap address *a1* to its correspondent attribute address *a2*.

Source file: <src/lib/graphics.pixels.fs>.

seal

```
seal ( -- )
```

Remove all word lists from the search order other than the word list that is currently on top of the search order. I.e., change the search order such that only the word list at the top of the search order will be searched.

Origin: Gforth.

See also: `only`, `set-order`, `#order`.

Source file: <src/lib/word_lists.fs>.

search

```
search ( ca1 len1 ca2 len2 -- ca3 len3 f )
```

Search the string *ca1 len1* for the string *ca2 len2*. If *f* is true, a match was found at *ca3* with *len3* characters remaining. If *f* is false there was no match and *ca3 len3* is *ca1 len1*.

Origin: Forth-94 (STRING), Forth-2012 (STRING).

See also: `compare`, `hunt`.

Source file: <src/kernel.z80s>.

search-wordlist

```
search-wordlist ( ca len wid -- 0 | xt 1 | xt -1 )
```

Find the definition identified by the string *ca len* in the word list identified by *wid*. If the definition is not found, return zero. If the definition is found, return its *xt* and one (1) if the definition is immediate, minus-one (-1) otherwise.

Origin: Forth-94 (SEARCH), Forth-2012 (SEARCH).

See also: `find-name`, `find-name-from`, `find-name-in`, `find`.

Source file: <src/lib/word_lists.fs>.

seclusion

```
seclusion ( -- wid1 wid2 )
```

Start a seclusion module. Private definitions follow.

Modules hide the internal implementation and leave visible the words of the outer interface.

wid1 is the identifier of the compilation word list before `seclusion` was executed. *wid2* is the identifier of the word list where private definitions of the seclusion module will be created. They are used by `-seclusion`, which marks the start of public definitions, `+seclusion`, which optionally marks the start of new private definitions, and `end-seclusion`, which ends the module.

Usage example:

```
seclusion
  \ Inner/private words.
-seclusion
  \ Interface/public words.
+seclusion
  \ More inner/private words.
-seclusion
  \ More interface/public words.
  \ Etc.
end-seclusion
```

A copy of *wid2* may be kept by the application in order to access private words later, e.g. `seclusion dup constant my-module`.

See also: `internal`, `isolate`, `module`, `package`, `privatize`.

Source file: <src/lib/modules.MISC.fs>.

seconds

```
seconds ( u -- )
```

Wait at least *u* seconds.

See also: `?seconds`, `ms`, `ticks`.

Source file: <src/lib/time.fs>.

sector#>dos

```
sector#>dos ( n -- x ) "sector-number-sign-to-dos"
```

Convert the sequential disk sector *n* to the disk sector id *x*, in the format required by G+DOS: The high byte of *x* is the track (0..79 for side 0; 128..207 for side 1); its low byte is the sector (1..10)..

Definition:

```
: sector#>dos ( n -- x )
  dup sectors/track mod 1+    ( n sector )
  swap dup 20 /               ( sector n track0 )
  swap sectors/track / 1 and  ( sector track0 side )
  negate 128 and or           ( sector track )
  sector>dos ;

  \ Notes:

  \ x (high byte) = track 0..79 for side 0, 128..207 for side 1
  \ x (low byte)  = sector 1..10
  \ track0        = 0..79
  \ track         = 0..207
  \ side          = 0..1
```

See also: `sectors/track`, `sector>dos`, `transfer-sector`.

Source file: <src/kernel.gplusdos.z80s>.

sector>dos

```
sector>dos ( sector track -- x ) "sector-to-dos"
```

Convert the 8-bit sector number *sector* and the 8-bit track number *track* to the 16-bit number *x* in the format used by G+DOS: The high byte of *x* is the track, and its low byte is the sector.

`sector>dos` is a factor of `block-sector#>dos`.

`sector>dos` is written in Z80. Its equivalent definition in Forth is the following:

```
: sector>dos ( sector track -- x ) flip or ;
```

Source file: <src/kernel.gplusdos.z80s>.

sectors-used

```
sectors-used ( -- n )
```

n is the number of sectors used in the current drive, not including the tracks used by the disk catalogue.

NOTE In order to count the sectors used, G+DOS has to do a catalogue. Therefore the execution of `sectors-used` includes `acat`. This problem may be solved in a future version of Solo Forth.

See also: `sectors-used@`, `drive-used`, `drive-unused`, `b/sector`, `tracks/cat`.

Source file: <src/lib/dos.gplusdos.fs>.

sectors-used@

```
sectors-used@ ( -- n )
```

n is the number of sectors used in the drive most recently catalogued, not including the tracks used by the disk catalogue.

NOTE G+DOS calculates the number of used sectors only during a catalogue, and then saves it into a variable in the Plus D memory. `sectors-used@` only fetches the value. Therefore the user must execute `cat` or `acat` before, in order to update the value, or use `sectors-used` directly.

See also: `drive-used`, `drive-unused`, `b/sector`, `tracks/cat`.

Source file: <src/lib/dos.gplusdos.fs>.

sectors/block

```
sectors/block ( -- b ) "sectors-slash-block"
```

A `cconstant`. *b* is the number of sectors per block.

See also: `b/sector`, `sectors/track`, `blocks/disk`.

Source file: <src/kernel.z80s>.

sectors/cat

```
sectors/cat ( -- b ) "sectors-slash-cat"
```

A `cconstant`, *b* is the number of sectors used by the disk catalogue (only one side of the disk is used for the catalogue).

See `tracks/cat`, `tracks/disk`, `sectors/disk`, `b/sector`, `sectors-used`, `drive-unused`.

Source file: <src/lib/dos.gplusdos.fs>.

sectors/disk

```
sectors/disk ( -- u ) "sectors-slash-disk"
```

A `constant`. *u* is the total number of sectors of a disk (on both sides).

See also: `tracks/cat`, `tracks/disk`, `b/sector`, `max-disk-capacity`, `sectors-used`, `drive-unused`.

Source file: <src/lib/dos.gplusdos.fs>.

sectors/track

```
sectors/track ( -- b ) "sectors-slash-track"
```

A `cconstant`. *b* is the number of sectors per track.

See also: `b/sector`, `sectors/block`, `blocks/disk`.

Source file: <src/kernel.z80s>.

sectors>capacity

```
sectors>capacity ( n1 -- n2 ) "sectors-to-capacity"
```

Convert number of disk sectors *n1* to the equivalent number of KiB *n2*, i.e. how much KiB can be stored in *n1* sectors.

See `sectors/disk`, `blocks/disk`, `max-disk-capacity`, `drive-unused`, `drive-used`.

Source file: <src/lib/dos.gplusdos.fs>.

see

```
see ( "name" -- )
```

Decode the word's definition *name*.

Origin: Forth-94 (TOOLS), Forth-2012 (TOOLS).

See also: `see-name`, `see-xt`, `see-colon`, `see-colon-body`, `see-colon-body>`.

Source file: <src/lib/tool.see.fs>.

see-colon

```
see-colon ( nt -- )
```

Decode the colon word's definition *nt*.

See also: `see`, `see-name`, `see-colon-body`.

Source file: <src/lib/tool.see.fs>.

see-colon-body

```
see-colon-body ( dfa -- )
```

Decode the colon word’s definition whose body is *dfa*. `see-colon-body` is a factor of `see-colon`.

See also: `see`, `see-colon-body>`, `see-xt`, `see-usage`.

Source file: <src/lib/tool.see.fs>.

see-colon-body>

```
see-colon-body> ( a -- ) "see-colon-body-to"
```

Decode the colon word’s definition from *a*, which is part of its body. `see-colon-body>` is useful to decode words that use `exit` in the midle of the definition, because `see` stops at the first `exit` found.

See also: `see-colon-body`, `see-xt`, `see-name`.

Source file: <src/lib/tool.see.fs>.

see-name

```
see-name ( nt -- )
```

Decode the word’s definition *nt*.

`see-name` is a factor of `see`.

See also: `see`, `see-xt`, `see-colon`.

Source file: <src/lib/tool.see.fs>.

see-usage

```
see-usage ( -- )
```

Display the usage of `see`. `see-usage` is executed when `manual-see` contains non-zero.

Source file: <src/lib/tool.see.fs>.

see-xt

```
see-xt ( xt -- ) "see-x-t"
```

Decode the word's definition *xt*.

The listing can be paused with the space bar, then stopped with the return key or resumed with any other key.

See also: `see`, `see-name`, `see-colon`.

Source file: <src/lib/tool.see.fs>.

sentry:

```
sentry: ( ca len wid "name" -- ) "s-entry-colon"
```

Create a string entry *name* in the `associative-list` *wid*, with value *ca len*.

See also: `entry:`, `centry:`, `2entry:`, `create-entry`.

Source file: <src/lib/data.associative-list.fs>.

set,

```
set, ( reg b -- ) "set-comma"
```

Compile the Z80 `assembler` instruction `SET b,reg`.

See also: `bit,`, `res,`, `add#,`.

Source file: <src/lib/assembler.fs>.

set-anon

```
set-anon ( x#n ... x#1 n -- )
```

Store the given *n* cells into the buffer pointed by `anon>`, which will be accessed by `anon`.

Usage example:

```
here anon> ! 5 cells allot

: test ( x4 x3 x2 x1 x0 -- )
  5 set-anon
  [ 0 ] anon ?     \ display _x0_
  123 [ 0 ] anon !
  [ 0 ] anon ?     \ display 123
  [ 2 ] anon ?     \ display _x2_
  555 [ 2 ] anon !
  [ 2 ] anon ?     \ display 555
  ;
```

Source file: <src/lib/locals.anon.fs>.

set-bit

```
set-bit ( b1 n -- b2 )
```

Set bit *n* of *b1*, returning the result *b2*.

See also: `bit?`, `set-bit`, `bit>mask`.

Source file: <src/lib/memory.MISC.fs>.

set-block-drives

```
set-block-drives ( c#n..c#1 n -- )
```

Set the drives specified by drive identifiers *c#n..c#1* as block drives. Subsequently drive *c#1* will be searched first for blocks from block number 0 to block number `blocks/disk 1-`; drive *c#2* will be searched for blocks from block number `blocks/disk` to block number `blocks/disk 2 * 1-`; and so on.

If *n* is zero, no drive is used for blocks.

NOTE `set-block-drives` sets `last-locatable` to the last block available on the new configuration, but `first-locatable` is not modified.

See also: `-block-drives`, `#block-drives`, `block-drive!`, `get-block-drives`.

Source file: <src/lib/dos.COMMON.fs>.

set-bright

```
set-bright ( f -- )
```

If *f* is `true`, turn bright on by setting the corresponding bit of the current attribute. If *f* is `false`, turn bright off by resetting the bit. Other non-zero values of *f* will turn bright on or off depending on them having a common bit with `bright-mask`.

See also: `get-bright`, `attr!`, `bright.`, `set-paper`, `set-ink`, `set-flash`, `bright-mask`.

Source file: <src/lib/display.attributes.fs>.

set-capslock

```
set-capslock ( -- )
```

Set capslock.

See also: `unset-capslock`, `capslock?`, `toggle-capslock`, `capslock`, `cset`.

Source file: <src/lib/keyboard.caps_lock.fs>.

set-circle-pixel

```
set-circle-pixel ( a -- )
```

Set the address *a* of the routine `circle-pixel` will jump to.

`set-circle-pixel` is used to make `circle-pixel` jump to `colored-circle-pixel`, `uncolored-circle-pixel`, or other routine provided by the application, therefore configuring `circle`.

Source file: <src/lib/graphics.circle.fs>.

set-code-file

```
set-code-file ( ca1 len1 ca2 len2 -- )
```

Configure `ufia` to use a code file in the current drive with filename *ca2 len2*, start address *ca1* and length *len1*.

See also: `set-filename`.

Source file: <src/lib/dos.gplusdos.fs>.

set-current

```
set-current ( wid -- )
```

Set the compilation word list to the word list identified by *wid*.

Definition:

```
: set-current ( wid -- ) current ! ;
```

Origin: Forth-94 (SEARCH), Forth-2012 (SEARCH).

Source file: <src/kernel.z80s>.

set-date

```
set-date ( day month year -- )
```

Set the current date. The default date is 2016-01-01. It can be fetch with `get-date`. The date is not updated by the system.

See also: `get-date`, `date`, `.date`, `leapy-year?`.

Source file: <src/lib/time.fs>.

set-drive

```
set-drive ( n -- ior )
```

Set the drive *n* (1 or 2) as the current one, returning the I/O result code *ior*. If the drive is successfully selected, *ior* is zero, otherwise it's an exception code. The default drive is 1.

See also: `get-drive`, `?set-drive`, `drive`, `set-block-drives`, `2-block-drives`.

Source file: <src/kernel.gplusdos.z80s>.

set-dticks

```
set-dticks ( d -- ) "set-d-ticks"
```

Set the system clock to *d* ticks.

See also: `set-ticks`, `dticks`, `reset-dticks`, `ticks/second`, `bench{`.

Source file: <src/lib/time.fs>.

set-esc-order

```
set-esc-order ( widn..wid1 n -- )
```

Set the escaped strings search order to the word lists identified by *widn..wid1*. Subsequently, word list *wid1* will be searched first, and word list *widn* searched last. If *n* is zero, empty the escaped strings search order.

See also: `get-esc-order`, `>esc-order`, `esc-standard-chars-wordlist`, `esc-block-chars-wordlist`, `esc-udg-chars-wordlist`.

Source file: <src/lib/strings.escaped.fs>.

set-filename

```
set-filename ( ca len -- )
```

Configure `ufia` to use filename *ca len* and the current drive.

See also: `-filename`, `/filename`.

Source file: <src/lib/dos.gplusdos.fs>.

set-flash

```
set-flash ( f -- )
```

If *f* is `true`, turn flash on by setting the corresponding bit of the current attribute. If *f* is `false`, turn flash off by resetting the bit. Other non-zero values of *f* will turn flash on or off depending on them having a common bit with `flash-mask`.

See also: `get-flash`, `attr!`, `flash.`, `set-paper`, `set-ink`, `set-bright`, `flash-mask`.

Source file: <src/lib/display.attributes.fs>.

set-font

```
set-font ( a -- )
```

Set address *a* as the current font by setting the system variable `os-chars`

`set-font` is used by all screen modes. The character bitmap *a* points to depends on the mode.

The last character used from the font can be configured by `last-font-char`.

See also: `get-font`, `rom-font`, `default-font`, `mode-32`, `mode-32iso`, `mode-42pw`, `mode-64ao`.

Source file: <src/kernel.z80s>.

set-heap

```
set-heap ( a u b -- )
```

Set the values of the current heap: its address *a* (returned by `heap`), its size *u* (returned by `/heap`) and its bank *b* (stored in `heap-bank`).

`set-heap` and `get-heap` are useful when more than one memory heap are needed by the application.

Source file: <src/lib/memory.allocate.COMMON.fs>.

set-ink

```
set-ink ( b -- )
```

Set ink color *b* (0..7) by modifying bits 0-2 of the current attribute.

`set-ink` is written in Z80. Its equivalent definition in Forth is the following:

```
: set-ink ( b -- ) attr@ unink-mask and or attr! ;
```

See also: `get-ink`, `attr!`, `ink.`, `set-paper`, `set-flash`, `set-bright`, `unink-mask`.

Source file: <src/lib/display.attributes.fs>.

set-menu

```
set-menu ( a1 a2 ca len col row n1 n2 -- )
```

Set the current menu to cursor coordinates *col row*, *n2* options, *n1* characters width, title *ca len*, actions table *a1* (a cell array of *n2* execution tokens) and option texts table *a2* (a cell array of *n2* addresses of counted strings).

See also: `new-menu`, `.menu`, `menu`, `menu-xy`, `menu-title`, `actions-table`, `menu-options`, `menu-width`, `menu-body-attr`, `menu-highlight-attr`, `menu-banner-attr`.

Source file: <src/lib/menu.sinclair.fs>.

set-mixer

```
set-mixer ( b -- )
```

Set the mixer register of the AY-3-8912 sound generator to *b*.

> Register 7 (Mixer - I/O Enable)
>
> This controls the enable status of the noise and tone mixers for the three channels, and also controls the I/O port used to drive the RS232 and Keypad sockets.
>
> | | |
> |---|---|
> | **Bit 0** | Channel A Tone Enable (0=enabled). |
> | **Bit 1** | Channel B Tone Enable (0=enabled). |
> | **Bit 2** | Channel C Tone Enable (0=enabled). |
> | **Bit 3** | Channel A Noise Enable (0=enabled). |
> | **Bit 4** | Channel B Noise Enable (0=enabled). |
> | **Bit 5** | Channel C Noise Enable (0=enabled). |
> | **Bit 6** | I/O Port Enable (0=input, 1=output). |
> | **Bit 7** | Not used. |

See also: `get-mixer`, `-mixer`, `!sound`.

Source file: <src/lib/sound.128.fs>.

set-mode-output

```
set-mode-output ( a -- )
```

Associate the output routine at *a* to the system channels "K", "S" and "P".

Source file: <src/lib/display.mode.COMMON.fs>.

set-order

```
set-order ( -1 | 0 | wid#n .. wid#1 n -- )
```

Set the search order to the word lists identified by *wid#n* .. *wid#1*. Subsequently, word list *wid1* will be searched first, and word list *wid#n* searched last. If *n* is zero, empty the search order. If *n* is minus one, set the search order to the implementation-defined minimum search order.

Definition:

```
: set-order ( -1 | 0 | wid#n .. wid#1 n -- )
  dup -1 = if  drop root-wordlist dup 2  then
  dup ?order  dup #order !
  0 ?do  i cells context + !  loop ;
```

Origin: Forth-94 (SEARCH), Forth-2012 (SEARCH).

See also: `get-order`, `>order`.

Source file: <src/kernel.z80s>.

set-paper

```
set-paper ( b -- )
```

Set paper color *b* (0..7) by modifying the corresponding bits of the current attribute.

`set-paper` is written in Z80. Its equivalent definition in Forth is the following:

```
: set-paper ( b -- ) papery attr@ unpaper-mask and or attr! ;
```

See also: `get-paper`, `attr!`, `paper.`, `set-ink`, `set-flash`, `set-bright`, `paper-mask`.

Source file: <src/lib/display.attributes.fs>.

set-pixel

```
set-pixel ( gx gy -- )
```

Set a pixel without changing its attribute on the screen or the current graphic coordinates. *gx* is 0..255; *gy* is 0..191.

See also: `plot`, `plot176`, `reset-pixel`, `toggle-pixel`, `xy>gxy`.

Source file: <src/lib/graphics.pixels.fs>.

set-pixel176

```
set-pixel176 ( gx gy -- ) "set-pixel-176"
```

Set a pixel without changing its attribute on the screen or the current graphic coordinates, and using only the top 176 pixel rows of the screen (the lower 16 pixel rows are not used). *gx* is 0..255; *gy* is 0..175.

See also: `set-save-pixel176`, `set-pixel`, `plot`, `plot176`, `reset-pixel`, `toggle-pixel`, `reset-pixel176`, `toggle-pixel176`, `xy>gxy176`.

Source file: <src/lib/graphics.pixels.fs>.

set-save-pixel176

```
set-save-pixel176 ( gx gy -- ) "set-save-pixel-176"
```

Set a pixel without changing its attribute on the screen, and using only the top 176 pixel rows of the screen (the lower 16 pixel rows are not used). *gx* is 0..255; *gy* is 0..175. `set-save-pixel176` updates the graphic coordinates (contrary to `set-pixel176`).

See also: `set-pixel`, `plot`, `plot176`, `reset-pixel`, `toggle-pixel`, `reset-pixel176`, `toggle-pixel176`.

Source file: <src/lib/graphics.pixels.fs>.

set-source

```
set-source ( ca len -- )
```

Set the memory zone *ca len* as the current source by pointing `input-buffer` to it.

Definition:

```
: set-source ( ca len -- ) input-buffer 2! >in off ;
```

See also: `>in`, `terminal>source`, `block>source`.

Source file: <src/kernel.z80s>.

set-tape-filename

```
set-tape-filename ( ca len -- )
```

Store filename *ca len* into the `tape-filename` field of `tape-header`.

Source file: <src/lib/tape.fs>.

set-tape-memory

```
set-tape-memory ( ca len -- )
```

Configure `tape-header` with the memory zone *ca len* (to be read or written), by storing *len* into `tape-`

`length` and *ca* into `tape-start`.

Source file: <src/lib/tape.fs>.

set-ticks

```
set-ticks ( d -- )
```

Set the system clock to *n* ticks.

See also: `set-dticks`, `ticks`, `reset-ticks`, `ticks/second`, `bench{`.

Source file: <src/lib/time.fs>.

set-time

```
set-time ( second minute hour -- )
```

Set the current time.

See also: `get-time`.

