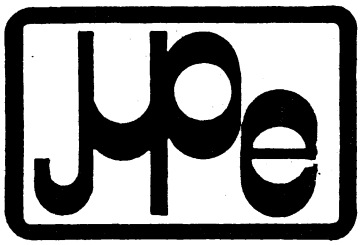


MicroProcessor Engineering Ltd
21 Hanley Road Shirley
Southampton SO1 5AP
Tel: 0703 780084

9900 FORTH

Laboratory Microsystems

and

MicroProcessor Engineering Ltd

Microprocessor Engineering Limited
21, Hanley Road, Shirley, Southampton, SO1 5AP

MicroProcessor Engineering 9900 FORTH

Rev 1.0

Microprocessor Engineering Limited
21, Hanley Road, Shirley, Southampton, SO1 5AP
0703-775482

MicroProcessor Engineering 9900 FORTH Rev 1.0 October 83

1. Introduction

9900 FORTH is a conversion of the Laboratory Microsystems Z-80 FORTH version 2.00. Without the help and assistance of Ray Duncan at Laboratory Microsystems this project would never have been started, let alone completed. It is our intention to keep this version upgraded as other Laboratory Microsystems FORTHs are enhanced.

2 9900 FORTH history

This FORTH actually arose out of a desire for a language that could easily be put into EPROM on our industrial boards. As we had had some experience with the Nautilis Systems range of FORTH cross-compilers, it became an obvious idea to write one for the 9900 on our Z-80 system. This we actually did more easily than we had expected. As we could not quite be bothered with blowing EPROMs all the time during testing, and the Marinchip 9900 system has a utility to read CP/M files we ended up by testing the code in RAM on the Marinchip. After that it would be so convenient to have a Laboratory Microsystems compatible FORTH on the 9900, and so here we are.

As a result we now have not just a disc-based FORTH for 9900 family computers running MDEX or NOS, we also have a 9900 family cross-compiler with both disc-based and ROMmable FORTH nuclei, and the capability of running any of the Nautilus cross-compilers issued through Laboratory Microsystems and many other sources.

3. Variations from Z-80 FORTH

Compatibility with the Laboratory Microsystems FORTHs is not 100%, as the file handling operations and system calls of MDEX and NOS are very different in some areas. In practice these difficulties are usually negligible. Most of the editor and utilities were converted without change.

The version of FORTH for the Powertran Cortex is supplied with two editors. EDIT80 is the original editor for users with an 80-column terminal on the serial port, and EDIT40 uses a 40-column screen such as is provided when using a television. In this mode screens are still 1k-bytes long,

but are presented as one screen of 32 lines of 32 characters, displayed in successive halves. The lines are renumbered to indicate which half is being displayed.

Microprocessor Engineering Limited
21, Hanley Road, Shirley, Southampton, SO1 5AP
0703-775482

9900 FORTH assembler Rev 1.0 October 83

1. Introduction

The 9900 FORTH assembler will cope with the full range of 9900 instructions. 9995 and 99105 instructions will be added in a future release. The assembler has been tested by compiling a complete FORTH nucleus. The syntax is modelled on that of the Texas Instruments assemblers, but, as is usual in FORTH assemblers, operands are presented before the opcode mnemonics.

2. How FORTH assemblers work

In this assembler, as in most other FORTH assemblers, the mnemonics are FORTH words which are executed at assembly time to compile data into the dictionary. Any modification of the opcode for addressing mode must be present before the opcode word executes, thus the requirement for operands before opcodes. For the same reason register values must be present before the register mode flags.

Examination of the source code in ASM99.SCR will show how this process is built up, and also shows one of the standard uses of the <BUILDS .. DOES> construct, which is used in nearly all modern FORTH assemblers.

Although this procedure appears very clumsy at first it has many advantages. The assembler is completely free form in that you can have as many instructions per line as you want, so the layout is now completely up to you. Because all the assembler words execute at assembly time you can include any amount of address arithmetic that is required. Reference to a predefined FORTH variable will return its address, so allowing that address to be used as an operand within a CODE definition.

3. Addressing mode syntax

The order of operands in the 9900 FORTH assembler is source, destination, opcode. Register numbers must occur before the register addressing mode operator, and addresses must occur before their addressing mode operators.

register addressing
source: n R where n is the register number. R0 through R15

are predefined, as are RA through RF.
destination: n ,R - ,R0 to ,R15 and ,RA to ,RF are predefined.
e.g 3 R ,R11 MOV is equivalent to MOV R3,R11.

register indirect
source: n *R
destination: n ,*R
e.g 4 *R 5 ,*R MOV is equivalent to MOV *R4,*R5

register indirect with auto-increment
source: n *R+
destination: n ,*R+ e.g. 6 *R+ 7 ,*R+ A is equivalent to A *R6+,*R7+

symbolic
source: addr @>
destination: addr ,@>
e.g FRED @> GEORGE ,@> MOV is equivalent to MOV @FRED,@GEORGE

indexed symbolic
source: addr reg @()
destination: addr reg ,@()
e.g. FRED 2 @() GEORGE 3 ,@() MOV is equivalent to MOV @FRED(2),@GEORGE(3)

immediate addressing
For TI processors immediate addressing is indicated by the opcode mnemonic, so no addressing mode indicator is required for this mode.
e.g 1 R 200 LI is equivalent to LI R1,200