Source file: <src/lib/time.fs>.

set-udg

```
set-udg ( a -- ) "set-u-d-g"
```

Set address *a* as the the current UDG set (characters 0..255), by changing the system variable `os-udg`. *a* must be the bitmap address of character 0.

See also: `get-udg`, `set-font`.

Source file: <src/lib/graphics.udg.fs>.

setx,

```
setx, ( disp regpi b --  ) "set-x-comma"
```

Compile the Z80 `assembler` instruction `SET b,(regpi+disp)`.

See also: `bitx,`, `resx,`, `addx,`, `adcx,`, `andx,`, `xorx,`, `orx,`, `incx,`.

Source file: <src/lib/assembler.fs>.

sfalign

```
sfalign ( -- ) "s-f-align"
```

If the data space is not single-float aligned, reserve enough space to make it so.

In Solo Forth, `sfalign` does nothing: it's an `immediate` `alias` of `noop`.

Origin: Forth-94 (FLOATING EXT), Forth-2012 (FLOATING EXT).

See also: `sfaligned`, `falign`, `dfalign`, `float`.

Source file: <src/lib/math.floating_point.rom.fs>.

sfaligned

```
sfaligned ( a -- fa ) "s-f-aligned"
```

fa is the first single-float-aligned address greater than or equal to *a*

In Solo Forth, `sfaligned` does nothing: it's an `immediate` `alias` of `noop`.

Origin: Forth-94 (FLOATING EXT), Forth-2012 (FLOATING EXT).

See also: `sfalign`, `faligned`, `dfaligned`, `float`.

Source file: <src/lib/math.floating_point.rom.fs>.

sign

```
sign ( n  --  )
```

If *n* is negative, add a minus sign to the beginning of the pictured numeric output string.

Definition:

```
: sign ( n -- ) 0< if '-' hold then ;
```

Origin: Forth 94 (CORE), Forth-2012 (CORE).

See also: `<#`, `#>`, `hold`.

Source file: <src/kernel.z80s>.

silence

```
silence ( -- )
```

Execute `-mixer` to disable the noise and tone mixers for the three channels of the AY-3-8912 sound generator. Then set the volume of the three channels to zero.

See also: `!volume`.

Source file: <src/lib/sound.128.fs>.

simple-accept

```
simple-accept ( ca1 len1 -- len2 )
```

Receive a string of at most *len1* characters. No characters are received or transferred if *len1* is zero. Display graphic characters as they are received.

Input terminates when the Return key is pressed. When input terminates, nothing is appended to the string or displayed on the screen.

The only control key accepted is Delete.

len2 is the length of the string stored at *ca1*.

`simple-accept` is the default action of the deferred word `accept` (see `defer`).

Definition:

```
: simple-accept ( ca len -- len' )
  over + over ( bot eot cur )
  begin  xkey dup 13 <> \ not carriage return?
  while ( bot eot cur c )
    dup 12 =  \ delete?
    if    drop  >r over r@ < dup  \ any chars?
          if  8 dup emit  bl emit  emit  then  r> +
    else  \ maybe printable
          >r  2dup <>  \ more?
          r@ [ bl 1- ] literal > and  \ and printable?
          if  r@ over c!  char+  r@ emit  then  r> drop
    then
  repeat ( bot eot cur c ) drop nip swap - ;
```

See also: `query`.

Source file: <src/kernel.z80s>.

sinclair-stripes

```
sinclair-stripes ( -- ca )
```

Return address *ca* where the following pair of UDG definitions, used to create Sinclair stripes, are stored:

```
0 0 0 0 0 0 0 1          X
0 0 0 0 0 0 1 1         XX
0 0 0 0 0 1 1 1        XXX
0 0 0 0 1 1 1 1       XXXX
0 0 0 1 1 1 1 1      XXXXX
0 0 1 1 1 1 1 1     XXXXXX
0 1 1 1 1 1 1 1    XXXXXXX
1 1 1 1 1 1 1 1   XXXXXXXX

1 1 1 1 1 1 1 0   XXXXXXX
1 1 1 1 1 1 0 0   XXXXXX
1 1 1 1 1 0 0 0   XXXXX
1 1 1 1 0 0 0 0   XXXX
1 1 1 0 0 0 0 0   XXX
1 1 0 0 0 0 0 0   XX
1 0 0 0 0 0 0 0   X
0 0 0 0 0 0 0 0
```

See also: `.sinclair-stripes`, `sinclair-stripes$`.

Source file: <src/lib/menu.sinclair.fs>.

sinclair-stripes$

```
sinclair-stripes$ ( -- ca len )
```

Return a string *ca len* containing the following character codes:

Table 33. Characters of `sinclair-stripes$`.

Code(s)	Meaning
$10 $02	set ink 2 (red)
$80	first stripe UDG
$11 $06	set paper 6 (yellow)
$81	second stripe UDG
$10 $04	set ink 4 (green)
$80	first stripe UDG

Code(s)	**Meaning**
$11 $05	set paper 5 (cyan)
$81	second stripe UDG
$10 $00	set ink 0 (black)
$80	first stripe UDG

Definitions for UDG codes $80 and $81 are provided optionally by `sinclair-stripes`.

See also: `.sinclair-stripes`.

Source file: <src/lib/menu.sinclair.fs>.

skip

```
skip ( ca1 len1 c -- ca2 len2 | ca1 len1 )
```

Skip over leading occurences of the character *c* in the string *ca1 len1*. Leave the address of the first non-matching character *ca2* and length remaining *len2*. If no characters were skipped leave *ca1 len1*.

Source file: <src/kernel.z80s>.

skip-sign?

```
skip-sign? ( ca len -- ca' len' f ) "skip-sign-question"
```

If number string *ca len* starts with a minus sign, remove it and return the result string *ca' len'* and a true flag *f*; else *ca' len'* is identical to *ca len* and *f* is false.

Definition:

```
: skip-sign? ( ca len -- ca' len' f )
  over c@ '-' = dup >r abs /string r> ;
```

See also: `number?`, `?negate`.

Source file: <src/kernel.z80s>.

sla,

```
sla, ( reg -- ) "s-l-a-comma"
```

Compile the Z80 `assembler` instruction `SLA reg`.

Source file: <src/lib/assembler.fs>.

slax,

```
slax, ( disp regpi --  ) "s-l-a-x-comma"
```

Compile the Z80 `assembler` instruction `SLA (regpi+disp)`.

See also: `rlcx,`, `rrcx,`, `rlx,`, `rrx,`, `srax,`, `sllx,`, `srlx,`, `bitx,`, `resx,`, `setx,`.

Source file: <src/lib/assembler.fs>.

slit

```
slit ( -- ca len ) "s-lit"
```

Return a string that is compiled after the calling word, and adjust the instruction pointer to step over the inline string.

Definition:

```
: slit ( -- ca len ) r@ count dup char+ r> + >r ;
```

Source file: <src/kernel.z80s>.

sliteral

```
sliteral "s-literal"
  Compilation: ( ca1 len1 -- )
  Run-time:    ( -- ca2 len1 )
```

Compile `slit` and string *ca len* in the current definition. At run-time `slit` will return string *ca1 len1* as *ca2 len1*.

`sliteral` is an `immediate` and `compile-only` word.

Definition:

```
: sliteral ( ca len -- )
  postpone slit s, ; immediate compile-only
```

Origin: Forth-94 (STRING), Forth-2012 (STRING).

See also: `s,`, `csliteral`.

Source file: <src/kernel.z80s>.

sll,

```
sll, ( reg -- ) "s-l-l-comma"
```

Compile the Z80 `assembler` instruction `SLL reg`.

Source file: <src/lib/assembler.fs>.

sllx,

```
sllx, ( disp regpi --  ) "s-l-l-x-comma"
```

Compile the Z80 `assembler` instruction `SLL (regpi+disp)`.

See also: `rlcx,`, `rrcx,`, `rlx,`, `rrx,`, `slax,`, `srax,`, `srlx,`, `bitx,`, `resx,`, `setx,`.

Source file: <src/lib/assembler.fs>.

slow-gxy>scra_

```
slow-gxy>scra_ ( -- a ) "slow-g-x-y-to-s-c-r-a-underscore"
```

Return address *a* of an alternative entry point to the PIXEL-ADD ROM routine ($22AA), to let the range of the y coordinate be 0..191 instead of 0..175.

`slow-gxy>scra_` is the default action of `gxy>scra_`.

When `fast-gxy>scra_` (which is faster but bigger, and requires the assembler) is needed, the application must use `need fast-gxy>scra_` before `need set-pixel` or any other word that needs `gxy>scra_`.

Input registers:

- C = x cordinate (0..255)
- B = y coordinate (0..191)

Output registers:

- HL = address of the pixel byte in the screen bitmap
- A = position of the pixel in the byte address (0..7), note: position 0=bit 7, position 7=bit 0.

See also: `gxy176>scra_`.

Source file: <src/lib/graphics.pixels.fs>.

slow-pixels

```
slow-pixels ( -- n )
```

Return the number *u* of pixels that are set on the screen. `slow-pixels` is the alternative action of the deferred word `pixels` (see `defer`). `slow-pixels` simply executes `bits` with the screen address and length on the stack.

See also: `fast-pixels`.

Source file: <src/lib/graphics.pixels.fs>.

sm/rem

```
sm/rem ( d n1 -- n2 n3 ) "s-m-slash-rem"
```

Symmetric division:

```
D = n3*n1+n2;

sign(n2) = sign(d1) or 0
```

Divide *d* by *n1*, giving the symmetric quotient *n3* and the remainder *n2*. Input and output stack arguments are signed.

Table 34. Symmetric Division Example

Dividend	Divisor	Remainder	Quotient
10	7	3	1
-10	7	-3	-1
10	-7	3	-1
-10	-7	-3	1

Definition:

```
: sm/rem ( d1 n1 -- n2 n3 ) \ symmetric signed division
  2dup xor >r  \  sign of quotient
  over >r      \  sign of remainder
  abs >r dabs r> um/mod
  swap r> ?negate
  swap r> ?negate ;
```

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `fm/mod`, `m/`.

Source file: <src/kernel.z80s>.

smove

```
smove ( ca1 len1 ca2 -- ) "s-move"
```

Move the string *ca1 len1* to *ca2*.

`smove` is the equivalent of the idiom `swap move`, but faster.

See also: `cmove`, `cmove>`, `move`.

Source file: <src/kernel.z80s>.

smudge

```
smudge ( -- )
```

Toggle the "smudge bit" of the `latest` definition’s name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without error.

`smudge` is obsolete. `hide` and `reveal` are used instead.

Origin: fig-Forth.

See also: `smudged`.

Source file: <src/lib/compilation.fs>.

smudge-mask

```
smudge-mask ( -- b )
```

A `cconstant`. *b* is the bitmask of the `smudge` bit.

See also: `word-length-mask`, `immediate-mask`, `compile-only-mask`.

Source file: <src/kernel.z80s>.

smudged

```
smudged ( nt -- )
```

Toggle the "smudge bit" of the given *nt*.

`smudged` is obsolete. `hidden` and `revealed` are used instead.

See also: `smudge`, `smudge-mask`.

Source file: <src/lib/compilation.fs>.

sound

```
sound ( b[0]..b[13] "name" -- )
```

Create a word *name* that will play the 14-byte sound defined by *b[0]..b[13]*.

See also: `sound,`, `play`, `edit-sound`.

Source file: <src/lib/sound.128.fs>.

sound,

```
sound, ( b[0]..b[13] -- ) "sound-comma"
```

Compile the 14-byte sound definition *b[0]..b[13]*.

See also: `play`, `sound`.

Source file: <src/lib/sound.128.fs>.

sound-register-port

```
sound-register-port ( -- a )
```

The I/O port used to select a register of the AY-3-8912 sound generator, before writing a value into it using `sound-write-port`, or before reading a value from it using `sound-register-port` again.

`sound-register-port` is a fast constant defined with `const`. Its value is $FFFD.

See also: `sound-write-port`, `!sound`, `@sound`.

Source file: <src/lib/sound.128.fs>.

sound-write-port

```
sound-write-port ( -- a )
```

The I/O port used to write to a register of the AY-3-8912 sound generator.

`sound-write-port` is a fast constant defined with `const`. Its value is $BFFD.

See also: `sound-register-port`, `!sound`, `@sound`.

Source file: <src/lib/sound.128.fs>.

source

```
source ( -- ca len )
```

ca is the address of, and *len* is the number of characters in, the input buffer.

Definition:

```
: source ( -- ca len )
  blk @ ?dup if block b/buf exit then
  input-buffer 2@ ;
```

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `input-buffer`, `set-source`, `blk`, `stream`, `nest-source`, `unnest-source`.

Source file: <src/kernel.z80s>.

source-id

```
source-id ( -- 0 | -1 ) "source-i-d"
```

Identify the input source as follows:

Table 35. Values returned by `source-id`.

Value	Input source
0	User input device
-1	String via `evaluate`

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

Source file: <src/kernel.z80s>.

sp

```
sp ( -- regp ) "s-p"
```

Return the identifier *reg* of the Z80 `assembler` register "SP".

See also: `a`, `b`, `c`, `d`, `e`, `h`, `l`, `m`, `ix`, `iy`.

Source file: <src/lib/assembler.fs>.

sp!

```
sp! ( a -- ) "s-p-store"
```

Store *a* into the stack pointer.

Source file: <src/kernel.z80s>.

sp0

```
sp0 ( -- a ) "s-p-zero"
```

A `user` variable. *a* is the address of a cell containing the address of the bottom of the data stack.

Origin: fig-Forth’s `s0`, Forth-79’s `s0`, Forth-83’s `s0`.

See also: `sp@`, `sp!`.

Source file: <src/kernel.z80s>.

sp@

```
sp@ ( -- a ) "s-p-fetch"
```

Fetch the content of the stack pointer. *a* is the address of the top of the stack just before `sp@` was executed.

Origin: fig-Forth, Forth-79 (Reference Word Set), Forth-83 (Controlled Reference Words).

See also: `sp!`, `sp0`.

Source file: <src/kernel.z80s>.

space

```
space ( -- )
```

Display one space.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `bl`, `emit`.

Source file: <src/kernel.z80s>.

spaces

```
spaces ( n -- )
```

If *n* is greater than zero, display *n* spaces.

`spaces` is written in Z80. Its equivalent definition in Forth is the following:

```
: spaces ( n -- ) bl swap emits ;
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `space`, `bl`, `emits`.

Source file: <src/kernel.z80s>.

specforth-editor

```
specforth-editor ( -- )
```

A `vocabulary` containing the Specforth block editor. When `specforth-editor` is loaded, it becomes the action of `editor`.

Table 36. Specforth block editor commands

Word	Description
`b ( -- )`	Used after `f` to backup the cursor by the length of the most recent text.
`c ( "ccc<eol>" -- )`	Copy in *ccc* to the cursor line at the cursor position.
`clear ( n -- )`	Clear block *n* with blanks and select for editing.
`copy ( n1 n2 -- )`	Copy block *n1* to block *n2*.
`d ( n -- )`	Delete line *n* but hold it in `pad`. Line 15 becomes free as all statements move up one line.
`delete ( n -- )`	Delete *n* characters prior to the cursor.
`e ( n -- )`	Erase line *n* with blanks.
`find, ( -- )`	Search for a match to the string at `pad`, from the cursor position until the end of block. If no match found issue an error message and reposition the cursor at the top of the block.
`h ( n -- )`	Hold line *n* at `pad` (used by system more often than by user).
`i ( n -- )`	Insert text from `pad` at line *n*, moving the old line *n* down. Line 15 is lost.

Word	Description
`l ( -- )`	List the current block.
`m ( n -- )`	Move the cursor by *n* characters. The position of the cursor on its line is shown by a "_" (underline).
`n ( -- )`	Find the next occurrence of the string found by an `f` command.
`p ( n "ccc<eol>" -- )`	Put *ccc* on line *n*.
`r ( n -- )`	Replace line *n* with the text in `pad`.
`s ( n -- )`	Spread at line *n*. Line *n* and following lines are are moved down one line. Line *n* becomes blank. Line 15 is lost.
`t ( n -- )`	Type line *n* and save in `pad`.
`till ( "ccc<eol>" -- )`	Delete on the cursor line from the cursor till the end of string *ccc*.
`x ( "ccc<eol>" -- )`	Find and delete the next occurrence of the string *ccc*.

See also: `gforth-editor`.

Source file: <src/lib/prog.editor.specforth.fs>.

split

```
split ( x -- b1 b2 )
```

b1 is the low byte of *x* and *b2* is the high byte of *x*.

Origin: IsForth, CHForth.

See also: `join`, `flip`.

Source file: <src/lib/math.operators.1-cell.fs>.

sqrt

```
sqrt ( n1 -- n2 ) "square-root"
```

Calculate integer square root *n2* of radicand *n1*. `sqrt` is a deferred word (see `defer`) which can execute `baden-sqrt` or `newton-sqrt`.

Source file: <src/lib/math.operators.1-cell.fs>.

sra,

```
sra, ( reg -- ) "s-r-a-comma"
```

Compile the Z80 `assembler` instruction `SRA reg`.

Source file: <src/lib/assembler.fs>.

srax,

```
srax, ( disp regpi --  ) "s-r-a-x-comma"
```

Compile the Z80 `assembler` instruction `SRA (regpi+disp)`.

See also: `rlcx,`, `rrcx,`, `rlx,`, `rrx,`, `slax,`, `sllx,`, `srlx,`, `bitx,`, `resx,`, `setx,`.

Source file: <src/lib/assembler.fs>.

srl,

```
srl, ( reg -- ) "s-r-l-comma"
```

Compile the Z80 `assembler` instruction `SRL reg`.

Source file: <src/lib/assembler.fs>.

srlx,

```
srlx, ( disp regpi --  ) "s-r-l-x-comma"
```

Compile the Z80 `assembler` instruction `SRL (regpi+disp)`.

See also: `rlcx,`, `rrcx,`, `rlx,`, `rrx,`, `slax,`, `srax,`, `sllx,`, `bitx,`, `resx,`, `setx,`.

Source file: <src/lib/assembler.fs>.

sstr1

```
sstr1 ( -- ca )
```

Return address *ca* of the stream number in the current `ufia`.

See also: `dstr1`, `fstr1`, `device`, `nstr1`, `nstr2`, `hd00`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

st#x,

```
st#x, ( 8b disp regpi -- ) "s-t-number-sign-x-comma"
```

Compile the Z80 `assembler` instruction `LD (regpi+disp),8b`.

See also: `stx,`.

Source file: <src/lib/assembler.fs>.

sta,

```
sta, ( a -- ) "s-t-a-comma"
```

Compile the Z80 `assembler` instruction `LD (a),A`, i.e. store the contents of register "A" into memory address *a*.

See also: `fta,`, `ld,`, `ld#,`.

Source file: <src/lib/assembler.fs>.

stack-cells

```
stack-cells ( -- n )
```

n is the maximum size of the data stack, in cells.

See also: `return-stack-cells`, `environment?`.

Source file: <src/lib/environment-question.fs>.

standard-number-point?

```
standard-number-point? ( c -- f ) "standard-number-point-question"
```

f is true if if character *c* is a valid point in a number. The only allowed point is period.

`standard-number-point?` is the default action of the deferred word `number-point?` (see `defer`), which is used in `number?`.

Definition:

```
: standard-number-point? ( c -- f ) '.' = ;
```

See also: `classic-number-point?`, `extended-number-point?`.

Source file: <src/kernel.z80s>.

stap,

```
stap, ( regp -- ) "s-t-a-p-comma"
```

Compile the Z80 `assembler` instruction `LD (regp),A`.

See also: `ftap,`.

Source file: <src/lib/assembler.fs>.

state

```
state ( -- a )
```

A `user` variable. *a* is the address of a cell containing the compilation-state flag, which is true when in compilation state, false otherwise.

Origin: fig-Forth, Forth-89 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `compiling?`, `[`, `]`.

Source file: <src/kernel.z80s>.

step

```
step
  Compilation: ( dest -- )
  Run-time:    ( R: n -- n' )
```

Compilation: (dest —)

Append the run-time semantics given below to the current definition. Resolve the destination of `for`.

Run-time: (R: n — n')

If the loop index is zero, discard the loop parameters and continue execution after the loop. Otherwise decrement the loop index and continue execution at the beginning of the loop.

`step` is an `immediate` and `compile-only` word.

NOTE `step` is usually called `next` in other Forth systems.

Origin: Z88 CamelForth.

Source file: <src/lib/flow.for.fs>.

sthl,

```
sthl, ( a -- ) "s-t-h-l-comma"
```

Compile the Z80 `assembler` instruction `LD (a),HL`, i.e. store the contents of register pair "HL" into memory address *a*.

See also: `fthl,`, `stp,`.

Source file: <src/lib/assembler.fs>.

storer

```
storer ( x a "name" -- )
```

Define a word *name* which, when executed, will cause that *x* be stored at *a*.

Origin: word `set` found in Forth-79 (Reference Word Set) and Forth-83 (Appendix B. Uncontrolled Reference Words).

Source file: <src/lib/data.storer.fs>.

stp,

```
stp, ( a regp -- ) "s-t-p-comma"
```

Compile the Z80 `assembler` instruction `LD (a),regp`, i.e. store the contents of pair register *regp* into memory address *a*.

NOTE	For the "HL" register there is a specific word: `fthl,`, which compiles shorten and faster code.

See also: `ftp,`.

Source file: <src/lib/assembler.fs>.

stpx,

```
stpx, ( disp regpi regp -- ) "s-t-p-x-comma"
```

Compile the Z80 `assembler` instructions required to store register pair *regp* into the address pointed by *regpi* plus *disp*.

Example: `16 ix h stpx,` will compile the Z80 instructions `LD (IX+16),L` and `LD (IX+17),H`.

See also: `ftpx,`, `stx,`.

Source file: <src/lib/assembler.fs>.

str<

```
str< ( ca1 len1 ca2 len2 -- f ) "s-t-r-less-than"
```

Is string *ca1 len1* lexicographically smaller than string *ca2 len2*?

See also: `str>`, `str=`, `str<>`, `compare`.

Source file: <src/lib/strings.MISC.fs>.

str<>

```
str<> ( ca1 len1 ca2 len2 -- f ) "s-t-r-not-equals"
```

Is string *ca1 len1* lexicographically not equal to string *ca2 len2*?

See also: `str=`, `str<`, `str>`, `compare`.

Source file: <src/lib/strings.MISC.fs>.

str=

```
str= ( ca1 len1 ca2 len2 -- f ) "s-t-r-equals"
```

f is true if string *ca1 len1* is lexicographically equal to string *ca2 len2*.

Definition:

```
: str= ( ca1 len1 ca2 len2 -- f ) compare 0= ;
```

See also: `str<>`, `str<`, `str>`, `compare`.

Source file: <src/kernel.z80s>.

str>

```
str> ( ca1 len1 ca2 len2 -- f ) "s-t-r-greater-than"
```

Is string *ca1 len1* lexicographically larger than string *ca2 len2*?

See also: `str<`, `str=`, `str<>`, `compare`.

Source file: <src/lib/strings.MISC.fs>.

stream

```
stream ( -- ca len )
```

String *ca len* is the the remaining stream `source`.

Definition:

```
: stream ( -- ca len ) source >in @ /string ;
```

See also: `>in`, `/string`.

Source file: <src/kernel.z80s>.

stream>

```
stream> ( n -- a )
```

Convert stream number *n* to address *a* of its corresponding element in `os-strms`.

See also: `first-stream`, `last-stream`, `stream?`.

Source file: <src/lib/os.fs>.

stream?

```
stream? ( -- false | n true )
```

If there's a closed stream, return its number *n* and `true`; otherwise return `false`.

See also: `os-strms`, `.os-strms`, `first-stream`, `last-stream`, `stream>`.

Source file: <src/lib/os.fs>.

string-char?

```
string-char? ( ca len c -- f ) "string-char-question"
```

Is char *c* in string *ca len*?

See also: `char-in-string?`, `char-position?`, `contains`, `compare`, `#chars`.

Source file: <src/lib/strings.MISC.fs>.

string-parameter

```
string-parameter ( -- ca len )
```

Return a string compiled after the calling word.

See `warning"` and `(warning"` for a usage example.

Source file: <src/lib/compilation.fs>.

string/

```
string/ ( ca1 len1 len2 -- ca2 len2 ) "string-slash"
```

Return the *len2* ending characters of string *ca1 len1*.

See also: `/string`.

Source file: <src/lib/strings.MISC.fs>.

string>source

```
string>source ( ca len -- ) "string-to-source"
```

Set the string *ca len* as the current source.

See also: `set-source`, `(source-id`.

Source file: <src/lib/parsing.fs>.

stringer

```
stringer ( -- a )
```

A `constant`. *a* is the base address of the `stringer`, which is the circular string buffer used by all string operations.

A program can move and resize the `stringer` if needed. Example:

```
stringer /stringer 2constant old-stringer
  \ Keep the address and length of the old stringer, in order
  \ to reuse its space later.

need !>

here 1024 dup allot !> /stringer !> stringer empty-stringer
  \ Create a new, 1024-byte ``stringer`` in data space.
```

The default `stringer` can be restored by `default-stringer`.

See also: `!>`, `/stringer`, `empty-stringer`, `+stringer`, `unused-stringer`, `fit-stringer`, `allocate-stringer`, `>stringer`.

Source file: <src/kernel.z80s>.

stx,

```
stx, ( reg disp regpi -- ) "s-t-x-comma"
```

Compile the Z80 `assembler` instruction `LD (regpi+disp),reg`.

See also: `st#x,`, `ftx,`.

Source file: <src/lib/assembler.fs>.

sub#,

```
sub#, ( b -- ) "sub-number-sign-comma"
```

Compile the Z80 `assembler` instruction `SUB b`.

Source file: <src/lib/assembler.fs>.

sub,

```
sub, ( reg -- ) "sub-comma"
```

Compile the Z80 `assembler` instruction `SUB reg`.

See also: `sbc,`, `add,`, `adc,`, `subp,`.

Source file: <src/lib/assembler.fs>.

subp,

```
subp, ( regp -- ) "sub-p-comma"
```

Compile the Z80 `assembler` instructions required to subtract register pair *regp* from register pair "HL".

Example: `d subp,` compiles the Z80 instructions `AND A` (to reset the carry flag) and `SBC DE`.

See also: `sbcp,`, `sub,`, `ldp,`, `tstp,`.

Source file: <src/lib/assembler.fs>.

substitute

```
substitute ( ca1 len1 ca2 len2 -- ca2 len3 n )
```

Perform substitution on the string *ca1 len1* placing the result at string *ca2 len3*, where *len3* is the length of the resulting string. An error occurs if the resulting string will not fit into *ca2 len2* or if *ca2* is the same as *ca1*. The return value *n* is positive or 0 on success and indicates the number of substitutions made. A negative value for *n* indicates that an error occurred, leaving *ca2 len3* undefined, and being *n* the exception code.

Substitution occurs left to right from the start of *ca1* in one pass and is non-recursive. When text of a potential substitution name, surrounded by "%" (ASCII $25) delimiters is encountered by `substitute`, the following occurs:

1. If the name is null, a single delimiter character is passed to the output, i.e., "%%" is replaced by "%". The current number of substitutions is not changed.
2. If the text is a valid substitution name acceptable to `replaces`, the leading and trailing delimiter characters and the enclosed substitution name are replaced by the substitution text. The current number of substitutions is incremented.
3. If the text is not a valid substitution name, the name with leading and trailing delimiters is passed unchanged to the output. The current number of substitutions is not changed.
4. Parsing of the input string resumes after the trailing delimiter.

If after processing any pairs of delimiters, the residue of the input string contains a single delimiter, the residue is passed unchanged to the output.

See also: `unescape`, `substitution-delimiter?`.

Source file: <src/lib/strings.substitute.fs>.

substitute-wordlist

```
substitute-wordlist ( -- wid )
```

Word list for substitution names and replacement texts.

See also: `replaces`.

Source file: <src/lib/strings.replaces.fs>.

substitution

```
substitution ( ca1 len1 -- ca2 )
```

Given a string *ca1 len1* create its substitution and storage space. Return the address of the buffer for the substitution text.

See also: `replaces`.

Source file: <src/lib/strings.replaces.fs>.

substitution-delimiter

```
substitution-delimiter ( -- c )
```

A character constant that returns the character used as delimiter by `substitute`. By default it's "%".

See also: `substitution-delimiter?`.

Source file: <src/lib/strings.substitute.fs>.

substitution-delimiter?

```
substitution-delimiter? ( ca -- f ) "substitution-delimiter-question"
```

Does *ca* contains the character hold in the character constant `substitution-delimiter`? If so return `true`, else return `false`.

`substitution-delimiter?` is a factor of `substitute`.

`substitution-delimiter?` is written in Z80. Its equivalent definition is Forth is the following:

```
: substitution-delimiter? ( ca -- f )
 c@ substitution-delimiter = ;
```

Source file: <src/lib/strings.substitute.fs>.

subx,

```
subx, ( disp regpi --  ) "sub-x-comma"
```

Compile the Z80 `assembler` instruction `SUB (regpi+disp)`.

See also: `sbcx,`, `addx,`.

Source file: <src/lib/assembler.fs>.

suffix?

```
suffix? ( ca1 len1 ca2 len2 -- f ) "suffix-question"
```

Is string *ca2 len2* the suffix of string *ca1 len1*?

See also: `-suffix`, `prefix?`.

Source file: <src/lib/strings.MISC.fs>.

swap

```
swap ( x1 x2 -- x2 x1 )
```

Exchange the top two stack items.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `2swap`, `over`, `tuck`.

Source file: <src/kernel.z80s>.

swap-current

```
swap-current ( wid1 -- wid2 )
```

Exchange the contents of the current compilation word list, which is identified by *wid2*, with the word list identified by *wid1*.

Origin: lpForth.

Source file: <src/lib/word_lists.fs>.

swapped

```
swapped ( i*x n1 n2 -- j*x )
```

Remove *n1* and *n2*. Swap elements *n1* and *n2* of the stack, being 0 the top of the stack. `0 1 swapped` is equivalent to `swap`.

Usage example:

```
  ( 1 2 3 4 5 ) 1 4 swapped ( 4 2 3 1 5 )
```

Source file: <src/lib/data_stack.fs>.

switch

```
switch ( x switch -- )
```

Execute the switch *switch* for the key *x*.

See also: `switch:`.

Source file: <src/lib/flow.switch-colon.fs>.

switch:

```
switch: ( "name" -- ) "switch-colon"
```

Create a new switch control structure *name*, which is a word list the clauses of the structure will be added to.

The keys can be 1-byte, 1-cell or 2-cell numbers, but the correspondent words must be used to create the clauses and execute them later:

Usage example:

```
switch: mynumber

\ Define clauses:

0       mynumber :clause  ( -- ) cr ." zero" ;
1       mynumber :cclause ( -- ) cr ." one" ;
2048    mynumber :clause  ( -- ) cr ." 2 KiB" ;
100000. mynumber :2clause ( -- ) cr ." big" ;

\ Execute the clauses:

0       mynumber switch
1       mynumber cswitch
2048    mynumber switch
100000. mynumber 2switch
```

New clauses can be added any time, in any order, with any key.

Clauses created with `:clause` (for 1-cell keys), `:cclause` (for character keys) and `:2clause` (for 2-cell keys) must be executed with `switch`, `cswitch` and `2switch` respectively. The smaller the key type, the less memory used by clauses in headers space (every clause is a definition whose name is the binary string of its key) and the less execution time, though the difference will be unimportant in most cases.

If a new clause is added with a previously used key, the new clause will replace the old one.

There's no default clause: if the a given key is not found, no code is executed and no exception is thrown.

Source file: <src/lib/flow.switch-colon.fs>.

switch]

```
switch] ( a -- ) "switch-bracket"
```

Terminate a switch structure (or the latest additions to it) by marking the end of its linked list. Discard the switch head *a* from the stack.

Origin: SwiftForth.

See also: `[switch`, `[+switch`, `runs`, `run:`.

Source file: <src/lib/flow.bracket-switch.fs>.

switcher

```
switcher ( i*x n a -- j*x )
```

Search the linked list from its head *a* for a match to the value *n*. If a match is found, discard *n* and execute the associated matched *xt*. If no match is found, leave *n* on the stack and execute the default *xt*.

`switcher` is a common factor of `:switch` and `[switch`, two variants of the same control structure.

Origin: SwiftForth.

Source file: <src/lib/flow.bracket-switch.fs>.

synonym

```
synonym ( "newname" "oldname" -- )
```

Create a definition for *newname* with the execution and compilation semantics of *oldname*. *newname* may be the same as *oldname*; when looking up *oldname*, *newname* shall not be found.

Synonyms have the execution token of the old word and, contrary to aliases created by `alias`, they also inherit its attributes `immediate` and `compile-only`.

Origin: Forth-2012 (TOOLS EXT).

Source file: <src/lib/define.synonym.fs>.

system-size

```
system-size ( -- len )
```

len is the size of the system, in bytes, i.e. the size of data/code space.

See also: `+origin`, `system-zone`, `turnkey`, `here`.

Source file: <src/lib/tool.turnkey.fs>.

system-zone

```
system-zone ( -- a len )
```

Return the start address *a* of the system and its length *len*, to be used as parameters for saving the system to tape or disk.

See also: `+origin`, `system-size`, `turnkey`.

Source file: <src/lib/tool.turnkey.fs>.

t

t

```
t ( u "ccc<eol>" -- )
```

A command of `gforth-editor`: Go to line *u* and `insert` *ccc*.

See also: `c`, `a`, `g`, `n`, `p`, `l`.

Source file: <src/lib/prog.editor.gforth.fs>.

t

```
t ( n -- )
```

A command of `specforth-editor`: Type line *n* and save in `pad`.

See also: `b`, `c`, `d`, `e`, `f`, `h`, `i`, `l`, `m`, `n`, `p`, `r`, `s`, `x`.

Source file: <src/lib/prog.editor.specforth.fs>.

tab

```
tab ( -- )
```

`emit` a `'tab'` (character code 6), so that the next character displayed will appear at the next 16-character column.

See also: `tabulate`.

Source file: <src/lib/display.control.fs>.

tabs

```
tabs ( n -- )
```

Emit *n* number of tab characters (character code 6).

See also: `tab`, `'tab'`.

Source file: <src/lib/display.control.fs>.

tabulate

```
tabulate ( -- )
```

Display the appropriate number of spaces to tabulate to the next position, using the value of `/tabulate`.

Note `tabulate` does not uses the "tab" control code, whose behaviour depends on the screen mode (in the default screen mode, it moves the cursor 16 positions to the right). `tabulate` prints spaces and is independent from the screen mode.

See `/tabulate`, `tab`.

Source file: <src/lib/display.control.fs>.

tape-file>

```
tape-file> ( ca1 len1 ca2 len2 -- ) "tape-file-from"
```

Read a tape file *ca1 len1* into a memory region *ca2 len2*.

- When *len1* is zero, it means the filename is unspecified, *ca1* is irrelevant and the first file must be loaded.
- When *ca2* is zero the destination address will be taken from the file header, i.e. the address the file was saved from.
- When *len2* is zero the zone size will be taken from the file header, i.e. the whole length of the file.

WARNING	If *len2* is not zero or the exact length of the file, the ROM routine returns to BASIC with "Tape loading error". This crashes the system, because in Solo Forth the lower screen has no lines, and BASIC cannot display the message.

See also: `>tape-file`, `(tape-file>`, `>file`.

Source file: <src/lib/tape.fs>.

tape-file>display

```
tape-file>display ( ca len -- ) "tape-file-to-display"
```

Read tape file *ca len* into the display memory.

See also: `display>tape-file`, `>tape-file`.

Source file: <src/lib/tape.fs>.

tape-filename

```
tape-filename ( -- ca )
```

Address of the filename in `tape-header`.

See also: `/tape-filename`, `set-tape-filename`, `last-tape-filename`.

Source file: <src/lib/tape.fs>.

tape-filetype

```
tape-filetype ( -- ca )
```

Address of the file type (one byte) in `tape-header`. Its default value is 3 (code file).

See also: `last-tape-filetype`.

Source file: <src/lib/tape.fs>.

tape-header

```
tape-header ( -- a )
```

Address of the tape header, which is used by the ROM routines. Its structure is the following:

Table 37. Structure of a tape header

Offset	Size	Description
+00	byte	filetype (3 for code files)
+01	10 chars	filename, padded with spaces
+11	cell	length
+13	cell	start address
+15	cell	not used for code files

When the first char of the filename is 255, it is regarded as a wildcard which will match any filename. The word `tape-file>` sets the wildcard when the provided filename is empty.

A second tape header, pointed by `last-tape-header`, follows the main one. It is used by the ROM routines while loading. It can be used by the application to know the details of the last tape file that was loaded.

IX addresses the first header, which must contain the data. The second header is used by the system when loading and verifying. Only the "CODE" file type column is relevant to Solo Forth.

Table 38. Detailed structure of both tape headers

First header	Second header	BASIC program	Num DATA	String DATA	CODE	Notes
IX+$00	IX+$11	0	1	2	3	File type
IX+$01	IX+$12	x	x	x	x	F ($FF if filename is null)
IX+$02	IX+$13	x	x	x	x	i
IX+$03	IX+$14	x	x	x	x	l
IX+$04	IX+$15	x	x	x	x	e
IX+$05	IX+$16	x	x	x	x	n
IX+$06	IX+$17	x	x	x	x	a
IX+$07	IX+$18	x	x	x	x	m
IX+$08	IX+$19	x	x	x	x	e
IX+$09	IX+$1A	x	x	x	x	.
IX+$0A	IX+$1B	x	x	x	x	Padding spaces
IX+$0B	IX+$1C	lo	lo	lo	lo	Total…
IX+$0C	IX+$1D	hi	hi	hi	hi	…length of datablock
IX+$0D	IX+$1E	Auto	-	-	Start	Various
IX+$0E	IX+$1F	Start	a-z	a-z	addr	($80 if no autostart).
IX+$0F	IX+$20	lo	-	-	-	Length of program only…
IX+$10	IX+$21	hi	-	-	-	…i.e. without variables

See also: `tape-filename`, `tape-filetype`, `tape-start`, `tape-length`, `any-tape-filename`, `?set-tape-filename`.

Source file: <src/lib/tape.fs>.

tape-length

```
tape-length ( -- a )
```

Address of the file length in `tape-header`.

See also: `last-tape-length`.

Source file: <src/lib/tape.fs>.

tape-start

```
tape-start ( -- a )
```

Address of the file start in `tape-header`.

See also: `last-tape-start`.

Source file: <src/lib/tape.fs>.

terminal

```
terminal ( -- )
```

Select the terminal as output.

See also: `printer`, `printing`, `page`.

Source file: <src/kernel.z80s>.

terminal>source

```
terminal>source ( -- ) "terminal-to-source"
```

Set the terminal as the current source.

Definition:

```
: terminal>source ( -- )
  blk off (source-id off tib #tib @ set-source ;
```

See also: `set-source`, `blk`, `tib`, `#tib`, `(source-id`, `block>source`.

Source file: <src/kernel.z80s>.

text

```
text ( "ccc<eol>" -- )
```

Part of `specforth-editor`: Parse the text string until end of line and store it into `pad` as a counted string, blank-filling the remainder of `pad` to `c/l` characters.

See also: `parse-all`.

Source file: <src/lib/prog.editor.specforth.fs>.

then

```
then
  Compilation: ( C: orig -- )
  Run-time:    ( -- )
```

Resolve the forward reference *orig*, usually left by `if` or `while`.

`then` is an `immediate` and `compile-only` `alias` of `>resolve`.

Definition:

```
' >resolve alias then immediate compile-only
  \ Compilation: ( C: orig -- )
  \ Run-time:    ( -- )
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `else`, `ahead`.

Source file: <src/kernel.z80s>.

thens

```
thens
  Compilation: ( C: cs-mark orig#1 ... orig#n -- )
  Run-time:    ( -- )
```

Compilation: Resolve all forward references *orig#1* … *orig#n* with `then` until `cs-mark` is found.

Run-time: Continue execution.

`thens` is an `immediate` and `compile-only` word.

`thens` is a factor of `endcase` and other control structures, but it’s also the end of the `cond` … `thens` structure. See `cond` for an usage example.

See also: `cs-test`, `andif`, `orif`.

Source file: <src/lib/flow.MISC.fs>.

there

```
there ( a -- )
```

Set *a* as the address of the data-space pointer. A non-standard counterpart of `here`.

Source file: <src/lib/compilation.fs>.

thiscase

```
thiscase ( x -- x x )
```

Part of a `thiscase` structure.

Usage example:

```
: say0      ( -- ) ." nul" ;
: say1      ( -- ) ." unu" ;
: say2      ( -- ) ." du" ;
: say-other ( -- ) ." alia" ;

: test ( x -- )
  thiscase  0 = ifcase  say0  exitcase
  thiscase  1 = ifcase  say1  exitcase
  thiscase  2 = ifcase  say2  exitcase
  othercase say-other ;
```

See also: `ifcase`, `exitcase`, `othercase`, `case`.