CRU instructions
TB, SBO, and SBZ require an offset which lies in the range -128..127.
e.g. 15 SBZ is equivalent to SBZ 15

jump instuctions
All the jump instructions should be given the target address. Calculation of the actual 8-bit offset is performed within the word corresponding to the jump instruction. Within a CODE definition an address may be left on the stack several assembler instructions before it is used, as long as it does not interfere with the intervening instructions. In general jump instuctions are better dealt with by the assembler structuring words dealt with below. The pseudo-op $ has been defined to mean the current value of the dictionary pointer (location counter).
e.g $ 10 + JEQ is equivalent to JEQ $+10

Assembly level structures

Just as it possible to use the FORTH structures BEGIN ... UNTIL, IF .. ELSE .. THEN at high level, so we have defined a set of identically named structures at assembly level. These cope with the problem of forward references. They act on a set of words which force the construction of a jump instruction whose opcode depends on the given status register

condition.
e.g. ?EQ IF .... ELSE .... ENDIF
The IF clause will be executed if the equal status bit is true. Note that this leads to the construction of a JNE opcode i.e all the jump opcodes are the logical opposite of what is indicated by the condition shown before the IF. As the 9900 family does not have a complete set of arithmetic jump instructions ?LT and ?GT do not exist, but their complements do.

Condition flags

?EQ	if equal
?NE	if not equal
?H	if logically greater than (higher)
?HE	if logically greater or equal
?L	if logically less
?LE	if logically less or equal
?NC	if no carry
?C	if carry
?O	if overflow set
?EP	if even parity
?AGE	if arithmetically greater or equal
?ALE	if arithmetically less or equal
UC	on no condition

Structures

?xx IF ... ELSE ... ENDIF
The IF clause is executed if the condition is true, otherwise the ELSE clause is executed.

BEGIN ... ?xx UNTIL
The loop is executed until the given condition is true. Note that BEGIN ... UC UNTIL gives a loop that never exits.

BEGIN ... AGAIN
An indefinite loop

BEGIN ... ?xx WHILE ... REPEAT
The loop is repeated while the stated condition is true.

5. Pseudo-ops

RT	compiles 0B *R B == B *R11 == RT
$	compiles HERE, the location counter value
NOOP	compiles $ 2 + JMP == JMP $+2, a no-op

6. Using assembly language words

FORTH words should be defined as assembler words by using CODE and END-CODE instead of : and ;. Unlike semi-colon (;), END-CODE does not compile a branch back to the FORTH interpreter. In MicroProcessor Engineering versions of FORTH, this branch is an indirect one through a register which contains the address of NEXT. Which register is used is given elsewhere in the manual.

7 Cross-compiler version of the assembler.

Users of the version of this assembler for the Nautilus cross-compiler should be aware of some differences. The jump instructions support forward referencing, and the structuring words are not present. The structuring words will be added on a later release.

If you are using the cross-compiler on a host processor other than a 9900 be sure to use the phrase ASSEMBLER EVEN rather than just ASSEMBLER when starting a section of code without a dictionary header. This will force the dictionary pointer onto an even address before assembling any code - remember the error checking may be sparser than you like it.

9900 FORTH User's Manual

2.0 Distribution Diskette

9900 FORTH is distributed on 2 standard eight-inch, soft-sectored, single-density diskettes which contain the following files :

forth	Executable FORTH interpreter
forth.scr	FORTH screen storage
forthle	FORTH with precompiled line editor
editor.scr	Source text for screen editor
asm99 .scr	Source text for 9900 assembler
latest.doc	User notes *

In addition, if the floating point option was purchased, your disks will contain some or all of the following files:

hfpbase	hardware assisted nucleus
hfpforth	hardware assisted interpreter
sfpbase	software floating point nucleus
sfpforth	software floating point interpreter
float.scr	source for floating point routines

* If present, this file contains additions or modifications that occurred too late to be included in printed documentation.

2.1 Installation

3.0 Getting started with FORTH

After booting up the MDEX operating system execute FORTH

```
.FORTH
```

The interpreter will load into memory, then print the identifying message (may vary from this example depending on type of system and amount of memory):

```
9900 FORTH version 2
(c) 1982 Laboratory Microsystems
Executing under MDEX 3.x
Using file:  a:forth.scr
Dictionary space available: 6532 bytes

OK
```

FORTH is now ready for use. In order to view the contents of the screen file, you can use the INDEX function to display the first line of each screen as follows:

```
1 50 INDEX <return>
```

You can also view an abbreviated screen index, formatted four across, by entering:

```
QX  <return>
```

To view a given screen of text, type the number of that screen followed by the word "LIST". For example, to read the memory MAP program which is stored in screen #1, you would type:

```
1 LIST <return>
```