Source file: <src/lib/flow.thiscase.fs>.

throw

```
throw ( k*x n -- k*x | i*x n )
```

If *n* is zero, drop it and continue. Otherwise, pop the topmost exception frame from the exception stack, along with everything on the return stack above that frame. Then restore the input source specification in use before the corresponding `catch` and adjust the depths of all stacks so that they are the same as the depths saved in the exception frame (*i* is the same number as the *i* in the input arguments to the corresponding `catch`), put *n* on top of the data stack, and transfer control to a point just after the `catch` that pushed that exception frame.

If the top of the stack is non-zero and there is no exception frame on the exception stack, i.e. the content of `catcher` is zero, `error` is executed with *n* on top of the stack.

Definition:

```
: throw ( k*x n -- k*x | i*x n )
  ?dup 0exit
  catcher @ ?dup 0= \ no catcher?
  if error then     \ ``error`` does not return
  rp!               \ restore previous return stack
  r> catcher !      ( n ) \ restore previous catcher
  r> swap >r        ( saved-SP ) ( R: n )
  sp! drop r>       ( n ) \ restore stack
  unnest-source ;   \ restore previous source specification
```

Origin: Forth-94 (EXCEPTION), Forth-2012 (EXCEPTION).

Source file: <src/kernel.z80s>.

thru

```
thru ( block1 block2 -- )
```

Load consecutively the blocks from *block1* through *block2*.

Origin: Forth-79 (Reference Word Set), Forth-83 (Controlled Reference Words), Forth-94 (BLOCK EXT), Forth-2012 (BLOCK EXT).

See also: `load`, `+thru`.

Source file: <src/lib/blocks.fs>.

thru-index-need

```
thru-index-need ( "name" -- )
```

If word *name* is found in the current search order, do nothing. Otherwise search the index word list for it. If found, execute it, causing its associated block be loaded. If not found, `throw` an exception #-277 ("needed, but not indexed").

`thru-index-need` is an alternative action of the deferred word `need` (see `defer`).

Source file: <src/lib/blocks.indexer.thru.fs>.

thru-index-needed

```
thru-index-needed ( ca len -- )
```

If word *ca len* is found in the current search order, do nothing. Otherwise search the index word list for it. If found, execute it, causing its associated block be loaded. If not found, `throw` an exception #-277 ("needed, but not indexed").

`thru-index-needed` is an alternative action of the deferred word `needed` (see `defer`).

Source file: <src/lib/blocks.indexer.thru.fs>.

thru-index-reneed

```
thru-index-reneed ( "name" -- )
```

Search the index word list for word "name". If found, execute it, causing its associated block be loaded. If not found, `throw` an exception #-277 ("needed, but not indexed").

`thru-index-reneed` is an alternative action of the deferred word `reneed` (see `defer`).

Source file: <src/lib/blocks.indexer.thru.fs>.

thru-index-reneeded

```
thru-index-reneeded ( ca len-- )
```

Search the index word list for word *ca len*. If found, load the block it's associated to. If not found, `throw` an exception #-277 ("needed, but not indexed").

`thru-index-reneeded` is an alternative action of the deferred word `reneeded` (see `defer`).

Source file: <src/lib/blocks.indexer.thru.fs>.

tib

```
tib ( -- a ) "t-i-b"
```

A `constant`. *a* is the address of the terminal input buffer.

Origin: Forth-83 (Required Word Set), Forth-94 (CORE EXT, obsolescent).

See also: `/tib`, `#tib`.

Source file: <src/kernel.z80s>.

ticks

```
ticks ( -- n )
```

Return the current count of clock ticks *n*, which is updated by the OS.

NOTE `ticks` returns the low 16 bits of the OS frames counter, which is increased by the OS interrupts routine every 20th ms. The counter is actually a 24-bit value, which can be fetched by `dticks`.

Origin: Comus.

See also: `set-ticks`, `reset-ticks`, `ticks/second`, `ticks>seconds`, `ms>ticks`, `os-frames`, `bench{`.

Source file: <src/lib/time.fs>.

ticks-pause

```
ticks-pause ( u -- )
```

Stop execution during at least *u* clock ticks.

See also: `?ticks-pause`, `basic-pause`, `seconds`, `ms`, `ticks/second`.

Source file: <src/lib/time.fs>.

ticks/second

```
ticks/second ( -- n ) "ticks-slash-second"
```

Return the number *n* of clock ticks per second.

See also: `ms/tick`, `dticks>seconds`, `dticks>cs`, `dticks>ms`, `ticks`.

Source file: <src/lib/time.fs>.

ticks>cs

```
ticks>cs ( n1 -- n2 ) "ticks-to-cs"
```

Convert clock `ticks` *n1* to centiseconds *n2*.

See also: `dticks>cs`, `ticks>seconds`, `ticks>ms`, `ticks/second`.

Source file: <src/lib/time.fs>.

ticks>ms

```
ticks>ms ( n1 -- n2 ) "ticks-to-ms"
```

Convert clock ticks *n1* to milliseconds *n2*.

See also: `ms>ticks`, `dticks>ms`, `ticks>seconds`, `ticks>cs`, `ticks/second`, `ticks`.

Source file: <src/lib/time.fs>.

ticks>seconds

```
ticks>seconds ( n1 -- n2 ) "ticks-to-seconds"
```

Convert clock ticks *n1* to seconds *n2*.

See also: `dticks>seconds`, `ticks>cs`, `ticks>ms`, `ticks/second`, `ticks`.

Source file: <src/lib/time.fs>.

till

```
till ( "ccc<eol>" -- )
```

A command of `specforth-editor`: Delete on the cursor line from the cursor till the end of string *ccc*.

Source file: <src/lib/prog.editor.specforth.fs>.

time&date

```
time&date ( -- second minute hour day month year ) "time-and-date"
```

Return the current time and date: second, minute, hour, day, month and year.

Origin: Forth-94 (FACILITY EXT), Forth-201 (FACILITY EXT).

See also: `get-time`, `get-date`, `set-time`, `set-date`, `.time&date`.

Source file: <src/lib/time.fs>.

timer

```
timer ( u -- )
```

For the time *u* in `ticks` display the elapsed time since then, also in `ticks`.

Origin: Comus.

See also: `dtimer`, `elapsed`, `ticks>seconds`, `ticks>cs`, `ticks>ms`.

Source file: <src/lib/time.fs>.

times

```
times ( u -- )
```

Repeat the next compiled instruction *u* times. If *u* is zero, continue executing the following instruction.

`times` is useful to implement complicated math operations, like shifts, multiply, divide and square root, from appropriate math step instructions. It is also useful in repeating auto-indexing memory instructions.

This structure is not nestable.

Usage example:

```
: blink ( -- ) 7 0 ?do  i border  loop  0 border ;
: blinking ( -- ) 100 times blink  ." Done" cr ;
```

Origin: cmForth's `repeats`.

See also: `dtimes`, `executions`, `for`, `?do`.

Source file: <src/lib/flow.times.fs>.

tnegate

```
tnegate ( t1 -- t2 ) "t-negate"
```

t2 is the negation of *t1*.

Source file: <src/lib/math.operators.3-cell.fs>.

to

```
to
  Interpretation: ( i*x "name" -- )
  Compilation:    ( "name" -- )
  Run-time:       ( i*x -- )
```

`to` is an `immediate` word.

Interpretation:

Parse *name*, which is a word created by `cvalue`, `value` or `2value`, and make *i*x* its value.

Compilation:

Parse *name*, which is a word created by `cvalue`, `value` or `2value`, and append the execution execution semantics given below to the current definition.

Run-time:

Make *i*x* the value of *name*.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `!>`, `c!>`, `2!>`, `toval`, `ctoval`, `2toval`.

Source file: <src/lib/data.value.fs>.

toarg

```
toarg ( -- ) "to-arg"
```

Set the store action for the next local variable created by `arguments`.

Loading `toarg` makes `@` the default action of `arguments` locals, which is hold in `arg-default-action`.

See also: `+toarg`.

Source file: <src/lib/locals.arguments.fs>.

toggle-capslock

```
toggle-capslock ( -- )
```

Toggle capslock.

See also: `set-capslock`, `unset-capslock`, `capslock?`, `capslock`, `ctoggle`.

Source file: <src/lib/keyboard.caps_lock.fs>.

toggle-pixel

```
toggle-pixel ( gx gy -- )
```

Toggle a pixel without changing its attribute on the screen or the current graphic coordinates. *gx* is 0..255; *gy* is 0..191.

See also: `set-pixel`, `reset-pixel`, `toggle-pixel176`, `set-pixel176`, `reset-pixel176`, `plot`, `plot176`.

Source file: <src/lib/graphics.pixels.fs>.

toggle-pixel176

```
toggle-pixel176 ( gx gy -- ) "toggle-pixel-176"
```

Toggle a pixel without changing its attribute on the screen or the current graphic coordinates, and using only the top 176 pixel rows of the screen (the lower 16 pixel rows are not used). *gx* is 0..255; *gy* is 0..175.

See also: `toggle-pixel`, `set-pixel`, `reset-pixel`, `set-pixel176`, `reset-pixel176`, `plot`, `plot176`.

Source file: <src/lib/graphics.pixels.fs>.

top

```
top ( -- )
```

Position the editing cursor at the top of the block, by setting `r#` to zero.

`top` is used by `specforth-editor` and `gforth-editor`.

Source file: <src/lib/prog.editor.COMMON.fs>.

toval

```
toval ( -- ) "to-val"
```

Change the default behaviour of words created by `val`: make them store a new value instead of returning its actual one.

`toval` and `val` are a non-parsing alternative to the standard `to` and `value`.

See also: `ctoval`, `2toval`.

Source file: <src/lib/data.val.fs>.

tr-dos

```
tr-dos ( -- ) "t-r-dos"
```

An alias of `noop` that is defined only in the TR-DOS version of Solo Forth. Its goal is to check the DOS a program is running on, using `defined` or `[defined]`.

`tr-dos` is an `immediate` word.

See also: `dos`, `g+dos`, `+3dos`.

Source file: <src/kernel.z80s>.

tracks/cat

```
tracks/cat ( -- b ) "tracks-slash-cat"
```

A `cconstant`, *b* is the number of tracks used by the disk catalogue (only one side of the disk is used for the catalogue).

See `tracks/disk`, `sectors/disk`, `b/sector`, `sectors-used`, `drive-unused`.

Source file: <src/lib/dos.gplusdos.fs>.

tracks/disk

```
tracks/disk ( -- b ) "tracks-slash-disk"
```

A `cconstant`. *b* is the number of tracks of a disk.

See also: `tracks/cat`, `sectors/disk`, `max-disk-capacity`, `b/sector`, `sectors-used`, `drive-unused`.

Source file: <src/lib/dos.gplusdos.fs>.

trail

```
trail ( -- nt )
```

Leave the *nt* of the topmost word in the first word list of the search order.

See also: `set-order`, `context`.

Source file: <src/lib/word_lists.fs>.

transfer-block

```
transfer-block ( u -- )
```

The block-level disk read-write linkage. Transfer block *u* to or from disk. The read or write mode must be previously set by `write-mode` or `read-mode`.

Definition:

```
: transfer-block ( u -- )
  >drive-block sectors/block * dup
     block-sector#>dos buffer-data
     transfer-sector throw
  1+ block-sector#>dos [ buffer-data b/sector + ] literal
     transfer-sector throw ;
```

See also: `transfer-sector`, `block-sector#>dos`, `>drive-block`.

Source file: <src/kernel.gplusdos.z80s>.

transfer-sector

```
transfer-sector ( x a -- ior )
```

The sector-level disk read-write linkage. Transfer sector *x* to or from the disk in the current drive. The read or write mode must be previously set by `write-mode` or `read-mode`.

- *x* = sector to be read or written
    - high byte = track 0..79, +128 if side 1
    - low byte = sector 1..10
- *a* = source or destination address

See also: `block-sector#>dos`, `transfer-block`.

Source file: <src/kernel.gplusdos.z80s>.

transient

```
transient ( u1 u2 -- )
```

Start transient code, reserving *u1* bytes of headers space for it, which will be allocated at the top of the far memory, and *u2* bytes of data space for it, which will be allocated at the top of the main memory. Therefore the memory used by the transient code must be known in advance.

The inner operation is: Save the current values of `dp`, `np` `current-latest`, `last-wordlist`, `limit` and `farlimit`; then reserve data and headers space as said and update `limit` and `farlimit` accordingly.

`transient` must be used before compiling the transient code.

Usage example:

```
2025 1700 transient

need assembler

end-transient

\ ...use assembler here...

forget-transient
```

The values of `limit` and `farlimit` must be preserved between `transient` and `end-transient`, because `forget-transient` restores them to their previous state, before `transient`.

Source file: <src/lib/modules.transient.fs>.

translate-char

```
translate-char ( c1 -- c1 | c2 )
```

Translate character *c1* using the current keyboard decoding table, pointed by `key-translation-table`.

Source file: <src/kernel.z80s>.

trim

```
trim ( ca1 len1 -- ca2 len2 )
```

Remove leading and trailing spaces from a string *ca len1*, returning the result string *ca2 len2*.

See also: `-leading`, `-trailing`.

Source file: <src/lib/strings.MISC.fs>.

true

```
true ( -- true )
```

Return a true flag, a single-cell value with all its bits set (equivalent to `-1`).

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `false`.

Source file: <src/kernel.z80s>.

tstp,

```
tstp, ( regp -- ) "t-s-t-p-comma"
```

Compile the Z80 `assembler` instructions required to test the register pair *regp* for zero. Register "A" is modified.

Example: `b tstp,` compiles the Z80 instructions `LD A,B` and `OR C`.

See also: `ldp,`, `subp,`, `cp#,`, `cp,`, `or,`, `ld,`.

Source file: <src/lib/assembler.fs>.

ttester

```
ttester ( -- )
```

Do nothing. `ttester` is used just for doing `need ttester`, loading `t{`, `->`, `}t` and other words, which are used by `hayes-test` and `forth2012-test-suite`..

Usage example:

```
T{ 1 2 3 swap -> 1 3 2 }T  ok
T{ 1 2 3 swap -> 1 2 2 }T
Incorrect result:
T{ 1 2 3 swap -> 1 2 2 }T ok
T{ 1 2 3 swap -> 1 2 }T
Wrong number of results:
T{ 1 2 3 swap -> 1 2 }T ok
```

See also: `hayes-tester`.

Source file: <src/lib/meta.tester.ttester.fs>.

tuck

```
tuck ( x1 x2 -- x2 x1 x2 )
```

Copy the first (top) stack item below the second stack item.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `dup`, `over`, `nip`, `nup`.

Source file: <src/kernel.z80s>.

turnkey

```
turnkey ( xt -- a len )
```

Prepare the system in order to save a copy of its current state. Return its start address *a* and length *len*, to be used as parameters for saving the system to disk. The saved copy will execute *xt* after the ordinary boot process.

Usage example:

```
' my-game turnkey s" my-game" >tape-file
```

WARNING	This word is experimental. See the source code for details.
WARNING	Since name fields are kept in a memory bank, the best way to save a modified Forth system is creating a snapshot with a ZX Spectrum emulator, or with the equivalent feature provided by certain interfaces or modern ZX Spectrum clones. `turnkey` and its related words are meant to save a Forth program that does not need to search the dictionary or use data already stored in paged memory.

See also: `boot`, `extend`, `system-zone`, `cold`.

Source file: <src/lib/tool.turnkey.fs>.

type

```
type ( ca len -- )
```

If *len* is greater than zero, display the character string *ca len*.

: type (ca len —) bounds ?do i c@ emit loop ;

See also: `type-udg`.

Source file: <src/kernel.z80s>.

type-ascii

```
type-ascii ( ca len -- )
```

If *len* is greater than zero, display the string *ca len*, using `emit-ascii` to make sure the characters are graphic ASCII characters.

See also: `type`, `fartype-ascii`.

Source file: <src/lib/display.type.fs>.

type-center-field

```
type-center-field ( ca1 len1 len2 -- )
```

If *len1* is greater than zero, display the character string *ca1 len1* at the center of a field of *len2* characters.

See also: `type-center-field-fit`, `type-center-field-crop`, `drop-type`, `type-left-field`, `type-right-field`, `gigatype-title`.

Source file: <src/lib/display.type.fs>.

type-center-field-crop

```
type-center-field-crop ( ca1 len1 len2 -- )
```

If *len1* is greater than zero, display the character string *ca1 len1* at the center of a field of *len2* characters, which is shorter than the string.

See also: `type-center-field-fit`, `type-center-field`.

Source file: <src/lib/display.type.fs>.

type-center-field-fit

```
type-center-field-fit ( ca1 len1 len2 -- )
```

If *len1* is greater than zero, display the character string *ca1 len1* at the center of a field of *len2* characters, which is longer than the string.

See also: `type-center-field-crop`, `type-center-field`.

Source file: <src/lib/display.type.fs>.

type-left-field

```
type-left-field ( ca1 len1 len2 -- )
```

If *len1* is greater than zero, display the character string *ca1 len1* at the left of a field of *len2* characters.

See also: `padding-spaces`, `type-right-field`, `type-center-field`.

Source file: <src/lib/display.type.fs>.

type-right-field

```
type-right-field ( ca1 len1 len2 -- )
```

If *len1* is greater than zero, display the character string *ca1 len1* at the right of a field of *len2* characters.

See also: `type-right-field-fit`, `type-right-field-crop`, `drop-type`, `type-left-field`, `type-center-field`.

Source file: <src/lib/display.type.fs>.

type-right-field-crop

```
type-right-field-crop ( ca1 len1 len2 -- )
```

Type string *ca1 len1* at the right of a field of *len2* characters, which is shorter than the string.

See also: `type-right-field`, `type-right-field-fit`.

Source file: <src/lib/display.type.fs>.

type-right-field-fit

```
type-right-field-fit ( ca1 len1 len2 -- )
```

Type string *ca1 len1* at the right of a field of *len2* characters, which is longer than the string.

See also: `type-right-field`, `type-right-field-crop`.

Source file: <src/lib/display.type.fs>.

type-udg

```
type-udg ( ca len -- ) "type-u-d-g"
```

If *len* is greater than zero, display the UDG character string *ca len*. All characters of the string are printed with `emit-udg`.

See also: `type`.

Source file: <src/lib/graphics.udg.fs>.

t{

```
t{ ( -- )
```

Part of `ttester`: Start a test.

See also: `->`, `}t`.

Source file: <src/lib/meta.tester.ttester.fs>.

u

u%

```
u% ( u1 u2 -- u3 ) "u-per-cent"
```

u1 is percentage *u3* of *u2*.

See also: `%`, `um*`, `um/mod`.

Source file: <src/lib/math.operators.1-cell.fs>.

u.

```
u. ( u -- ) "u-dot"
```

Display *u* in free field format.

Definition:

```
: u. ( u -- ) s>d ud. ;
```

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `u.r`, `ud.`, `.`.

Source file: <src/kernel.z80s>.

u.r

```
u.r ( u n -- ) "u-dot-r"
```

Display *u* right aligned in a field *n* characters wide. If the number of characters required to display *u* is greater than *n*, all digits are displayed with no leading spaces in a field as wide as necessary.

Origin: Forth-79 (Reference Word Set)[7], Forth-83 (Controlled Reference Word Set)[8], Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `ud.r`, `.r`, `u.`.

Source file: <src/lib/display.numbers.fs>.

u.s

```
u.s ( -- )
```

Display, using `u.`, the values currently on the data stack.

See also: `.s`, `depth`, `.depth`.

Source file: <src/lib/tool.list.stack.fs>.

u<

```
u< ( u1 u2 -- f ) "u-less-than"
```

f is true if and only if *u1* is less than *u2*.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `<`, `u>`, `0<` ,`umin`.

Source file: <src/kernel.z80s>.

u<=

```
u<= ( u1 u2 -- f ) "u-less-or-equal"
```

f is `true` if and only if *u1* is less than or equal to *u2*.

See also: `u>=`, `<=`, `0<=`.

Source file: <src/lib/math.operators.1-cell.fs>.

u>

```
u> ( u1 u2 -- f ) "u-greater-than"
```

f is true if and only if *u1* is greater than *u2*.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `>`, `u<`, `0>`, `umax`.

Source file: <src/kernel.z80s>.

u>=

```
u>= ( u1 u2 -- f ) "u-greater-or-equal"
```

f is `true` if and only if *u1* is greater than or equal to *u2*.

See also: `u<=`, `>=`, `0>=`.

Source file: <src/lib/math.operators.1-cell.fs>.

u>str

```
u>str ( u -- ca len ) "u-to-s-t-r"
```

Convert *u* to string *ca len*.

See also: `n>str`, `ud>str`, `d>str`, `char>string`.

Source file: <src/lib/strings.MISC.fs>.

u>ud

```
u>ud ( u -- ud )
```

Extend a single unsigned number *u* to form a double unsigned number *ud*. `u>ud` is just an alias of `0`.

See also: `s>d`.

Source file: <src/lib/math.operators.1-cell.fs>.

uallot

```
uallot ( n -- ) "u-allot"
```

If *n* is greater than zero, reserve *n* bytes of user data space. If *n* is less than zero, release *n* bytes of user data space. If *n* is zero, leave the user data-space pointer unchanged. An exception is thrown if the user-data pointer is out of bounds after the operation.

See also: `udp`, `ucreate`, `?user`, `user`, `2user`.

Source file: <src/lib/data.user.fs>.

ucreate

```
ucreate ( "name" -- ) "u-create"
```

Parse *name*. *Create a header _name* which points to the first available offset within the user area. When *name* is later executed, its absolute user area storage address is placed on the stack. No user space is allocated.

See also: `uallot`, `user`, `2user`, `?user`.

Source file: <src/lib/data.user.fs>.

ud*

```
ud* ( ud1 ud2 -- ud3 ) "u-d-star" "u-d-star"
```

Multiply *ud1* by *ud2* giving the product *ud3*.

See also: `d*`, `um*`, `m*`, `*`.

Source file: <src/lib/math.operators.2-cell.fs>.

ud.

```
ud. ( ud -- ) "u-d-dot"
```

Display an usigned double number *ud*.

See also: `ud.r`, `d.`, `u.`.

Source file: <src/lib/display.numbers.fs>.

ud.r

```
ud.r ( ud n -- ) "u-d-dot-r"
```

Display *ud* right aligned in a field *n* characters wide. If the number of characters required to display *ud* is greater than *n*, all digits are displayed with no leading spaces in a field as wide as necessary.

See also: `u.r`, `d.`, `ud.`.

Source file: <src/lib/display.numbers.fs>.

ud/mod

```
ud/mod ( ud1 u2 -- u3 ud4 ) "u-d-slash-mod"
```

An unsigned mixed magnitude math operation which leaves a double quotient *ud4* and remainder *u3*, from a double dividend *ud1* and single divisor *u2*.

Definition:

```
: ud/mod ( ud1 u1 -- urem udquot )
  >r 0 r@ um/mod -rot r> um/mod rot ;
```

Origin: fig-Forth’s `m/mod`, Gforth, Z88 CamelForth.

Source file: <src/kernel.z80s>.

ud>str

```
ud>str ( ud -- ca len ) "u-d-to-s-t-r"
```

Convert *ud* to string *ca len*.

See also: `u>str`, `d>str`, `char>string`.

Source file: <src/lib/strings.MISC.fs>.

udg!

```
udg! ( b0..b7 c -- ) "u-d-g-store"
```

Store the 8-byte bitmap *b0..b7* into UDG *c* (0..255) of the UDG font pointed by `os-udg`. *b0* is the first (top) scan. *b7* is the last (bottom) scan.

See also: `udg:`, `udg>`.

Source file: <src/lib/graphics.udg.fs>.

udg-at-xy-display

```
udg-at-xy-display ( col row c -- ) "u-d-g-at-x-y-display"
```

Display UDG *c* (0..255) at cursor coordinates *col row*. `udg-at-xy-display` is much faster than a combination of `at-xy` and `emit-udg`, because no ROM routine is used, the cursor coordinates are not updated and the screen attributtes are not changed (only the character bitmap is displayed).

See also: `at-xy-display-udg`.

Source file: <src/lib/graphics.udg.fs>.

udg-blank

```
udg-blank  ( -- ca ) "u-d-g-blank"
```

A `cvariable`. *ca* is the address of a byte containing the character used by `udg-group`, `udg-block`, `,udg-block` and others as a grid blank. By default it's '.'.

See also: `udg-dot`, `udg-scan>binary`.

Source file: <src/lib/graphics.udg.fs>.

udg-block

```
udg-block ( width height c "name..." -- ) "u-d-g-block"
```

Parse a UDG block, and store it from UDG character *c* (0..255). *width* and *height* are in characters. The maximum *width* is 7 (imposed by the size of Forth source blocks). *height* has no maximum, as the UDG block can ocuppy more than one Forth block (provided the Forth block has no index line, i.e. `load-program` is used to load the source).

The scans can be formed by binary digits, by the characters hold in `udg-blank` and `udg-dot`, or any combination of both notations.

Usage example:

```
0 cconstant mass-udg
2 cconstant mass-height
5 cconstant mass-width

mass-width mass-height mass-udg udg-block

..XXXX....XXXX....XXXX....XXXX....XXXX..
.XXXXXX..XXXXXX..XXXXXX..XXXXXX..X.XXXX.
XXXXXXXXXXXXXXXXXXXXXXXXX.XXXXXXX.XXXXXX
XXXXXXXXXXXXXXXXX.XXXXXXX.XXXXXXXXXXXXXX
XXXXXXXXX.XXXXXXX.XXXXXXXXXXXXXXXXXXXXXX
XX..XXXXXX.XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
.XXXXXX..XXXXXX..XXXXXX..XXXXXX..XXXXXX.
..XXXX....XXXX....XXXX....XXXX....XXXX..
..XXXX....XXXX....XXXX....XXXX....XXXX..
.XXXXXX..XXXXXX..XXXXXX..XXXXXX..X.XXXX.
XXXXXXXXXXXXXXXXXXXXXXXXX.XXXXXXX.XXXXXX
XXXXXXXXXXXXXXXXX.XXXXXXX.XXXXXXXXXXXXXX
XXXXXXXXX.XXXXXXX.XXXXXXXXXXXXXXXXXXXXXX
XX..XXXXXX.XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
.XXXXXX..XXXXXX..XXXXXX..XXXXXX..XXXXXX.
..XXXX....XXXX....XXXX....XXXX....XXXX..

: .mass ( -- )
  mass-height 0 ?do
    mass-width 0 ?do
      i j mass-width * + mass-udg + emit-udg
    loop cr
  loop ;

cr .mass
```

See also: `,udg-block`, `csprite`, `udg-group`.

Source file: <src/lib/graphics.udg.fs>.

udg-dot

```
udg-dot  ( -- ca ) "u-d-g-dot"
```

A `cvariable`. *ca* is the address of a byte containing the character used by `udg-group`, `udg-block` `,udg-block` and others as a grid blank. By default it’s 'X'.

See also: `udg-blank`, `udg-scan>binary`.

Source file: <src/lib/graphics.udg.fs>.

udg-group

```
udg-group ( width height c -- ) "u-d-g-group"
```

Parse a group of UDG definitions organized in *width* columns and *height* rows, and store them starting from UDG character *c* (0..255). The maximum *width* is 7 (imposed by the size of Forth source blocks). *height* has no maximum, as the UDG block can ocuppy more than one Forth block (provided the Forth block has no index line, i.e. `load-program` is used to load the source).

The UDG scans can be formed by binary digits, by the characters hold in `udg-blank` and `udg-dot`, or any combination of both notations. The UDG scans must be separated with at least one space.

Usage example:

```
5 1 140 udg-group

..XXXX.. ..XXXX.. ..XXXX.. ..XXXX.. ..XXXX..
.XXXXXX. .XXXXXX. .XXXXXX. .XXXXXX. .X.XXXX.
XXXXXXXX XXXXXXXX XXXXXXXX X.XXXXXX X.XXXXXX
XXXXXXXX XXXXXXXX X.XXXXXX X.XXXXXX XXXXXXXX
XXXXXXXX X.XXXXXX X.XXXXXX XXXXXXXX XXXXXXXX
XX..XXXX XX.XXXXX XXXXXXXX XXXXXXXX XXXXXXXX
.XXXXXX. .XXXXXX. .XXXXXX. .XXXXXX. .XXXXXX.
..XXXX.. ..XXXX.. ..XXXX.. ..XXXX.. ..XXXX..
```

See also: `udg-block`.

Source file: <src/lib/graphics.udg.fs>.

udg-ocr

```
udg-ocr ( n -- ) "u-d-g-o-c-r"
```

Set `ocr` to work with the first *n* chars of the current UDG set, pointed by `os-udg`.

See also: `ocr-font`, `ocr-first`, `ocr-chars`, `ascii-ocr`, `set-udg`.

Source file: <src/lib/graphics.ocr.fs>.

udg-scan>binary

```
udg-scan>binary ( ca len -- ) "u-d-g-scan-to-binary"
```

Convert the characters `udg-blank` and `udg-dot` found in UDG scan string *ca len* to '0' and '1' respectively.

See also: `udg-scan>number?`. `udg-group`, `udg-block`, `,udg-block`.

Source file: <src/lib/graphics.udg.fs>.

udg-scan>number

```
udg-scan>number ( ca len -- n ) "u-d-g-scan-to-number"
```

If UDG scan string *ca len*, after being processed by `udg-scan>binary`, is a valid binary number, return the result *n*. Otherwise `throw` exception #-290 (invalid UDG scan).

See also: `udg-scan>number?`, `udg-dot`, `udg-blank`.

Source file: <src/lib/graphics.udg.fs>.

udg-scan>number?

```
udg-scan>number? ( ca len -- n true | false ) "u-d-g-scan-to-number-question"
```

Is UDG scan string *ca len* a valid binary number? If so, return *n* and `true`; else return `false`. The string is processed by `udg-scan>binary` first.

See also: `udg-scan>number`, `udg-dot`, `udg-blank`.

Source file: <src/lib/graphics.udg.fs>.

udg-width

```
udg-width ( -- b ) "u-d-g-width"
```

b is the width of a UDG (User Defined Graphic), in pixels.

See also: `/udg`, `udg!`.

Source file: <src/lib/graphics.udg.fs>.

udg:

```
udg: ( b0..b7 c "name" -- ) "u-d-g-colon"
```

Create a `cconstant` *name* for UDG char *c* (0..255) and store the 8-byte bitmap *b0..b7* into that UDG char. *b0* is the first (top) scan. *b7* is the last (bottom) scan.

See also: `udg!`, `udg>`.

Source file: <src/lib/graphics.udg.fs>.

udg>

```
udg> ( c -- a ) "u-d-g-to"
```

Convert UDG number *n* (0..255) to the address *a* of its bitmap, pointed by `os-udg`.

See also: `udg!`, `udg:`, `/udg*`, `get-udg`.

Source file: <src/lib/graphics.udg.fs>.

udp

```
udp ( -- a ) "u-d-p"
```

A `user` variable. *a* is the address of a cell containing an offset from the start of the current user area to the free space in it.

Source file: <src/kernel.z80s>.

ufia

```
ufia ( -- a ) "u-f-i-a"
```

Return address *a* of a buffer currently used as UFIA (User File Information Area), a 24-byte structure which describes a file.

Solo Forth words use `ufia` for G+DOS calls. G+DOS uses its own buffers `ufia1` and `ufia2` for internal operations.

`ufia` is a `constant`. Its default value is `ufia0`, and can be set with `default-ufia`. The value is changed with `!>` by some words.

Table 39. UFIA structure

Offset	Bytes	Meaning
+0	1	Drive number (1, 2 or '*' ($2A) for current)
+1	1	Directory entry number
+2	1	Stream number
+3	1	Device density type ('d'=DD, 'D'=SD)
+4	1	Directory description (see next table)
+5	10	File name (padded with spaces)

Offset	Bytes	Meaning
+15	1	File type
+16	2	Length of file
+18	2	Start address
+20	2	Length of a BASIC program
+22	2	Autostart line of a BASIC program

Table 40. Directory description codes

Code	Type	CAT string	ROM-ID
0	ERASED (free entry)	n/a	n/a
1	BASIC	BAS	0
2	NUMBER ARRAY	D.ARRAY	1
3	STRING ARRAY	$.ARRAY	2
4	CODE	CDE	3
5	48K SNAPSHOT	SNP 48k	n/a
6	MICRODRIVE	MD.FILE	n/a
7	SCREEN$	SCREEN$	n/a
8	SPECIAL	SPECIAL	n/a
9	128K SNAPSHOT	SNP 128k	n/a
10	OPENTYPE	OPENTYPE	n/a
11	EXECUTE	EXECUTE	n/a

NOTE The original UFIA field names are used, except `device`, whose original name is "lstr1":

See also: `/ufia`, `>ufiax`, `dstr1`, `fstr1`, `sstr1`, `device`, `nstr1`, `nstr2`, `hd00`, `hd0b`, `hd0d`, `hd0f`, `hd11`.

Source file: <src/lib/dos.gplusdos.fs>.

ufia0

```
ufia0 ( -- a ) "u-f-i-a-zero"
```

Return address *a* of a buffer used as UFIA (User File Information Area), a 24-byte structure which describes a file. `ufia0` is the default value of `ufia`.

See also: `default-ufia`.

Source file: <src/lib/dos.gplusdos.fs>.

ufia1

```
ufia1 ( -- a ) "u-f-i-a-1"
```

a is the address of G+DOS UFIA1 (in the Plus D memory). A UFIA (User File Information Area) is a 24-byte structure which describes a file. See `ufia` for a detailed description.

See also: `/ufia`, `ufia2`, `dfca`.

Source file: <src/lib/dos.gplusdos.fs>.

ufia2

```
ufia2 ( -- a ) "u-f-i-a-2"
```

A `constant`. *a* is the address of G+DOS UFIA2 (in the Plus D memory). A UFIA (User File Information Area) is a 24-byte structure which describes a file. See `ufia` for a detailed description.

See also: `/ufia`, `ufia1`, `dfca`.

Source file: <src/lib/dos.gplusdos.fs>.

um*

```
um* ( u1 u2 -- ud ) "u-m-star"
```

Multiply *u1* by *u2*, giving the unsigned double-cell product *ud*. All values and arithmetic are unsigned.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `m*`, `*`, `d*`.

Source file: <src/kernel.z80s>.

um/mod

```
um/mod ( ud u1 -- u2 u3 ) "u-m-slash-mod"
```

Divide *ud* by *u1*, giving the quotient *u3* and the remainder *u2*. All values and arithmetic are unsigned.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `/mod`, `du/mod`, `mod`, `*/mod`.

Source file: <src/kernel.z80s>.

umax

```
umax ( u1 u2 -- u1 | u2 ) "u-max"
```

u3 is the greater of *u1* and *u2*.

See also: `umin`, `max`, `dmax`, `u>`.

Source file: <src/kernel.z80s>.

umin

```
umin ( u1 u2 -- u1 | u2 ) "u-min"
```

u3 is the lesser of *u1* and *u2*.

See also: `umax`, `min`, `dmin`, `u<`.

Source file: <src/kernel.z80s>.

unbright-mask

```
unbright-mask ( -- b )
```

A `cconstant`. *b* is the inverted bitmask of the bit used to indicate the bright status in an attribute byte.

See also: `bright-mask`, `brighty`, `set-bright`, `attr!`. `unflash-mask`, `unpaper-mask`, `unink-mask`.

Source file: <src/lib/display.attributes.fs>.

uncolored-circle-pixel

```
uncolored-circle-pixel ( -- a )
```

a is the address of a subroutine that `circle` can use to draw its pixels. This routine sets a pixel without changing its color attributes on the screen (like `set-pixel`). Therefore it's faster than its alternative `colored-circle-pixel` (0.6 its execution speed).

`set-circle-pixel` sets the routine used by `circle`. See the requirements of such routine in the documentation of `circle-pixel`.

Source file: <src/lib/graphics.circle.fs>.

undefined?

```
undefined? ( ca len -- f ) "undefined-question"
```

Find name *ca len*. If the definition is not found after searching the active search order, return true, else return false.

Definition:

```
: undefined? ( ca len -- f ) find-name 0= ;
```

See also: `defined?`, `defined`, `find-name`.

Source file: <src/kernel.z80s>.

under+

```
under+ ( n1|u1 x n2|u2 -- n3|u3 x ) "under-plus"
```

Add *n2|u2* to *n1|u1*, giving the sum *n3|u3*.

`under+` is written in Z80. Its definition in Forth is the following:

```
: under+ ( n1|u1 x n2|u2 -- n3|u3 x ) rot + swap ;
```

Origin: Comus.

See also: `+under` ,`+`.

Source file: <src/lib/math.operators.1-cell.fs>.

undo

```
undo ( `name`-- )
```

Parse *name*, which is the name of a word created by `doer`, and make it do nothing.

See also: `make`, `;and`.

Source file: <src/lib/flow.doer.fs>.

unescape

```
unescape ( ca1 len1 ca2 -- ca2 len2 )
```

Replace each "%" character in the input string *ca1 len1* by two "%" characters. The output is represented by *ca2 len2*. The buffer at *ca2* shall be big enough to hold the unescaped string.

If you pass a string through `unescape` and then `substitute`, you get the original string.

Origin: Forth-2012 (STRING EXT).

See also: `replaces`.

Source file: <src/lib/strings.MISC.fs>.

unflash-mask

```
unflash-mask ( -- b )
```

A `cconstant`. *b* is the inverted bitmask of the bit used to indicate the flash status in an attribute byte.

See also: `flash-mask`, `flashy`, `set-flash`, `attr!`, `unbright-mask`, `unpaper-mask`, `unink-mask`.

Source file: <src/lib/display.attributes.fs>.

unink-mask

```
unink-mask ( -- b )
```

A `cconstant`. *b* is the inverted bitmask of the bits used to indicate the ink in an attribute byte.

See also: `ink-mask`, `set-ink`, `attr!`, `unpaper-mask`, `unbright-mask`, `unflash-mask`.

Source file: <src/lib/display.attributes.fs>.

unlink-internal

```
unlink-internal ( nt xtp -- )
```

Unlink all words defined between the latest pair `internal` and `end-internal`, linking the first word after `end-internal` to the word before `internal`, thus making all the internal words skipped by the dictionary searches.

Usage example:

```
internal

: hello ( -- ) ." hello" ;

end-internal

: salute ( -- ) hello ;

unlink-internal

salute  \ ok!
hello   \ error!
```

At least one word must be defined between `end-internal` and `unlink-internal`.

The alternative word `hide-internal` can be used instead of `unlink-internal` in order to keep the internal words searchable.

Source file: <src/lib/modules.internal.fs>.

unlocated

```
unlocated ( block -- )
```

A deferred word (see `defer`) called in the loop of `located`, when the word searched for is not located in *block*. Its default action is `drop`, which is changed by `use-fly-index` in order to index the blocks on the fly.

Source file: <src/lib/002.need.fs>.

unloop

```
unloop ( -- ) ( R: loop-sys -- )
```

Discard the `loop` control parameters *loop-sys* for the current nesting level. An `unloop` is required for each nesting level before the definition may be exited with `exit`.

Origin: Forth-94 (CORE), Forth-2012 (CORE).

See also: `leave`, `do`, `?do`, `+loop`.

Source file: <src/kernel.z80s>.

unmarker

```
unmarker ( a -- )
```

Restore the system to the state before the correspondig `marker` was created. The data that describes the state of the system was stored at *a* by `marker,`. The restoration process is the following:

First set the data-space pointer to *a* (`there`), then restore the data stored at *a*: the name-space pointer (`np!`), the latest definition pointers (`last` and `lastxt`), the word lists pointer (`last-wordlist`), the current compilation word list (`set-current`), the search order (`@order`) and the word lists (`@wordlists`).

`unmarker` is a factor of `marker`.

Source file: <src/lib/tool.marker.fs>.

unneeding

```
unneeding ( "name" -- f )
```

Parse *name*. If there’s no unresolved `need`, `needed`, `reneed` or `reneeded`, return `false`. Otherwise, if *name* is the needed word specified by the last execution of `need` or `needed`, return `false`, else return `true`.

See also: `needing`.

Source file: <src/lib/002.need.fs>.

unnest

```
unnest ( R: nest-sys -- )
```

Discard the calling definition specified by *nest-sys*. Before exiting the current definition, a program shall remove any parameters the calling definition had placed on the return stack.

`unnest` is an `alias` of `rdrop`.

Origin: DX-Forth.

See also: `rp`, `exit`, `next`.

Source file: <src/kernel.z80s>.

unnest-source

```
unnest-source ( R: source-sys -- )
```

Restore the source specification described by *source-sys*, which was left by `nest-source`.

`unnest-source` is a `compile-only` word.

Definition:

```
: unnest-source ( R: source-sys -- )
  r>
  r> #tib !
  r> blk !
  r> >in !
  r> (source-id !
  2r> input-buffer 2!
  >r ; compile-only
```

See also: `#tib`, `blk`, `>in`, `(source-id`, `input-buffer`.

Source file: <src/kernel.z80s>.

unpaper-mask

```
unpaper-mask ( -- b )
```

A `cconstant`. *b* is the inverted bitmask of the bits used to indicate the paper in an attribute byte.

See also: `paper-mask`, `papery`, `set-paper`, `attr!`, `unink-mask`, `unbright-mask`, `unflash-mask`.