To compile a given screen of text, type the number of that screen followed by the word "LOAD". For example, to compile the MAP program which is stored in screen #1, enter:

```
1 LOAD <return>
```

Now to invoke the function itself, type:

```
MAP   <return>
```

It is often convenient to enter a number of commands on the same line, which are processed from left to right by the interpreter. For example:

```
1 LIST 1 LOAD MAP  <return>
```

3.1 Customizing the FORTH interpreter

The distribution FORTH file uses 12 kbytes of memory. Since MDEX occupies about 12 kbytes, you will need at least 32 kbytes of RAM in your system. FORTH does not automatically adjust its stacks and buffers to take advantage of additional memory when it is available. Youc an expand the FORTH runtime area up to a maximum of 48 kbytes (leaving 12 kbytes for MDEX in a 60 kbyte system) or change the number of screen buffers by using the utility in screen #2.

Wait for the system identification message and "ok" prompt. Then enter:

```
2 LOAD REALLOCATE
```

You will be asked to enter the total size in kbytes of the memory available for use by FORTH, then the number of screens to be buffered. Remember that each screen buffer subtracts 1 kbyte from the amount of dictionary space. Entries should be in the form of a decimal number followed by a carriage return. The program will calculate the new addresses of the buffers and stacks, set all necessary system pointers, then a cold start will be executed and you will again see the system identification message. At this point you can rewrite the executable image and make the changes permanent simply by entering:

```
SAVE FORTH <return>
```

3.2 Floating point systems

Users who purchased the hardware or software floating point extensions will receive an extra diskette containing the files xxxBASE, xxxFORTH, and FLOAT.SCR (where xxx=HFP for hardware assist and xxx=SFP for the full software version). xxxBASE is the assembled nucleus, containing the appropriate software or hardware floating point primitives. FLOAT.SCR contains the high level FORTH source code and conversion tables. xxxFORTH is the executable interpreter/compiler, it is created by loading the appropriate screens from FLOAT.SCR on top of xxxBASE.

xxxFORTH is complete in itself and is the only file you need to transfer to your working disks. The xxxBASE and .SCR file are provided for users who wish to make basic changes in the way the floating point software functions.

If you purchased the hardware assisted floating point software alone,

you will need to modify the system variables AP-DATA and AP-CONTROL to match the port addresses of your AMD 9511 board. If you bought the software and S-100 board as a package from Laboratory Microsystems the port addresses are set up properly when shipped.

The floating point nucleus contains the necessary machine primitives as well as modifications to E, NUMBER, and DLITERAL for floating point number conversion and compilation. In all other respects the floating point runtime package is software compatible with the basic FORTH.COM load module.

4.0 Writing Programs in FORTH

The FORTH runtime package is actually a remarkably compact interpreter, compiler, and virtual memory management system. Any command or sequence of commands may be executed directly from the keyboard or from disk storage. Programs in FORTH are "compiled" from combinations of existing commands (represented by vocabulary "words"), new commands as defined by the user, and control structures such as IF...ELSE...THEN or DO...LOOP. Usually new routines are developed interactively and incrementally at the console; the final version is saved on the disk where it may be invoked from the keyboard or by another program.

The beauty of FORTH lies in its extensibility and flexibility. New vocabulary words, functions, and even data types can be added to the language at will in either "high-level" or machine code. Programs are built up in the same manner as people organize their thinking; by successively creating new functions in terms of previously defined functions, forming hierarchies of increasing levels of abstraction. Language components which are unnecessary for a particular application can be deleted from the runtime package by the sophisticated user to minimize memory requirements.

If your experience in programming has been restricted to such sequentially organized languages as BASIC or FORTRAN, you will initially find both reading and writing programs in FORTH rather obscure. On one hand, FORTH can be practically self-documenting: the language lends itself well to bottom-up design; names of functions may be freely chosen to describe what they do; and embedded comments may be lengthy since they do not effect the size of compiled code. On the other hand, the most efficient and elegant FORTH programs maintain most working variables on the stack, which makes reading (and debugging) the code a real mental exercise.

You may find it profitable to study the demonstration programs supplied with 9900 FORTH as a guide to style. Read the glossary documentation and spend a few hours trying out each function and observing its effect on the stack. A number of useful manuals may be purchased from FORTH Interest Group (P. O. Box 1105, San Carlos, California, 94070) or from Mountain View Press (P. O. Box 4656,

Mountain View, California, 94040), The August 1980 issue of BYTE and the September 1981 and September 1982 issues of Dr. Dobb's Journal were completely devoted to FORTH tutorials and programs. Additional helpful articles may be found in the February 1982 issue of Dr. Dobb's Journal and the March 1982 issue of IEEE COMPUTER.

5.0 FORTH virtual memory

The FORTH interpreter uses a standard MDEX disk file to store screens of text. The disk file and RAM buffer are treated together as a large virtual memory. Physical sectors are read in from the disk as needed into buffers that are allocated on a least-recently-used basis. The user program can mark a sector as changed with the UPDATE function, this forces the block to be rewritten to the disk before releasing the buffer. Disc I/O is performed via the standard MDEX random access functions, each successive eight blocks of the data file corresponds to one FORTH screen. If the interpreter requests a screen which has not been previously allocated, the driver initializes it to spaces and flags it for a forced disk write before passing it to the program.