Source file: <src/lib/display.attributes.fs>.

unpick

```
unpick ( x#u...x#1 x#0 x u -- x...x#1 x#0 )
```

Remove *x* and *u*. Replace *x#u* with *x*. `0 unpick` is equivalent to `nip` (but much slower).

See also: `pick`.

Origin: LaForth’s `put`.

Source file: <src/lib/data_stack.fs>.

unresolved

```
unresolved ( n -- a )
```

Convert element *n* of the cell array pointed by `unresolved>` to its address *a*. `unresolved>` is used to store unresolved addresses during the compilation of `code` words, as a simpler alternative to the Z80 `assembler labels` created by `l:`.

Usage examples (extracted from `ocr`):

---- 0 d stp, >amark 0 unresolved ! \ modify the code to get the screen address later \ (...) 0 d ldp#, \ restore the screen address >amark 0 unresolved @ !

here jr, >rmark 2 unresolved ! \ (...) 2 unresolved @ >rresolve ----

Source file: <src/lib/assembler.fs>.

unresolved0>

```
unresolved0> ( -- a ) "unresolved-zero-greater-than"
```

Address *a* is the default value of `unresolved>`: an 8-cell array.

Source file: <src/lib/assembler.fs>.

unresolved>

```
unresolved> ( -- a ) "unresolved-greater-than"
```

A `variable`. Address *a* contains the address of a cell array accessed by `unresolved`. Its default value is `unresolved0>`, which is an 8-cell array.

The cell array pointed by `unresolved>` is used to store `unresolved` addresses during the compilation of `code` words. This method is a simpler alternative to the Z80 `assembler` `labels` created by `l:`.

See `unresolved` for a usage example.

Source file: <src/lib/assembler.fs>.

unset-capslock

```
unset-capslock ( -- )
```

Unset capslock.

See also: `set-capslock`, `capslock?`, `toggle-capslock`, `capslock`, `creset`.

Source file: <src/lib/keyboard.caps_lock.fs>.

until

```
until
  Compilation: ( C: dest -- )
  Run-time:    ( f -- )
```

Compilation: Compile a conditional `0branch` to the backward reference *dest*, usually left by `begin`.

Run-time: If *f* is zero, continue execution at the location specified by *dest*.

`until` is an `immediate` and `compile-only` word.

Definition:

```
: until \ Compilation: ( C: dest -- )
        \ Run-time:    ( f -- )
  compile 0branch <resolve ; immediate compile-only
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `while`, `again`, `repeat`, `<resolve`, `compile`, `0until`, `-until`, `+until`.

Source file: <src/kernel.z80s>.

unused

```
unused ( -- u )
```

u is the amount of space remaining in the region addressed by `here`, in bytes. This region includes the transient spaces addressed by `pad` and `hold`.

Definition:

```
: unused ( -- u ) limit @ here - ;
```

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `limit`, `here`, `farunused`, `os-unused`, `fyi`, `greeting`.

Source file: <src/kernel.z80s>.

unused-stringer

```
unused-stringer ( -- n )
```

Return the number *n* of free characters in the `stringer`.

`unused-stringer` is written in Z80. Its equivalent definition if Forth is the following:

```
: unused-stringer ( -- n ) +stringer @ ;
```

See also: `+stringer`.

Source file: <src/kernel.z80s>.

up

```
up ( -- a ) "u-p"
```

A `variable`. *a* is the address of a cell containing the user area pointer.

Origin: fig-Forth.

See also: `/user`, `user`.

Source file: <src/kernel.z80s>.

up0

```
up0 ( -- a ) "u-p-zero"
```

A `constant`. *a* is the default address of the user area.

Source file: <src/kernel.z80s>.

update

```
update ( -- )
```

Mark the current block buffer as modified. The block will subsequently be transferred automatically to disk should its buffer be required for storage of a different block, or upon execution of `flush` or `save-buffers`.

Origin: Forth-83 (Required Word Set), Forth-94 (BLOCK), Forth-2012 (BLOCK).

Source file: <src/lib/blocks.fs>.

updated?

```
updated? ( -- f ) "updated-question"
```

f is true if the current disk buffer is marked as modified.

Definition:

```
: updated? ( -- f ) buffer-id 0< ;
```

See also: `update`, `empty-buffers`, `buffer-id`.

Source file: <src/kernel.z80s>.

upper

```
upper ( c -- c' )
```

Convert *c* to uppercase *c'*.

See also: `uppers`, `lower`, `upper_`.

Source file: <src/lib/strings.MISC.fs>.

upper_

```
upper_ ( -- a ) "upper-underscore"
```

Return address *a* of a routine that converts the ASCII character in the A register to uppercase.

See also: `upper`, `lower_`.

Source file: <src/lib/strings.MISC.fs>.

uppers

```
uppers ( ca len -- )
```

Convert string *ca len* to uppercase.

See also: `uppers1`, `lowers`, `upper`.

Source file: <src/lib/strings.MISC.fs>.

uppers1

```
uppers1 ( ca len -- ) "uppers-one"
```

Change the first char of string *ca len* to uppercase.

See also: `uppers`, `upper`.

Source file: <src/lib/strings.MISC.fs>.

use-default-located

```
use-default-located ( -- )
```

Set the default actions of `located` and `unlocated`: Search the blocks.

`use-default-located` is a common factor of `use-no-index` and `use-thru-index`.

Source file: <src/lib/002.need.fs>.

use-default-need

```
use-default-need ( -- )
```

Set the default actions of `need`, `needed`, `reneed`, and `reneeded`: Use `locate` to search the blocks.

`use-default-need` is a common factor of `use-no-index` and `use-fly-index`.

Source file: <src/lib/002.need.fs>.

use-fly-index

```
use-fly-index ( -- )
```

Set the alternative action of `need`, `needed`, `reneed`, `reneeded`, `located` and `unlocated` in order to use the blocks index and index the searched blocks on the fly.

The default action of all said words can be restored by `use-no-index`.

See also: `use-thru-index`.

Source file: <src/lib/blocks.indexer.fly.fs>.

use-no-index

```
use-no-index ( -- )
```

Set the default action of `need`, `needed`, `reneed`, `reneeded` and `unlocated`: Use `locate` to search the blocks.

The alternative actions are set by `use-thru-index` and `use-fly-index`.

See also: `use-default-need`, `use-default-located`.

Source file: <src/lib/002.need.fs>.

use-thru-index

```
use-thru-index ( -- )
```

Change the action of `need`, `needed`, `reneed`, `reneeded`, `located` and `unlocated` in order to use the blocks index created by `make-thru-index`.

The default action of all said words can be restored by `use-no-index`.

See also: `use-fly-index`.

Source file: <src/lib/blocks.indexer.thru.fs>.

user

```
user ( n "name" -- )
```

Parse *name*. Create a user variable *name* in the first available offset within the user area. When *name* is later executed, its absolute user area storage address is placed on the stack.

See also: `2user`, `ucreate`, `uallot`, `?user`.

Source file: <src/lib/data.user.fs>.

ut*

```
ut*   ( ud u -- t ) "u-t-star"
```

t is the signed product of *ud* times *u*.

Source file: <src/lib/math.operators.3-cell.fs>.

ut/

```
ut/   ( ut n -- d ) "u-t-slash"
```

Divide a triple unsigned number *ut* by a single number *n* giving the double number result *d*.

Source file: <src/lib/math.operators.3-cell.fs>.

V

val

```
val ( x "name" -- )
```

Create a definition for *name* that will place *x* on the stack (unless `toval` is used first) and then will execute `init-val`.

`val` is an alternative to the standard `value`.

See also: `cval`, `2val`, `variable`, `constant`.

Source file: <src/lib/data.val.fs>.

value

```
value ( x "name" -- )
```

Create a definition *name* with initial value *x*. When *name* is later executed, *x* will be placed on the stack. `to` can be used to assign a new value to *name*.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `cvalue`, `2value`, `constant`, `variable`, `val`.

Source file: <src/lib/data.value.fs>.

var

```
var ( m v size "name" -- m v' )
```

Define a variable with *size* bytes.

Source file: <src/lib/objects.mini-oof.fs>.

variable

```
variable ( "name" -- )
```

Parse *name*. `create` a definition for *name*, which is referred to as a "variable". `allot` one `cell` of data space, the data field of *name*, to hold the contents of the variable. When *name* is later executed, the address of its data field is placed on the stack.

The program is responsible for initializing the contents of the variable.

Definition:

```
: variable ( "name" -- ) create cell allot ;
```

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `cvariable`, `2variable`, `constant`.

Source file: <src/kernel.z80s>.

version

```
version ( -- ca )
```

ca is the address of a 9-byte data table containing the Solo Forth version, as follows:

+0	major (one byte)
+1	minor (one byte)
+2	patch (one byte)
+3	pre-release identifier (one byte): 'd' for "dev", 'p' for "pre", 'r' for "rc", zero if none
+4	pre-release (one cell)
+6	build (double-cell number)

See also: `.version`.

Source file: <src/kernel.z80s>.

vertical-curtain

```
vertical-curtain ( b -- )
```

Wash the screen with the given color attribute *b* from the left and right columns to the middle.

See also: `horizontal-curtain`.

Source file: <src/lib/graphics.cls.fs>.

view

```
view ( "name" -- )
```

List the block where *name* is defined, i.e. the first block where *name* is in the index line (surrounded by spaces). If *name* cannot be found, `throw` an exception #-286 ("not located").

See also: `locate`, `list`.

Source file: <src/lib/tool.list.blocks.fs>.

vocabulary

```
vocabulary ( "name" -- )
```

Create a vocabulary *name*. A vocabulary is a named word list. Subsequent execution of *name* replaces the first entry in the search order with the word list associated to *name*. When *name* becomes the compilation word list new definitions will be appended to *name*'s word list.

Origin: Forth-83 (Required Word Set).

See also: `wordlist`, `definitions`, `wordlist-of`, `set-current`.

Source file: <src/lib/word_lists.fs>.

W

w/o

```
w/o ( -- fam ) "w-o"
```

Return the "write only" file access method *fam*.

See also: `r/o`, `r/w`, `bin`.

Origin: Forth-94 (FILE), Forth-2012 (FILE).

Source file: <src/lib/dos.gplusdos.fs>.

wacat

```
wacat ( ca len -- ) "w-a-cat"
```

Show a wild-card abbreviated disk catalogue of the current drive using the wild-card filename *ca len*. See the Plus D manual for wild-card syntax.

See also: `cat`, `acat`, `wcat`, `(cat`, `set-drive`.

Source file: <src/lib/dos.gplusdos.fs>.

warm

```
warm ( -- )
```

Do a "warm" restart of the Forth system: Make the `terminal` the current output device, restore the previous display mode (in case `warm` is automatically executed after reentering from BASIC), clear the screen and `abort`.

Definition:

```
: warm ( -- ) display restore-mode page abort ;
```

See also: `cold`, `restore-mode`, `page`.

Source file: <src/kernel.z80s>.

warn

```
warn ( ca len -- ca len )
```

Check if *ca len* already exists in the compilation word list. If so, and if the content of `warnings` is not zero, do a configurable action, usually issue a warning message.

`warn` is a deferred word (see `defer`) which is called by `header,` and whose default action is `noop`. Alternative actions are provided by `message-warn`, `error-code-warn` and `error-warn`.

Source file: <src/kernel.z80s>.

warning"

```
warning"
  Compilation: ( "ccc<quote>" -- )
  Execution:   ( f -- )
"warning-quote"
```

Compilation:

Parse and compile *ccc* delimited by a double quote.

Execution:

If *f* is not zero, display the compiled message *ccc*; else do nothing.

Source file: <src/lib/exception.fs>.

warnings

```
warnings ( -- a )
```

A `user` variable. *a* is the address of a cell containing a flag. If it's zero, no warning is shown when a compiled word is not unique in the compilation word list. Its default value is `true`.

Source file: <src/lib/compilation.fs>.

wat-xy

```
wat-xy ( col row -- ) "w-at-x-y"
```

Store *col row* as the `current-window` cursor coordinates and set the cursor coordinates accordingly. The upper left corner of the `window` is column zero, row zero.

See also: `at-wxy`, `at-xy`.

Source file: <src/lib/display.window.fs>.

wave-display

```
wave-display ( -- )
```

Modify the screen bitmap with a water effect. At the end the original image is restored.

See also: `invert-display`, `fade-display`.

Source file: <src/lib/graphics.display.fs>.

wblank

```
wblank ( -- ) "w-blank"
```

Fill the `current-window` by displaying as many blanks (character `bl`) as needed, starting from the top left corner of the `window`. Finally, reset the cursor position of the window at the upper left corner (column 0, row 0).

`wblank` is a slower but lighter alternative to `wcls`.

See also: `wstamp`, `whome`, `wspace`.

Source file: <src/lib/display.window.fs>.

wcat

```
wcat ( ca len -- ) "w-cat"
```

Show a wild-card disk catalogue of the current drive using the wild-card filename *ca len*. See the Plus D manual for wild-card syntax.

See also: `cat`, `acat`, `wacat`, `(cat`, `set-drive`.

Source file: <src/lib/dos.gplusdos.fs>.

wcls

```
wcls ( -- ) "w-c-l-s-"
```

Clear the `current-window` with the current attribute and reset its cursor position at the upper left corner (column 0, row 0).

See also: `attr-wcls`, `wblank`, `attr@`, `whome`, `clear-rectangle`, `cls`.

Source file: <src/lib/display.window.fs>.

wcolor

```
wcolor ( b -- ) "w-color"
```

Color the `current-window` with color attribute *b*.

See also: `attr-wcls`, `color-rectangle`.

Source file: <src/lib/display.window.fs>.

wcolumns

```
wcolumns ( -- ca ) "w-columns"
```

ca is the address of a byte containing the width in characters of the `current-window`.

See also: `wx`, `wy`, `wx0`, `wy0`, `wrows`.

Source file: <src/lib/display.window.fs>.

wcr

```
wcr ( -- ) "w-c-r"
```

Cause subsequent output to the `current-window` appear at the beginning of the next line.

WARNING When the end of the `window` is reached, the cursor is set to the top left corner with `whome`. In a future version of the code, the window will be scrolled.

See also: `?wcr`, `wcr`.

Source file: <src/lib/display.window.fs>.

wdump

```
wdump ( a len -- ) "w-dump"
```

Show the contents of *len* cells from *a*.

Source file: <src/lib/tool.dump.fs>.

wemit

```
wemit ( c -- ) "w-emit"
```

Display character *c* in the `current-window`.

See also: `wtype`, `wspace`, `emit`.

Source file: <src/lib/display.window.fs>.

wfreecolumns

```
wfreecolumns ( -- n ) "w-free-columns"
```

n is the number of free columns in the current line of the `current-window`.

See also: `wcolumns`.

Source file: <src/lib/display.window.fs>.

where

```
where ( -- )
```

Display information about the last error: block number, line number and a picture of where it occurred. If the error was in the command line, nothing is displayed.

Origin: Forth-79 (Reference Word Set).

See also: `error-pos`, `error`.

Source file: <src/lib/tool.debug.where.fs>.

while

```
while
  Compilation: ( C: dest -- orig dest )
  Run-time:    ( f -- )
```

Compilation: Put the location of a new unresolved forward reference *orig* onto the control-flow stack, under the existing *dest*. Usually *orig* and *dest* are resolved by `repeat`.

Run-time: If *f* is zero, continue execution at the location specified by the resolution of *orig*.

`while` is an `immediate` and `compile-only` word.

Definition:

```
: while \ Compilation: ( C: dest -- orig dest )
        \ Run-time:    ( f -- )
  postpone if cs-swap ; immediate compile-only
```

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `if`, `until`, `cs-swap`, `postpone`, `0while`, `-while`, `+while`.

Source file: <src/kernel.z80s>.

white

```
white ( -- b )
```

A `cconstant` that returns 7, the value that represents the white color.

See also: `black`, `blue`, `red`, `magenta`, `green`, `cyan`, `yellow`, `contrast`, `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

white-noise

```
white-noise ( -- )
```

White noise for ZX Spectrum 48. *u* is the duration in number of sample bytes.

Source file: <src/lib/sound.48.fs>.

whome

```
whome ( -- ) "w-home"
```

Set the `current-window` cursor coordinates to its top left corner: column zero, row zero.

See also: `wat-xy`.

Source file: <src/lib/display.window.fs>.

width

```
width ( -- a )
```

A `user` variable. *a* is the address of a cell containing the maximum number of letters saved in the compilation of a definition name. It must be 1 thru 31, with a default value of 31. The name character count and its natural characters are saved, up to the value in `width`. The value may be changed at any time within the above limits.

Origin: fig-Forth.

Source file: <src/kernel.z80s>.

window

```
window ( col row columns rows -- a )
```

Create a window definition with top left corner at *col row*, with a width *columns* and a height *rows* (both in characters). The internal cursor position of the window in set to its top left corner. *a* is the address of the window data structure, which is `/window` bytes long and has the following structure:

Table 41. Data structure created by `window`*:*

Byte offset	Description
+0	x cursor coordinate
+1	y cursor coordinate
+2	window left x coordinate on screen
+3	window top y coordinate on screen
+4	width in columns
+5	heigth in rows

Windows do not use standard output words like `emit` and `type`. Instead, they use specific words named with the "w" prefix: `wemit`, `wtype`, `wcls`, etc.

NOTE At the moment there’s no word to display numbers in a window. Therefore numbers must be converted to strings first and displayed with `wemit`.

WARNING At the moment windows are not aware of display modes that dont’t use 32 characters per line (e.g. `mode-64ao`, `mode-42pw`). If windows are used when such mode is active, the layout of the output will be wrong.

See also: `current-window`, `wx`, `wy`, `wx0`, `wy0`, `wcolumns`, `wrows`.

Source file: <src/lib/display.window.fs>.

wipe-rectangle

```
wipe-rectangle ( column row width height -- )
```

Clear a screen rectangle at the given character coordinates and of the given size in characters. Only the bitmap is cleared. The color attributes remain unchanged.

See also: `clear-rectangle`, `color-rectangle`, `wcls`.

Source file: <src/lib/graphics.rectangle.fs>.

within

```
within ( n1|u1 n2|u2 n3|u3 -- f )
```

Perform a comparison of a test value n1|u1 with a lower limit *n2|u2* and an upper limit *n3|u3*, returning `true` if either (*n2|u2* < *n3|u3* and (*n2|u2* ⇐ *n1|u1* and *n1|u1* < *n3|u3*)) or (*n2|u2* > *n3|u3* and (*n2|u2* ⇐ *n1|u1* or *n1|u1* < *n3|u3*)) is true, returning `false` otherwise.

Origin: Forth-94 (CORE EXT), Forth-2012 (CORE EXT).

See also: `between`, `polarity`.

Source file: <src/lib/math.operators.1-cell.fs>.

within-of

```
within-of
  Compilation: ( C: -- of-sys )
  Run-time:    ( x1 x2 x3 -- | x1 )
```

A variant of `of`.

Compilation:

Put *of-sys* onto the control flow stack. Append the run-time semantics given below to the current definition. The semantics are incomplete until resolved by a consumer of *of-sys*, such as `endof`.

Run-time:

If *x1* is not in range *x2 x3*, as calculated by `within`, discard *x2 x3* and continue execution at the location specified by the consumer of *of-sys*, e.g., following the next `endof`. Otherwise, consume also *x1* and continue execution in line.

`within-of` is an `immediate` and `compile-only` word.

Usage example:

```
: test ( x -- )
  case
    1          of ." one"                          endof
    2 5 within-of ." within two and five; not five" endof
    5          of ." five"                         endof
  endcase ;
```

See also: `case`, `between-of`, `(within-of`.

Source file: <src/lib/flow.case.fs>.

wltype

```
wltype ( ca len -- ) "w-l-type"
```

Display string *ca len* in the `current-window`, left justified.

See also: `wtype`, `wemit`, `ltype`.

Source file: <src/lib/display.window.fs>.

word

```
word ( c "<chars>ccc<char>" -- ca )
```

Skip leading *c* character delimiters from the input stream. Parse the next text characters from the input stream, until a delimiter *c* is found, storing the packed character string beginning at *ca* (which is the current address returned by `here`), as a counted string (the character count in the first byte), and with one blank at the end (not included in the count).

This word is obsolescent. Its function is superseeded by `parse` and `parse-name`.

NOTE The requirement to follow the string with a space is obsolescent and was included in Forth-94 as a concession to existing programs that use `convert` (superseded by `>number`). A program shall not depend on the existence of the space. The requirement to follow the string with a space was removed from Forth-2012.

Origin: Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

Source file: <src/lib/parsing.fs>.

word-length-mask

```
word-length-mask ( -- b )
```

A `cconstant`. *b* is the bitmask of the word length.

See also: `smudge-mask`, `immediate-mask`, `compile-only-mask`.

Source file: <src/kernel.z80s>.

wordlist

```
wordlist ( -- wid )
```

Create a new word list and return its identifier *wid*, which is the address of the following data structure (`/wordlist` bytes long):

Table 42. Data structure created by `wordlist`*.*

Cell	Description
0	*nt* of the latest definition in the word list
1	*wid* of the previous word list, or zero
2	*nt* of the word-list name, or zero

Definition:

```
: wordlist ( -- wid ) here wordlist, ;
```

See also: `wordlist,`, `set-order`, `vocabulary`, `last-wordlist`, `wordlist>last`, `wordlist>link`, `wordlist>name`, `/wordlist`, `wordlists`, `dump-wordlists`.

Source file: <src/kernel.z80s>.

wordlist,

```
wordlist, ( -- ) "wordlist-comma"
```

Compile in data space the contents of a new word list.

Definition:

```
: wordlist, ( -- )
  here 0 , last-wordlist @ , last-wordlist ! 0 , ;
```

See also: `wordlist`, `last-wordlist`, `wordlist>last`, `wordlist>link`, `wordlist>name`, `/wordlist`.

Source file: <src/kernel.z80s>.

wordlist-name!