The number of screens which can be stored in the disk file is usually limited by the size of the media. On an eight-inch single-density diskette, there is room for slightly more than 200 screens. On larger media such as "Winchester" disk drives, each screen file may contain up to a maximum of 4095 screens.

The screen data file supplied on the distribution diskette is named FORTH.SCR. This is the file that is opened by default if the FORTH interpreter is invoked by the simple command

FORTH <return>

The user has the option of specifying other files for screen storage. This is done by giving the name of the desired file in the original command line. The screen storage file does NOT have to reside on the same drive as the interpreter. Format:

FORTH unit/filename.ext <return>

If the data file cannot be located, the interpreter will ask you whether you wish for a new file to be created, and will either establish the file and proceed or exit to MDEX depending on your response. It is suggested that you use the extension ".SCR" for all FORTH screen storage files, in order to locate them easily on a directory listing.

Example, to load the FORTH interpreter and use the file "MYFILE.SCR" on drive 2 for screen data storage, you would type:

FORTH 2/MYFILE.SCR <return>

5.1 Changing Screen Files

The word USING allows you to change to a different screen file from the one specified in the original command line without leaving the FORTH interpreter/compiler. This word may only be used as an entry from the keyboard; it may not be compiled or executed while LOADing. If the drive assignment is not given, the "current" disk drive will be used. The extension is not forced. USING's action is as follows: all updated buffers are written to the disk and the current screen file is closed. If the specified screen file cannot be located a warning message is displayed and the previous screen file is reopened, otherwise a success message is displayed and all disk buffers are cleared.

Example (the user's entry is underlined):

```
USING EDITOR   Current screen file:  1 editor

OK
```

5.2 Disk Transfer Errors

9.0 Interpreter locations of interest to users

The FORTH interpreter locations listed below are "guaranteed". Their significance will remain unchanged as new versions of 9900 FORTH are released, although their contents may vary.

location	significance
0100	jump to cold start
0104	jump to warm start
0108-010A	version number
010C	Address of topmost word in FORTH vocabulary
010E	Character to be used as backspace command (supplied as 08H)
0110	Initial user area pointer
0112	Initial data stack pointer
0114	Initial return stack pointer
011A	Initial value of WARNING
011C	Initial value of FENCE
011E	Initial dictionary pointer
0120	Initial VOC-LINK
012A	CFA of user's abort routine
012C	CFA of user's quit routine

10.0 Use of CPU registers in FORTH

FORTH	Reg	preservation rules
IP	9	preserved across FORTH words
RP	10	preserved across FORTH words
W	11	Sometimes output from NEXT. May be altered before JMP'ing to NEXT.
SP	8	Should be used only as a data stack across FORTH words. May be used within FORTH words if restored before transferring to NEXT.
	7	Pointer to next
	0,1,2,3	temporary usage during calculation

11.0 Comments on Console Input and Output

11.1 Special Control Characters

The code control/H (08H) is treated as a cursor backspace command, and causes the sequence <BS SP BS> to be sent back to the terminal.

The backspace command is specified by the contents of word 010EH, and may be changed by the user to a more convenient key. For example, to change the backspace command character to 7FH, you would enter:

```
DECIMAL 127 14 +ORIGIN !  ( resets the bootup literal )

SAVE FORTH  <return>      ( to create a new FORTH     file)
```

The character control/C (^C) causes FORTH to immediately terminate execution and return control to MDEX via a "warm boot". In general, we advise that you avoid using this control code as a means of exit from FORTH, as the usual sequence of closing the disk screen file is bypassed and unpredictable amounts of data may be lost.

All other control codes (including ^X, ^U, and ^S) have no special significance under 9900 FORTH at present.

11.2 Selection of Output Device

The vocabulary words CONSOLE and PRINTER have been added to allow the user to select the primary output device during program execution. They affect any characters transmitted directly or indirectly using the function EMIT. For example, the following command line would print a triad starting with screen #3 on the line printer, then return control to the system console:

```
PRINTER 3 TRIAD CONSOLE <return>
```

11.3 Configuration of 9900 FORTH for your terminal

Cursor control functions for the demonstration programs are loaded from screen #46. When you purchase the FORTH system, these functions are set up for a Televideo 950 CRT, and must be modified for your terminal type. Program screens are supplied in the distribution file FORTH.SCR for a number of common CRT models. If your terminal appears in the list below, you can quickly customize the system as follows:

```
A>FORTHLE    (wait for system id and "OK", then type)

EDITOR  nn  46  COPY  FLUSH  BYE  <return>
```

Where "nn" is the appropriate screen number from the list below:

60	Beehive B-100
61	Televideo family
62	Lear ADM 3
63	Soroq IQ 120
64	Intertube
65	Hazeltine 15xx
66	IBM 3101
67	Synertek KTM 3/80
68	DEC VT-52
69	Micro-Term ACT-IVB
70	ADDS Regent series
71	Xerox Personal Computer
72	Zenith Z-19 or Heath H-19
73	HP 2621A
74	ADDS Viewpoint

If your terminal does not appear in the table above, you may use the line editor to make the required changes, since it is not terminal dependent. Carefully read the line editor documentation, then enter the FORTH interpreter and follow the sequence below.

```
A>FORTHLE                    (wait for system id and "OK")
```

```
EDITOR  <return>                ( select editor vocabulary )
46 LIST <return>                ( display screen #46 and )
                                ( prepare to edit it )
TOP <return>                    ( move the cursor to start )
                                ( of first line of screen )

                                ( now use the P function of )
                                ( the editor to change the )
                                ( contents of desired lines )
```

... for example, to change line 4 so that the CLEARSCREEN function transmits an ASCII formfeed code, you enter:

```
4 P  : CLEARSCREEN 12 EMIT ;  <return>
```

... when all necessary lines have been changed, type:

```
FLUSH BYE
```

The updated sectors are written to the disk and the interpreter exits. If things go awry during editing and you want to start over, you can use the COPY function to restore screen #46 from screen #61.

Similar changes will have to be made to screen #8 in the file EDITOR.SCR for the full screen editor to work correctly. The cursor control functions supplied in FORTH.SCR can be used as a model. See the separate documentation section for the screen editor.

12.0 Screen Auto-load

Z-80 FORTH has the capability of automatically loading a user defined screen at cold start time. This facility may be used to immediately compile and start up an application program, thereby protecting the FORTH interpreter/compiler from access by the computer operator. The variable BOOT-SCREEN must be initialized to the desired screen number and a new FORTH file created using the SAVE command. If the value of BOOT-SCREEN is zero (as it is on the distribution disk), FORTH will skip past the screen load, print the system ID, and prepare to interpret from the keyboard in the usual manner.

After editing your startup commands into the screen file, you can test them by setting the value of BOOT-SCREEN to the screen number then executing COLD from the keyboard. When satisfied with the result, enter: SAVE filename <return> to create a new file with BOOT-SCREEN initialized to the startup screen number. For example, to create a FORTH that will load the contents of screen #16 at cold start time:

```
A>FORTH          (wait for system ID and "OK", then enter)

DECIMAL  16  BOOT-SCREEN  !  <return>
```

SAVE FORTH <return>

13.0 Creating FORTH application programs

Two guaranteed locations are provided in the boot-up literals to help you create custom precompiled FORTH application programs. By storing the CFA of your main routine into the variable "UQUIT" at location 012Ch, FORTH is forced directly into your application program at startup and will not interpret from the keyboard. You can also place the CFA of a custom error handling routine into the location "UABORT" at 012Ah. For example, if the source text for your program can be compiled by LOADing screen #40, and the main control routine is named 'MY-PROGRAM', the following sequence would create a new file that executes your application automatically.

```
DECIMAL
40 LOAD                              ( compile user's program)
' MY-PROGRAM CFA 44 +ORIGIN !        ( store CFA into UQUIT )
SAVE FORTH                           ( create a new      file )
```

For another example, see screen #1 of the file EDITOR.SCR which uses this capability to create EDITOR.

Please note that the 9900 FORTH file is proprietary software and is sold for use by a single purchaser. However, we wish to strongly encourage the further development of programming tools written in FORTH. If you wish to create and resell custom applications that contain 9900 FORTH as part of the run-time module, please contact us!

FORTH Quick Reference

n,n1 ...	16 bit signed number
d,d1 ...	32 bit signed number
u	16 bit unsigned number
addr	memory address
b	8 bit byte
c	ASCII character
f	boolean flag

Stack inputs and outputs are shown, top of s right. Not all vocabulary items are given in this summary. See the manual's Glossary section for complete explanations.

STACK MANIPULATION

-DUP	n -> n ?	duplicate n if not zero
2DROP	d ->	discard double number
2DUP	d -> d d	duplicate double number
2SWAP	d1 d2 -> d2 d1	reverse top two double numbers
>R	n ->	move to "return stack"
DEPTH	-> n	leave # of items on stack
DROP	n ->	discard single number
DUP	n -> n n	duplicate single number
OVER	n1 n2 -> n1 n2 n1	copy 2nd item to top
PICK	n1 -> n2	copy n1-th item to top
R	-> n	copy from "return stack"
R>	-> n	move from "return stack"
ROLL	n1 -> n2	rotate n1-th item to top
ROT	n1 n2 n3 -> n2 n3 n1	rotate 3rd item to top
SWAP	n1 n2 -> n2 n1	reverse top two single numbers

NUMBER BASES

BINARY	->	set binary system base
DECIMAL	->	set decimal base
HEX	->	set hexadecimal base
BASE	-> addr	system variable containing current number base