```
wordlist-name! ( nt wid -- ) "wordlist-name-store"
```

Store *nt* as the name associated to the word list identified by *wid*. *nt* is stored into the name field of the word-list metadata.

See also: `wordlist`, `wordlist-name@`, `wordlist>name`.

Source file: <src/lib/word_lists.fs>.

wordlist-name@

```
wordlist-name@ ( wid -- nt|0 ) "wordlist-name-fetch"
```

Fetch from the word-list identifier *wid* its associated name *nt*, or zero if the word list has no associated name.

See also: `wordlist`, `wordlist-name!`, `wordlist>name`.

Source file: <src/lib/word_lists.fs>.

wordlist-of

```
wordlist-of ( "name" -- wid )
```

Return the word-list identifier *wid* associated to vocabulary *name*.

Origin: eForth’s `widof`.

See also: `wordlist`, `vocabulary`.

Source file: <src/lib/word_lists.fs>.

wordlist-words

```
wordlist-words ( wid -- )
```

List the definition names in word list *wid*.

See also: `words`, `wordlists`.

Source file: <src/lib/tool.list.words.fs>.

wordlist>last

```
wordlist>last ( wid -- a ) "wordlist-to-last"
```

Return the field address *a* of `wordlist` identifier *wid*, which holds the name token of the latest word defined in *wid*.

As *a* is the first field of a word-list structure, `wordlist>last` is provided only for legibility. It is an `immediate alias` of `noop`.

See also: `wordlist>name`, `wordlist>link`, `/wordlist`, `last`, `latest`.

Source file: <src/lib/word_lists.fs>.

wordlist>link

```
wordlist>link ( wid -- a ) "wordlist-to-link"
```

Return the link field address *a* of the `wordlist` identifier *wid*, which holds the word-list identifier of the previous word list defined in the system.

See also: `wordlist>name`, `wordlist>last`, `/wordlist`.

Source file: <src/lib/word_lists.fs>.

wordlist>name

```
wordlist>name ( wid -- a ) "wordlist-to-name"
```

Return the address *a* which holds the *nt* of the `wordlist` identifier *wid* (or zero if the word list has no associated name).

See also: `wordlist>link`, `wordlist>last`, `/wordlist`.

Source file: <src/lib/word_lists.fs>.

wordlist>vocabulary

```
wordlist>vocabulary ( wid "name" -- ) "wordlist-to-vocabulary"
```

Create a vocabulary *name* for the word list identified by *wid*.

See also: `wordlist`, `vocabulary`, `latest>wordlist`, `wordlists`.

Source file: <src/lib/word_lists.fs>.

wordlists

```
wordlists ( -- )
```

List all wordlists defined in the system, either by name (if they have an associated name) or by number (its word list identifier, if they have no associated name). The word lists are listed in reverse chronological order: The first word list listed is the most recently defined.

See also: `.wordlist`, `words`, `wordlist-words`, `wordlist`, `last-wordlist`.

Source file: <src/lib/tool.list.word_lists.fs>.

wordlists,

```
wordlists, ( -- ) "wordlists-comma"
```

Store all of the current word lists in the data space, updating `dp`.

`wordlists,` is a factor of `marker,`.

See also: `@wordlists`, `order,`, `wordlist`.

Source file: <src/lib/tool.marker.fs>.

words

```
words ( -- )
```

List the definition names in the first word list of the search order.

Origin: Forth-83 (Uncontrolled Reference Words), Forth-94 (TOOLS), Forth-2012 (TOOLS).

See also: `wordlist-words`, `wordlists`.

Source file: <src/lib/tool.list.words.fs>.

words#

```
words# ( -- n ) "words-number-sign"
```

Return number *n* of words defined in the first word list of the search order.

Source file: <src/lib/tool.list.words.fs>.

words-like

```
words-like ( "name" -- )
```

List the definition names, from the first word list of the search order, that contain substring "name".

Source file: <src/lib/tool.list.words.fs>.

write-block

```
write-block ( n -- )
```

Write the buffer to disk block *n*.

Definition:

```
: write-block ( n -- ) write-mode transfer-block ;
```

See also: `write-mode`, `transfer-block`, `read-block`, `block`.

Source file: <src/kernel.z80s>.

write-mode

```
write-mode ( -- )
```

Set the write mode for `transfer-sector` and `transfer-block`.

See also: `read-mode`.

Source file: <src/kernel.gplusdos.z80s>.

wrows

```
wrows ( -- ca ) "w-rows"
```

ca is the address of a byte containing the heigth in rows of the `current-window`.

See also: `wx`, `wy`, `wx0`, `wy0`, `wcolumns`.

Source file: <src/lib/display.window.fs>.

wspace

```
wspace ( -- ) "w-space"
```

Display one space in the `current-window`.

See also: `space`.

Source file: <src/lib/display.window.fs>.

wstamp

```
wstamp ( c -- ) "w-stamp"
```

Fill the `current-window` by displaying as many characters *c* as needed, starting from the top left corner. The cursor position of the window is not changed.

See also: `wblank`, `wcls`, `wemit`.

Source file: <src/lib/display.window.fs>.

wtype

```
wtype ( ca len -- ) "w-type"
```

Display string *ca len* in the `current-window`.

See also: `wltype`, `wemit`, `ltype`.

Source file: <src/lib/display.window.fs>.

wtype+

```
wtype+ ( ca len -- ) "w-type-plus"
```

Display string *ca len* in the `current-window` and update the `window` coordinates accordingly.

Source file: <src/lib/display.window.fs>.

wtyped

```
wtyped ( -- a ) "w-typed"
```

A `variable`. *a* is the address o a cell containing a flag indicating if a space-delimited substring was found and displayed in the `current-window`. Otherwise, the string must be broken in order to fit the current line of the `window`.

`wtyped` is used by `wtype+` and `wltype`.

Source file: <src/lib/display.window.fs>.

wx

```
wx ( -- ca ) "w-x"
```

ca is the address of a byte containing the x cursor coordinate of the `current-window`.

See also: `wy`, `wx0`, `wy0`, `wcolumns`, `wrows`.

Source file: <src/lib/display.window.fs>.

wx+!

```
wx+! ( n -- ) "w-x-plus-store"
```

Add *n* character positions to the column cursor coordinate of the current `window`. `wx+!` is a factor of `wtype+`.

Source file: <src/lib/display.window.fs>.

wx0

```
wx0 ( -- ca ) "w-x-zero"
```

ca is the address of a byte containing the left x coordinate on screen of the `current-window`.

See also: `wx`, `wy`, `wy0`, `wcolumns`, `wrows`.

Source file: <src/lib/display.window.fs>.

wy

```
wy ( -- ca ) "w-y"
```

ca is the address of a byte containing the y cursor coordinate of the `current-window`.

See also: `wx`, `wx0`, `wy0`, `wcolumns`, `wrows`.

Source file: <src/lib/display.window.fs>.

wy0

```
wy0 ( -- ca ) "w-y-zero"
```

ca is the address of a byte containing the top y coordinate on screen of the `current-window`.

See also: `wx`, `wy`, `wx0`, `wcolumns`, `wrows`.

Source file: <src/lib/display.window.fs>.

X

x

```
x ( "ccc<eol>" -- )
```

A command of `specforth-editor`: Find and delete the next occurrence of the string *ccc*.

See also: `b`, `c`, `d`, `e`, `f`, `h`, `i`, `l`, `m`, `n`, `p`, `r`, `s`, `t`, `text`, `find`, `delete`.

Source file: <src/lib/prog.editor.specforth.fs>.

x1

```
x1 ( -- a ) "x-one"
```

A `2variable` used by `adraw176` and `aline176`.

See also: `y1`, `incx`, `incy`.

Source file: <src/lib/graphics.lines.fs>.

x>

```
x> ( -- x ) ( X: x -- ) "x-from"
```

Move *x* from the current `xstack` to the data stack.

See also: `x>`, `x@`.

Source file: <src/lib/data.xstack.fs>.

x>gx

```
x>gx ( col -- gx ) "x-to-g-x"
```

Convert cursor coordinate *col* (0..31) to graphic coordinate *gx* (0..255).

`x>gx` is an `alias` of `8*`.

See also: `xy>gxy`, `xy>gxy176`.

Source file: <src/lib/display.cursor.fs>.

x>gx

```
x>gx ( col -- gx ) "x-to-g-x"
```

Convert cursor column *col* to graphic x coordinate *gx*.

See also: `y>gy`, `gx>x`.

Source file: <src/lib/graphics.pixels.fs>.

x@

```
x@ ( -- x ) ( X: x -- x ) "x-fetch"
```

Copy *x* from the current `xstack` to the data stack.

See also: `x>`, `>x`.

Source file: <src/lib/data.xstack.fs>.

xclear

```
xclear ( -- ) "x-clear"
```

Clear the current `xstack`.

See also: `xdrop`, `2xdrop`, `xp0`, `xp`.

Source file: <src/lib/data.xstack.fs>.

xdepth

```
xdepth ( -- n ) "x-depth"
```

n is the number of single-cells values contained in the current `xstack`.

See also: `.xs`, `xlen`.

Source file: <src/lib/data.xstack.fs>.

xdrop

```
xdrop ( X: x -- ) "x-drop"
```

Remove *x* from the `xstack`.

See also: `>x`, `x>`.

Source file: <src/lib/data.xstack.fs>.

xdup

```
xdup ( X: x -- x x ) "x-dup"
```

Duplicate *x* in the current `xstack`.

See also: `2xdup`.

Source file: <src/lib/data.xstack.fs>.

xfree

```
xfree ( -- ) "x-free"
```

Free the space used by the current `xstack`, which was created by `allocate-xstack`.

Source file: <src/lib/data.xstack.fs>.

xkey

```
xkey ( -- c ) "x-key"
```

Show a cursor, wait for the next terminal key struck; if it's the caps lock key, toggle caps and keep waiting; else leave the character code *c* of the key struck.

See also: `key`, `-keys`.

Source file: <src/kernel.z80s>.

xlen

```
xlen ( -- n ) "x-len"
```

n is the length of the current `xstack`, in bytes.

See also: `xdepth`.

Source file: <src/lib/data.xstack.fs>.

xliteral

```
xliteral ( x -- ) "x-literal"
```

If *x* is a byte, execute `cliteral`, else execute `literal`.

`xliteral` is used in `interpret-table` to compile the single-cell literals. It is useful as an alternative to `literal`, in order to optimize the code when *x* is unknown.

`xliteral` is an `immediate` and `compile-only` word.

Definition:

```
: xliteral ( x -- )
  dup byte? if   postpone cliteral exit
           then postpone literal ; immediate compile-only
```

See also: `2literal`, `]xl`, `byte?`.

Source file: <src/kernel.z80s>.

xor

```
xor ( x1 x2 -- x3 ) "x-or"
```

x3 is the bit-by-bit exclusive-or of *x1* with *x2*.

Origin: fig-Forth, Forth-79 (Required Word Set), Forth-83 (Required Word Set), Forth-94 (CORE), Forth-2012 (CORE).

See also: `or`, `and`, `negate`, `0=`, `dxor`.

Source file: <src/kernel.z80s>.

xor#,

```
xor#, ( b -- ) "x-or-number-sign-comma"
```

Compile the Z80 `assembler` instruction `XOR b`.

See also: `or#,`, `and#,`, `add#,`, `sub#,`.

Source file: <src/lib/assembler.fs>.

xor,

```
xor, ( reg -- ) "x-or-comma"
```

Compile the Z80 `assembler` instruction `XOR reg`.

See also: `and,`, `or,`.

Source file: <src/lib/assembler.fs>.

xorx,

```
xorx, ( disp regpi --  ) "x-or-x-comma"
```

Compile the Z80 `assembler` instruction `XOR (regpi+disp)`.

See also: `xorx,`, `orx,`, `cpx,`.

Source file: <src/lib/assembler.fs>.

xover

```
xover ( X: x1 x2 -- x1 x2 x1 ) "x-over"
```

Place a copy of *x1* on top of the `xstack`.

Source file: <src/lib/data.xstack.fs>.

xp

```
xp ( -- a ) "x-p"
```

A `variable`. Address *a* holds the address of the current `xstack` pointer.

Source file: <src/lib/data.xstack.fs>.

xp0

```
xp0 ( -- a ) "x-p-zero"
```

Initial address of the current `xstack` pointer.

Source file: <src/lib/data.xstack.fs>.

xpick

```
xpick ( u -- x#u ) ( X: x#u...x#0 -- x#u...x#0 ) "x-pick"
```

Remove *u*. Copy *x#u* from the current `xstack` to the data stack.

Source file: <src/lib/data.xstack.fs>.

xsize

```
xsize ( -- n ) "x-size"
```

Size of the current `xstack` in bytes.

Source file: <src/lib/data.xstack.fs>.

xstack

```
xstack ( a -- ) "x-stack"
```

Make the extra stack *a* the current one. *a* is the address returned by `allot-xstack` or `allocate-xstack` when the extra stack was created.

Extra stacks grow towards high memory. *a* is the address of a table that contains the metadata of the xstack, which is the following:

```
+0 = initial value of the stack pointer (1 cell below the
     stack space)
+2 = stack pointer
+4 = maximum size in bytes
```

`xp0`, `xp` and `xsize` are used to access the contents of the table.

See also: `estack`.

Source file: <src/lib/data.xstack.fs>.

xt-replaces

```
xt-replaces ( xt ca len -- ) "x-t-replaces"
```

Set *xt* (whose execution returns the address and length of a string) as the text to substitute for the substitution named by *ca len*. If the substitution does not exist it is created.

The name of a substitution should not contain the "%" delimiter character.

See also: `replaces`, `substitute`, `unescape`, `substitute-wordlist`.

Source file: <src/lib/strings.xt-replaces.fs>.

xt-substitution

```
xt-substitution ( ca len -- a ) "x-t-substitution"
```

Given a string *ca len* create its substitution and storage space. Return the address that will hold the execution token of the substitution.

See also: `xt-replaces`.

Source file: <src/lib/strings.xt-replaces.fs>.

xy

```
xy ( -- col row ) "x-y"
```

Return the current column and row of the text cursor.

`xy` is a deferred word (see `defer`) whose default action is `mode-32-xy`.

See also: `at-xy`.

Source file: <src/kernel.z80s>.

xy>attr

```
xy>attr ( col row -- b ) "x-y-to-attribute-a"
```

Return the color attribute *b* of the given cursor coordinates *col row*.

See also: `xy>attra`, `xy>attra_`, `xy>gxy`.

Source file: <src/lib/display.cursor.fs>.

xy>attra

```
xy>attra ( col row -- a ) "x-y-to-attribute-a"
```

Return the color attribute address *a* of the given cursor coordinates *col row*.

See also: `xy>attr`, `xy>attra_`, `xy>gxy`.

Source file: <src/lib/display.cursor.fs>.

xy>attra_

```
xy>attra_ ( -- a ) "x-y-to-attribute-a-underscore"
```

Return the address *a* of a Z80 routine that calculates the attribute address of a cursor position. This routine is a modified version of the ROM routine at 0x2583.

Input:

- D = column (0..31) - E = row (0..23)

Output:

- HL = address of the attribute in the screen

See also: `xy>attra`, `xy>attr`, `xy>gxy`.

Source file: <src/lib/display.cursor.fs>.

xy>gxy

```
xy>gxy ( col row -- gx gy ) "x-y-to-g-x-y"
```

Convert cursor coordinates *col row* to graphic coordinates *gx gy*. *col* is 0..31, *row* is 0..23, *gx* is 0..255 and *gy* is 0..191.

See also: `xy>attra`, `xy>attr`, `xy>gxy176`, `plot`, `set-pixel`.

Source file: <src/lib/display.cursor.fs>.

xy>gxy176

```
xy>gxy176 ( col row -- gx gy ) "x-y-to-g-x-y-176"
```

Convert cursor coordinates *col row* to graphic coordinates *gx gy* (as used by Sinclair BASIC, i.e. the lower 16 pixel rows are not used). *col* is 0..31, *row* is 0..23, *gx* is 0..255 and *gy* is 0..175.

`xy>gxy176` is provided to make it easier to adapt Sinclair BASIC programs.

See also: `xy>gxy`, `plot176`, `set-pixel176`.

Source file: <src/lib/display.cursor.fs>.

xy>r

```
xy>r ( -- ) ( R: -- col row ) "x-y-to-r"
```

Save the current cursor coordinates to the return stack.

See also: `r>xy`, `save-mode`.

Source file: <src/lib/display.cursor.fs>.

xy>scra

```
xy>scra ( col row -- a ) "x-y-to-s-c-r-a"
```

Convert cursor coordinates *col row* to their correspondent screen address *a*.

See also: `xy>scra_` , `gxy>scra`.

Source file: <src/lib/display.cursor.fs>.

xy>scra_

```
xy>scra_ ( -- a ) "x-y-to-s-c-r-a-underscore"
```

Return address *a* of a Z80 routine that calculates the screen address correspondent to given cursor coordinates.

Input registers:

- B = y coordinate (0..23)
- C = x coordinate (0..31)

Output registers:

- DE = screen address

See also: `xy>scra`, `gxy>scra_`.

Source file: <src/lib/display.cursor.fs>.

y

y

```
y ( -- )
```

A command of `gforth-editor`: Yank deleted string.

See also: `d`, `dl`, `l`, `delete`, `insert`.

Source file: <src/lib/prog.editor.gforth.fs>.

y/n

```
y/n ( -- c ) "y-slash-n"
```

Wait for a valid `key` press for a "yes/no" question and return its code *c*, which is `"y"` or `"n"`.

See also: `y/n?`.

Source file: <src/lib/keyboard.yes-question.fs>.

y/n?

```
y/n? ( c -- f ) "y-slash-n-question"
```

Is character *c*, converted to lowercase, a valid answer for a "y/n" question? I.e., is *c* the current value of `"y"` or `"n"`?

See also: `yes?`, `no?`, `y/n`.

Source file: <src/lib/keyboard.yes-question.fs>.

y1

```
y1 ( -- a ) "y-one"
```

A `2variable` used by `adraw176` and `aline176`.

See also: `x1`, `incx`, `incy`.

Source file: <src/lib/graphics.lines.fs>.

y>gy

```
y>gy ( row -- gy ) "y-to-g-y"
```

Convert cursor coordinate *row* (0..23) to graphic coordinate *gy* (0..191).

See also: `xy>gxy`, `x>gx`.

Source file: <src/lib/display.cursor.fs>.

y>gy

```
y>gy ( row -- gy ) "y-to-g-y"
```

Convert cursor coordinate *row* to graphic coordinate *gy*.

See also: `x>gx`, `gy>y`.

Source file: <src/lib/graphics.pixels.fs>.

yellow

```
yellow ( -- b )
```

A `cconstant` that returns 6, the value that represents the yellow color.

See also: `black`, `blue`, `red`, `magenta`, `green`, `cyan`, `white`, `contrast`, `papery`, `inversely`.

Source file: <src/lib/display.attributes.fs>.

yes?

```
yes? ( -- f ) "yes-question"
```

Wait for a valid `key` press for a `y/n` question and return `true` if it’s the current value of `"y"`, else return `false`.

See also: `no?`, `y/n?`.

Source file: <src/lib/keyboard.yes-question.fs>.

Z

z?