ARITHMETIC AND LOGICAL

*	n1 n2 -> product	multiply
*/	n1 n2 n3 -> n4	n4 = n1 * n2 / n3
*/MOD	n1 n2 n3 -> n4 n5	n4 remainder, n5 quotient
+	n1 n2 -> sum	addition
+-	n1 n2 -> n3	apply sign of n2 to n1
-	n1 n2 -> difference	subtraction

/	n1 n2 -> quotient	division
/MOD	n1 n2 -> n3 n4	n3 remainder, n4 quotient
1+	n1 -> n2	increment by one
1-	n1 -> n2	decrement by one
2*	n1 -> n2	signed multiply by two
2+	n1 -> n2	increment by two
2-	n1 -> n2	decrement by two
2/	n1 -> n2	signed divide by two
ABS	n -> absolute	absolute value of single
AND	u1 u2 -> and	bitwise logical and
D+	d1 d2 -> sum	double precision add
D+-	d1 n -> d2	apply sign of n to d1
D-	d1 d2 -> difference	double precision subtract
DABS	d -> absolute	absolute value of double
DMINUS	d -> -d	change sign
M*	n1 n2 -> d	double precision product
M/	d n1 -> n2 n3	mixed magnitude divide
M/MOD	ud1 u2 -> u3 u4	unsigned mixed divide
MAX	n1 n2 -> max	leave larger number
MIN	n1 n2 -> min	leave smaller number
MINUS	n -> -n	change sign
MOD	n1 n2 -> mod	leave remainder of n1/n2
OR	u1 u2 -> or	bitwise logical or
S->D	n -> d	sign extend to double
TOGGLE	addr b ->	complement addr by b
U*	u1 u2 -> ud	unsigned multiply
U/	ud1 u1 -> urem uquot	unsigned divide
XOR	u1 u2 -> xor	bitwise exclusive or

COMPARISON

0<	n -> f	true if n < zero
0=	n -> f	true if n = zero
0>	n -> f	true if n > zero
<	n1 n2 -> f	true if n1 < n2
=	n1 n2 -> f	true if n1 = n2
>	n1 n2 -> f	true if n1 > n2
D<	d1 d2 -> f	true if d1 < d2
D=	d1 d2 -> f	true if d1 = d2
D>	d1 d2 -> f	true if d1 > d2
U<	u1 u2 -> f	true if u1 < u2

MEMORY

!	n addr ->	store single at addr
+!	n addr ->	add n to word at address
+ORIGIN	n -> addr	offset from origin
,	n ->	store word into dictionary
2!	d addr ->	store double at addr

2@	addr -> d	fetch double number
@	addr -> n	fetch single number
ALLOT	n ->	reserve dictionary space
C!	b addr ->	store byte at addr
C,	b ->	store byte into dictionary
C@	addr -> b	fetch byte

DISK ACCESS

-->	->	continue with next screen
BLK	-> addr	variable, current block
BLOCK	n -> addr	read disk block n to addr
BOOT-SCREEN	-> addr	variable, cold boot block
DISK-ERROR	-> addr	variable, disk status
EMPTY-BUFFERS	->	clear disk buffers
FLUSH	->	force write updated buffers
LOAD	n ->	compile/execute screen n
R/W	addr block f ->	flag: 0=read, 1=write
SCR	-> addr	variable, current screen
SCREEN-FCB	-> addr	screen file control block
THRU	n1 n2 ->	LOAD screens n1 through n2
UPDATE	->	mark last BLOCK for write

MDEX SYSTEM INTERFACE

?TERMINAL	-> f	true if key depressed
BYE	->	return to
CONSOLE	->	select video for output
CR	->	send CR-LF sequence
DIR	->	show disk directory
EMIT	c ->	send one character
JSYS	parameter block address -> status	MDEX service request
KEY	-> c	read one ASCII char
PRINTER	->	select printer for output
SAVE	->	write memory image

OUTPUT FORMATTING

#	d -> d	convert next digit
#>	d -> addr u	make string ready for TYPE
#S	d -> 0 0	convert rest of digits
.	n ->	print signed number
."		print string up to "
.CPU	->	print CPU identity
.FCB	addr ->	print filename from fcb
.LINE	line scr ->	print line of text
.R	n fieldwidth ->	print right-justified

.STACK	->	display parameter stack
<#	->	start output string
?	addr ->	print contents of addr
BL	-> blank	leave ASCII blank
C/L	-> n	leave characters per line
D.	d ->	print double number
D.R	d fieldwidth ->	print double right justified
HOLD	c ->	insert char in output string
ID.	addr ->	print definition's name
IN	-> addr	variable, input stream pointer
MESSAGE	n ->	print line n of screen 4+
OUT	-> addr	variable, output char. count
SIGN	n d -> d	insert sign into output string
SPACE	->	print a space
SPACES	n ->	print n spaces
TYPE	addr u ->	type string of u characters
U.	u ->	print unsigned number

CONVERSION AND STRING HANDLING

-TRAILING	addr n1 -> addr n2	suppress trailing blanks
BLANKS	addr n ->	store n blanks at addr
CMOVE	addr1 addr2 len ->	move byte string
COUNT	addr1 -> addr2 n	followed by TYPE
DIGIT	c n1 -> ff OR n2 tf	convert c by base n1
ENCLOSE	adr c -> adr n1 n2 n3	text scanning primitive
ERASE	addr n ->	clear n bytes to zeros
EXPECT	addr count ->	read string from terminal
FILL	addr n b ->	store n bytes of b at addr
MATCH	n1 n2 n3 n4 -> f n5	string matching for editors
NUMBER	addr -> d	convert to double number
QUERY	->	input string to TIB
S=	addr1 addr2 len -> f	true if strings identical
WORD	c ---	read input to delimiter c

DISPLAY

INDEX	n1 n2 ->	title lines for scr n1 to n2
LIST	n ->	display screen n
QX	->	quick screen index
SHOW	n1 n2 ->	scr n1 thru n2 on printer
TRIAD	n ->	display 3 screens from n
VLIST	->	display CONTEXT vocabulary

Contents of file FORTH.SCR on FORTH distribution disk

Screen(s)	Contents
Ø	System identification and credits.
1	Memory allocation map display. To compile type: 1 LOAD <return>. Then to view memory addresses of system buffers and stacks type: MAP <return>.
2	FORTH system memory allocation program. Allows you to modify the dictionary size and number of disk buffers. To compile enter: 2 LOAD <return>. To run the program enter: REALLOCATE <return>. After you see the system ID again, enter: SAVE FORTH <return> to write out the executable image to the disk and make the changes permanent.
4-5	Reserved for Interpreter/Compiler error messages.
6	Reserved for Assembler error messages.
7	Reserved for Data Base Management error messages.
8	Integer 1-dimensional and 2-dimensional array definitions. During compilation, "n ARRAY xxx" creates an array of n words named xxx, and initializes it to zeros. During execution, "n xxx" leaves the indexed address into the array on the top of the stack. During compilation, "x y 2ARRAY xxx" creates an array with dimensions x by y named xxx and initializes it to zeros. During execution, "x y xxx" returns an indexed address into the array for coordinates (x,y). For example, "4 1Ø xxx @" would fetch the contents of coordinate (4,1Ø) in array xxx. See also multi-dimensional array definition in screens 57-59.
9	Random Number Generator, by J. E. Rickenbacker. In order to compile the random function type: 9 LOAD <return>. Then any number followed by the word RANDOM will return a random value between zero and that number. For example, 5Ø RANDOM will return a value between Ø and 49. A short program to test this

fig-FORTH Portable Line Editor

The Editor supplied with 9900 FORTH has been adapted from the portable Editor written by W. Ragsdale and placed in the public domain through FORTH Interest Group.

In screen-oriented commands such as COPY or LIST, 'n' refers to a screen number. In line oriented commands such as T or D, 'n' is a line number 0-15. 'Text' indicates a string of arbitrary length, with or without embedded blanks, terminated by a carriage return. 'PAD' refers to a buffer which holds a single line of text, it serves as a work area from which data may be copied to or from desired locations in the current editing screen. Each command must end with a carriage return, when it is successfully executed the system replies 'OK'.

Note that after loading the Editor, the current screen and current cursor position are undefined until a LIST and then a TOP command are executed.

Command	Explanation
10 LOAD	Load and compile Editor commands from Forth screen file.
EDITOR	Select Editor vocabulary and make commands available for use.
FORTH	Leave Editor vocabulary and return to normal Forth vocabulary.
n LIST	Display screen 'n' and select it for future editing.
L	Display currently selected program screen and location of cursor.
TOP	Home cursor to upper left corner of screen.
n CLEAR	Clear screen 'n' to blanks.
n1 n2 COPY	Copy the contents of screen 'n1' to screen 'n2'. The contents of screen 'n1' are unchanged, the previous contents of screen 'n2' are lost.
n T	Display the contents of line 'n' of the current screen, also leave a copy of the line in PAD. The cursor is left at the start of the selected line.
n H	Copy the contents of line 'n' to PAD, leave the line unchanged.

n E	Erase the contents of line 'n'.
n D	Delete the contents of line 'n', all higher numbered lines are 'rolled up' one line. The contents of line 15 are duplicated. In addition, the text of the line that was deleted is saved in PAD.
n S	Spread the text of the current screen, making line 'n' blank, and rolling down the remainder of the screen by one line. The previous contents of line 15 are lost.
n R	Replace the contents of line 'n' with the text in PAD.
n P text	Replace the contents of line 'n' with the following text.
n I	Insert the text at PAD onto line 'n', all higher numbered lines are rolled down by one line, the previous contents of line 15 are lost.
n M	Move the cursor across the number of characters given by the signed amount 'n'. A negative number moves the cursor left, a positive number moves it to the right.
B	Back up the cursor by the text in PAD.
F text	Find the next occurrence of the given text, types the line on which the text was found. Sets the cursor at the end of the matching text, and leaves a copy of the matched text in PAD. The search begins at the current cursor position and continues until a match is found or the end of the screen is encountered. If there is no match, an error message is issued and the cursor is left at the start of the screen.
N	Find the next match of the current contents of PAD. Otherwise works like the F command. Can be used in conjunction with the F command to successively search for the same word or phrase.
X text	Find and delete the following text, search is limited to the current line. Leaves the cursor at the point of deletion. The right end of the line is blank filled.
C text	Spread the current line at the cursor and copy in the following text. Characters that spill off the right end of the line are lost. The cursor is left at the end of the inserted text.
TILL text	Deletes contents of the current line up to and including the specified text.
FLUSH	Write all updated blocks to disk.