```
z? ( -- op ) "z-question"
```

Return the opcode *op* of the Z80 `assembler` instruction `jp z`, to be used as condition and consumed by `?ret,`, `?jp,`, `?call,`, `?jr,`, `aif`, `rif`, `awhile`, `rwhile`, `auntil` or `runtil`.

See also: `nz?`, `c?`, `nc?`, `po?`, `pe?`, `p?`, `m?`.

Source file: <src/lib/assembler.fs>.

{

{

```
{ ( -- )
```

Part of `hayes-tester`: Start a Hayes test.

See also: `->`, `}`.

Source file: <src/lib/meta.tester.hayes.fs>.

{do

```
{do "curly-bracket-do"
  Compilation: ( C: -- dest )
  Run-time:    ( -- )
```

Start a `{do` control structure.

See also: `do}`, `|do|`, `do>`.

Source file: <src/lib/flow.dijkstra.fs>.

{if

```
{if "curly-bracket-if"
  Compilation: ( -- cs-mark )
```

Start a `{if` control structure.

See also: `if}`, `if>`, `|if|`.

Source file: <src/lib/flow.dijkstra.fs>.

|

|*

```
|* ( -- ) "bar-star"
```

Compile the `multiply` ROM `calculator` command.

See also: `|/`, `|**`.

Source file: <src/lib/math.calculator.fs>.

|**

```
|** ( -- ) "bar-star-star"
```

Compile the `to-power` ROM `calculator` command.

See also: `|sqrt`, `|*`.

Source file: <src/lib/math.calculator.fs>.

|+

```
|+ ( -- ) "bar-plus"
```

Compile the `addition` ROM `calculator` command.

See also: `|-`.

Source file: <src/lib/math.calculator.fs>.

|-

```
|- ( -- ) "bar-minus"
```

Compile the `subtract` ROM `calculator` command.

See also: `|+`.

Source file: <src/lib/math.calculator.fs>.

|/

```
|/ ( -- ) "bar-slash"
```

Compile the `division` ROM `calculator` command.

See also: `|mod`, `|*`.

Source file: <src/lib/math.calculator.fs>.

|0

```
|0 ( -- ) "bar-zero"
```

Compile the ROM calculator command that stacks 0.

See also: `|half`, `|1`, `|10`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|0<

```
|0< ( -- ) "bar-zero-less"
```

Compile the `less-0` ROM `calculator` command.

See also: `|0=`, `|0>`, `|=`, `|<>`, `|>`, `|<`, `|<=`, `|>=`.

Source file: <src/lib/math.calculator.fs>.

|0=

```
|0= ( -- ) "bar-zero-equals"
```

Compile the `not` ROM `calculator` command.

See also: `|0<`, `|0>`, `|=`, `|<>`, `|>`, `|<`, `|<=`, `|>=`.

Source file: <src/lib/math.calculator.fs>.

|0>

```
|0> ( -- ) "bar-zero-greater"
```

Compile the `greater-0` ROM `calculator` command.

See also: `|0=`, `|0<`, `|=`, `|<>`, `|>`, `|<`, `|<=`, `|>=`.

Source file: <src/lib/math.calculator.fs>.

|0branch

```
|0branch ( -- ) "bar-zero-branch"
```

Compile ROM `calculator` commands `|0=` and `|?branch` to do a jump when the floating-point TOS is zero.

See also: `|branch`, `|?branch`.

Source file: <src/lib/math.calculator.fs>.

|1

```
|1 ( -- ) "bar-one"
```

Compile the ROM calculator command that stacks 1.

See also: `|0`, `|half`, `|10`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|10

```
|10 ( -- ) "bar-ten"
```

Compile the ROM calculator command that stacks 10.

See also: `|0`, `|half`, `|1`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|2dup

```
|2dup ( -- ) "bar-two-dup"
```

Compile the ROM `calculator` commands to do `2dup`, using `|>mem` and `|mem>` (calculator memory positions 1 and 2 are used).

See also: `|drop`, `|dup`, `|swap`, `|over`.

Source file: <src/lib/math.calculator.fs>.

|<

```
|< ( -- ) "bar-less"
```

Compile the `no-less` ROM `calculator` command.

WARNING	This calculator command doesn’t work fine when used from Forth. See its source file for details.

See also: `|0=`, `|0<`, `|0>`, `|=`, `|<>`, `|>`, `|<=`, `|>=`.

Source file: <src/lib/math.calculator.fs>.

|<=

```
|<= ( -- ) "bar-less-equals"
```

Compile the `no-l-eql` ROM `calculator` command.

WARNING	This calculator command doesn’t work fine when used from Forth. See its source file for details.

See also: `|0=`, `|0<`, `|0>`, `|=`, `|<>`, `|>`, `|<`, `|>=`.

Source file: <src/lib/math.calculator.fs>.

|<>

```
|<> ( -- ) "bar-not-equals"
```

Compile the `nos-neql` ROM `calculator` command.

WARNING	This calculator command doesn’t work fine when used from Forth. See its source file for details.

See also: `|0=`, `|0<`, `|0>`, `|=`, `|>`, `|<`, `|<=`, `|>=`.

Source file: <src/lib/math.calculator.fs>.

|<mark

```
|<mark ( -- dest ) "bar-from-mark"
```

Leave the address *dest* of the current data-space pointer as the destination of a ROM `calculator` backward branch which will later be resolved by `|<resolve`.

Typically used before either `|branch`, `|?branch` or `|0branch`.

Source file: <src/lib/math.calculator.fs>.

|<resolve

```
|<resolve ( dest -- ) "bar-from-resolve"
```

Resolve a ROM `calculator` backward branch by compiling the displacement from the current position to address *dest*, which was left by `|<mark`.

Source file: <src/lib/math.calculator.fs>.

|=

```
|= ( -- ) "bar-equals"
```

Compile the `nos-eql` ROM `calculator` command.

WARNING	This calculator command doesn’t work fine when used from Forth. See its source file for details.

See also: `|0=`, `|0<`, `|0>`, `|<>`, `|>`, `|<`, `|<=`, `|>=`.

Source file: <src/lib/math.calculator.fs>.

|>

```
|> ( -- ) "bar-greater"
```

Compile the `no-grtr` ROM `calculator` command.

WARNING	This calculator command doesn’t work fine when used from Forth. See its source file for details.

See also: `|0=`, `|0<`, `|0>`, `|=`, `|<>`, `|<`, `|<=`, `|>=`.

Source file: <src/lib/math.calculator.fs>.

|>=

```
|>= ( -- ) "bar-greater-equals"
```

Compile the `no-gr-eql` ROM `calculator` command.

WARNING This calculator command doesn’t work fine when used from Forth. See its source file for details.

See also: `|0=`, `|0<`, `|0>`, `|=`, `|<>`, `|>`, `|<`, `|<=`.

Source file: <src/lib/math.calculator.fs>.

|>mark

```
|>mark ( -- a ) "bar-greater-mark"
```

Compile space for the displacement of a ROM `calculator` forward branch which will later be resolved by `|>resolve`.

Typically used before either `|branch`, `|?branch` or `|0branch`.

Source file: <src/lib/math.calculator.fs>.

|>mem

```
|>mem ( n -- ) "bar-to-mem"
```

Compile the `st-mem` ROM `calculator` command for memory number *n* (0..5).

NOTE `st-mem` copies the floating-point TOS to the the calculator memory number *n*, but does not remove it from the floating-point stack.

Source file: <src/lib/math.calculator.fs>.

|>resolve

```
|>resolve ( orig -- ) "bar-to-resolve"
```

Resolve a ROM `calculator` forward branch by storing the displacement from *orig* to the current position into *orig*, which was left by `|>mark`.

Source file: <src/lib/math.calculator.fs>.

|?branch

```
|?branch ( -- ) "bar-question-branch"
```

Compile the `jump-true` ROM `calculator` command.

See also: `|0branch`, `|branch`.

Source file: <src/lib/math.calculator.fs>.

|abs

```
|abs ( -- ) "bar-abs"
```

Compile the `abs` ROM `calculator` command.

See also: `|sgn`, `|int`, `|truncate`.

Source file: <src/lib/math.calculator.fs>.

|acos

```
|acos ( -- ) "bar-a-cos"
```

Compile the `acos` ROM `calculator` command.

See also: `|asin`, `|atan`, `|cos`, `|sin`, `|tan`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|asin

```
|asin ( -- ) "bar-a-sin"
```

Compile the `asin` ROM `calculator` command.

See also: `|acos`, `|atan`, `|cos`, `|sin`, `|tan`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|atan

```
|atan ( -- ) "bar-a-tan"
```

Compile the `atan` ROM `calculator` command.

See also: `|acos`, `|asin`, `|cos`, `|sin`, `|tan`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|branch

```
|branch ( -- ) "bar-branch"
```

Compile the `jump` ROM `calculator` command.

See also: `|0branch`, `|?branch`.

Source file: <src/lib/math.calculator.fs>.

|cos

```
|cos ( -- ) "bar-cos"
```

Compile the `cos` ROM `calculator` command.

See also: `|acos`, `|asin`, `|atan`, `|sin`, `|tan`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|do|

```
|do| "bar-do-bar"
  Compilation: ( C: orig dest -- dest )
```

Part of the `{do` control structure.

Source file: <src/lib/flow.dijkstra.fs>.

|drop

```
|drop ( -- ) "bar-drop"
```

Compile the `delete` ROM `calculator` command.

See also: `|dup`, `|swap`, `|over`, `|2dup`.

Source file: <src/lib/math.calculator.fs>.

|dup

```
|dup ( -- ) "bar-dup"
```

Compile the `duplicate` ROM `calculator` command.

See also: `|drop`, `|swap`, `|over`, `|2dup`.

Source file: <src/lib/math.calculator.fs>.

|else

```
|else ( orig1 -- orig2 ) "bar-else"
```

Compile a ROM `calculator` unconditional `|branch` and return the address *orig2* of its destination address, to be resolved by `|then`; then resolve the forward reference *orig1*, left by `|if`.

See also: `|>mark`, `|>resolve`.

Source file: <src/lib/math.calculator.fs>.

|exp

```
|exp ( -- ) "bar-exp"
```

Compile the `exp` ROM `calculator` command.

See also: `|ln`.

Source file: <src/lib/math.calculator.fs>.

|from-here

```
|from-here ( a -- n ) "bar-from-here"
```

Calculate the displacement *n* from the current data-space pointer to address *a*. Used by `|>resolve` and `|<resolve`.

Source file: <src/lib/math.calculator.fs>.

|half

```
|half ( -- ) "bar-half"
```

Compile the ROM calculator command that stacks 1/2.

See also: `|0`, `|1`, `|10`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|if

```
|if ( -- orig ) "bar-if"
```

Compile a ROM `calculator` conditional `|0branch` and return the address *orig* of its destination address, to be resolved by `|else` or `|then`.

See also: `|>mark`.

Source file: <src/lib/math.calculator.fs>.

|if|

```
|if| "bar-if-bar"
  Compilation: ( count -- count )
               ( C: orig...orig1 -- orig...orig2 )
```

Part of the `{if` control structure.

Source file: <src/lib/flow.dijkstra.fs>.

|int

```
|int ( -- ) "bar-int"
```

Compile the `int` ROM `calculator` command.

See also: `|abs`, `|truncate`.

Source file: <src/lib/math.calculator.fs>.

|ln

```
|ln ( -- ) "bar-l-n"
```

Compile the `ln` ROM `calculator` command.

See also: `|exp`.

Source file: <src/lib/math.calculator.fs>.

|mem>

```
|mem> ( n -- ) "bar-mem-to"
```

Compile the `get-mem` ROM `calculator` command for memory number *n* (0..5).

Source file: <src/lib/math.calculator.fs>.

|mod

```
|mod ( -- ) "bar-mod"
```

Compile the `n-mod-m` ROM `calculator` command.

See also: `|/`.

Source file: <src/lib/math.calculator.fs>.

|negate

```
|negate ( -- ) "bar-negate"
```

Compile the `negate` ROM `calculator` command.

See also: `|abs`, `|sgn`.

Source file: <src/lib/math.calculator.fs>.

|over

```
|over ( -- ) "bar-over"
```

Compile the ROM calculator commands to do `over`, using `|>mem` and `|mem>` (calculator memory positions 1 and 2 are used).

See also: `|drop`, `|dup`, `|swap`, `|2dup`.

Source file: <src/lib/math.calculator.fs>.

|pi2/

```
|pi2/ ( -- ) "bar-pi-two-slash"
```

Compile the ROM calculator command that stacks pi/2.

See also: `|0`, `|half`, `|1`, `|10`, `|acos`, `|asin`, `|atan`, `|sin`, `|cos`, `|tan`.

Source file: <src/lib/math.calculator.fs>.

|re-stack

```
|re-stack ( r -- r' ) "bar-re-stack"
```

Compile the `re-stack` ROM `calculator` command.

Source file: <src/lib/math.calculator.fs>.

|sgn

```
|sgn ( -- ) "bar-s-g-n"
```

Compile the `sgn` ROM `calculator` command.

See also: `|abs`, `|negate`.

Source file: <src/lib/math.calculator.fs>.

|sin

```
|sin ( -- ) "bar-sin"
```

Compile the `sin` ROM `calculator` command.

See also: `|acos`, `|asin`, `|atan`, `|cos`, `|tan`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|sqrt

```
|sqrt ( -- ) "bar-s-q-r-t"
```

Compile the `sqr` ROM `calculator` command.

See also: `|**`.

Source file: <src/lib/math.calculator.fs>.

|swap

```
|swap ( -- ) "bar-swap"
```

Compile the `exchange` ROM `calculator` command.

See also: `|drop`, `|dup`, `|over`, `|2dup`.

Source file: <src/lib/math.calculator.fs>.

|tan

```
|tan ( -- ) "bar-tan"
```

Compile the `tan` ROM `calculator` command.

See also: `|acos`, `|asin`, `|atan`, `|cos`, `|sin`, `|pi2/`.

Source file: <src/lib/math.calculator.fs>.

|then

```
|then ( orig -- ) "bar-then"
```

Resolve the forward reference *orig*, left by `|else` or `|if`, the `calculator` conditional control-flow structure.

See also: `|>resolve`.

Source file: <src/lib/math.calculator.fs>.

|truncate

```
|truncate ( -- ) "bar-truncate"
```

Compile the `truncate` ROM `calculator` command.

See also: `|abs`, `|int`.

Source file: <src/lib/math.calculator.fs>.

}

}

```
} ( a1 n -- a2 ) "right-curly-bracket"
```

If in range, return address *a2* of the *n* item of the 1-cell array *a1*. Otherwise `throw` an exception #-272 ("array index out of range").

See also: `1array`, `array>items`.

Source file: <src/lib/data.array.noble.fs>.

}

```
} ( i*x -- )
```

Part of `hayes-tester`: End a Hayes test by comparing stack (expected) contents with saved (actual) contents.

See also: `{`, `->`.

Source file: <src/lib/meta.tester.hayes.fs>.

}bench

```
}bench ( -- d ) "curly-bracket-bench"
```

Return the current value of the clock ticks.

See also: `bench{`, `dticks`, `bench.`, `}bench.`.

Source file: <src/lib/time.fs>.

}bench.

```
}bench. ( -- ) "curly-bracket-bench-dot"
```

Stop timing and display the result.

See also: `bench{`, `}bench`, `bench.`.

Source file: <src/lib/time.fs>.

}private

```
}private ( -- ) "curly-bracket-private"
```

End private definitions. See `privatize` for a usage example.

Source file: <src/lib/modules.privatize.fs>.

}t

```
}t ( i*x -- )
```

Part of `ttester`: End a test by comparing stack (expected) contents with saved (actual) contents.

See also: `t{`, `->`.

Source file: <src/lib/meta.tester.ttester.fs>.

}}

```
}} ( a1 n1 n2 -- a2 ) "double-right-curly-bracket"
```

Return address *a2* of the *n1*,*n2* item of the 2-dimension array *a1*. Data stored row-wise.

See also: `2array`.

Source file: <src/lib/data.array.noble.fs>.

~

~fid-link

```
~fid-link ( a1 -- a2 ) "tilde-f-i-d-link"
```

A `field:`. Convert a UFIA data structure address *a1* to field address *a2*, which contains the address of the previous structure.

See also: `ufia`, `/ufia`, `/fid`.

Source file: <src/lib/dos.gplusdos.fs>.

~~

```
~~ ( -- ) "tilde-tilde"
```

Compile the name token, block and line of the current definition, and `(~~`.

`~~` is an `immediate` and `compile-only` word.

Origin: Gforth.

See also: `(~~`, `~~?`, `~~y`, `~~quit-key`, `~~resume-key`, `~~info`, `~~control` `~~before-info`, `~~after-info`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~?

```
~~? ( -- a ) "tilde-tilde-question"
```

A `variable`. *a* is the address of a cell containing a flag. When the flag is true, the debugging code compiled by `~~` is executed, else ignored. Its default value is true.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~after-info

```
~~after-info ( -- ) "tilde-tilde-after-info"
```

Executed at the end of the debugging code compiled by `~~`. `~~after-info` is a deferred word (see `defer`). Its default action is `~~restore-xy`, which restores the cursor coordinates.

See also: `~~before-info`, `~~restore-xy`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~before-info

```
~~before-info ( -- ) "tilde-tilde-before-info"
```

Executed at the start of the debugging code compiled by `~~`. `~~before-info` is a deferred word (see `defer`). Its default action is `~~save-xy`, which saves the cursor coordinates.

See also: `~~after-info`, `~~save-xy`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~control

```
~~control ( -- ) "tilde-tilde-control"
```

Keyboard control used by the debug points compiled by `~~`: If the contents of `~~quit-key` and `~~resume-key` are zero do nothing, else wait for a key press in an endless loop: If the pressed key equals the contents of `~~quit-key`, then execute `quit`; if the pressed key equals the contents of `~~resume-key`, then exit.

See also: `~~control?`, `~~press?`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~control?

```
~~control? ( -- f ) "tilde-tilde-control-question"
```

Is there any key to be checked by `~~control`?

`~~control?` is part of the `~~` tool.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~info

```
~~info ( -- ) "tilde-tilde-info"
```

Show the debugging info compiled by `~~` and the current contents of the data stack. `~~info` is a deferred word (see `defer`) whose default action is `(~~info`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~press?

```
~~press? ( c ca -- f ) "tilde-tilde-press-question"
```

Is the character stored at *ca* not zero and equal to *c*? `~~press?` is a factor of `~~control` used to check key presses, in the code compiled by `~~`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~quit-key

```
~~quit-key ( -- ca ) "tilde-tilde-quit-key"
```

A `cvariable`. *ca* is the address of a character containing the key code used to quit at the debugging points compiled by `~~`. If its value is not zero, `~~control` will wait for a key press in order to quit the debugging. Its default value is the code of 'q'.

See also: `~~resume-key`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~restore-xy

```
~~restore-xy ( -- ) "tilde-tilde-restore-x-y"
```

Restore the cursor coordinates. `~~restore-xy` is the default action of `~~after-info`.

`~~restore-xy` is part of the `~~` tool.

See also: `~~save-xy`, `~~before-info`, `~~xy-backup`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~resume-key

```
~~resume-key ( -- ca ) "tilde-tilde-resume-key"
```

A `cvariable`. *ca* is the address of a character containing the key code used to resume execution at the debugging points compiled by `~~`. If `~~resume-key` contains zero, `~~control` will not wait for a key. If `~~resume-key` contains $FF, `~~control` will wait for any key. Otherwise `~~control` will wait for the key stored at `~~resume-key`, whose default value is `bl`, the code of the space character.

See also: `~~quit-key`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~save-xy

```
~~save-xy ( -- ) "tilde-tilde-save-x-y"
```

Save the cursor coordinates. `~~save-xy` is the default action of `~~before-info`.

`~~save-xy` is part of the `~~` tool.

See also: `~~restore-xy`, `~~after-info`, `~~xy-backup`.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~xy-backup

```
~~xy-backup ( -- a ) "tilde-tilde-x-y-backup"
```

A `2variable`. *a* is the address of a double cell that holds cursor coordinates saved and restored by the default actions of `~~before-info` and `~~after-info`.

`~~xy-backup` is part of the `~~` tool.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

~~y

```
~~y ( -- ca ) "tilde-tilde-y"
```

A `cvariable`. *ca* is the address of a character containing the row the debugging information compiled by `~~` will be printed at. Its default value is zero.

Source file: <src/lib/tool.debug.tilde-tilde.fs>.

---

[3] In Forth-79, if *n2* is less than 1, no leading blanks are supplied.

[4] In Forth-83, if the number of characters required to display *n1* is greater than *n2*, an error condition exists, which depends on the system.

[5] In fig-Forth the size of each disk buffer was the size of a disk sector, usually 128 bytes by the time.

[6] In Forth-83, if the number of characters required to display *d* is greater than *n*, an error condition exists, which depends on the system.

[7] In Forth-79, if the number of characters required to display *u* is greater than *n*, no leading spaces are given.

[8] In Forth-83, if the number of characters required to display *u* is greater than *n*, an error condition exists, which depends on the system.