Mini Screen Editor

The Mini Screen Editor supplied with Z-80 FORTH is derived from a simple example published in Forth Dimensions, Vol. II, No. 3, page 83. A full screen editor is supplied in the file EDITOR.SCR and is described in a separate documentation section.

The source text for the mini screen editor is supplied in screens #20 to #22. Also loaded during compilation are cursor control functions from screen #46 and the case control statement in screen #52. As supplied on the distribution disk, the cursor functions are configured for a Televideo 950 CRT. In order to use the editor, you will need to modify the routines CLEARSCREEN (clear screen and home cursor), GOTOXY (move cursor to address x,y), and CLREOS (clear screen from current cursor position to end).

The editor's cursor movement keyboard commands are defined by constants in screen #20, and may be readily modified to make them more convenient on your terminal.

Command	Explanation
20 LOAD	Load and compile mini screen editor from Forth storage file.
n EDIT	Read screen 'n' and make it available for editing. When your cursor control functions are properly set up, you will observe the following sequence of events: - screen is cleared and cursor jumps to home position. - title 'Screen # n ' is printed in upper left corner. - current contents of screen are displayed, with line numbers 0-15 at left. - cursor jumps to line 18 and four lines giving the keyboard commands are displayed. - cursor is placed at the first character on the first line of screen contents. You may now freely move the cursor throughout the screen using the commands given below, modifying and entering new text as desired. The display always shows the exact updated contents of the text block.
<ESC>	Pushing the Escape key causes the editor to exit. The cursor will jump to line 18, the screen is cleared from line 18 to end, and the message 'OK' is displayed. Be sure to use the command 'FLUSH' to ensure that your edited text is written to the disk.
<RETURN>	The Carriage Return key (New Line key on some terminals) causes the cursor to jump from its current position to the start of the next text line. The

contents of the screen are unchanged.

<Ctl/R> Control/R (^R) causes the cursor to move to the right one position. If the cursor is at the end of a line, it will be moved to the beginning of the next line. The contents of the screen are unchanged.

<Ctl/L> Control/L (^L) causes the cursor to move to the left one position. If the cursor is at the start of a line, it will be moved to the end of the previous line. The contents of the screen are unchanged.

<Ctl/D> Control/D (^D) moves the cursor down one line. If the cursor is already within the last text line, it will be moved to the end of the line. The contents of the screen are unchanged.

<Ctl/U> Control/U (^U) moves the cursor up one line. If the cursor is already within the first text line, it will be moved to the left end of the line. The contents of the screen are unchanged.

<Ctl/E> Control/E (^E) sets the entire text area to blanks, and leaves the cursor at the beginning of the first line.

<DEL> The delete key backs up the cursor (same effect as Ctl/L).

<Ctl/I> Control/I (^I) functions as a horizontal tab. For each depression of this key, the cursor will jump to the right 8 spaces. If the cursor is already at the end of a line, it will be moved to the beginning of the next line.

FORTH Full Screen Editor

The file EDITOR.SCR contains the source text for a powerful high level FORTH screen based editor. This editor is compatible with FORTH version 2.xx. The program is terminal dependent for the CLEARSCREEN and GOTOXY functions in screen #8 of the source listing, which must be modified appropriately for your CRT.

The capabilities of this screen editor include:

* full visual editing with up to date screen contents displayed at all times
* independent command and edit modes
* single and multiple screen copy utilities
* screen file and disk directories
* character insert
* character delete
* word delete
* line insert
* line delete
* line erase
* separate "line stack" for temporary storage of text
* screen erase
* forward and back tabbing to tab stops or words, user may set tab stop width
* string search and replace
* cursor movement in any direction
* print menu of available control codes
* go to next screen or previous screen
* while editing a screen, all changes may be discarded at any time and original contents of screen restored

Screen #8 of the full screen editor source text contains the functions CLEARSCREEN and GOTOXY. When you receive the disk these are set up for a Televideo 950 terminal. These functions are terminal dependent and must be modified for your particular CRT model.

CLEARSCREEN — Stack effect: 0 -> 0
Clears the terminal screen and leaves the cursor in the home position (upper left corner of screen). On some models of CRT this requires two separate escape sequences. In addition, some terminals require enough time to complete this function that several rubouts must be sent after the escape sequence to prevent text characters being transmitted before the terminal is ready.

GOTOXY — Stack effect: 2 -> 0
Calling sequence: x y GOTOXY
Leaves the cursor in the position on the screen defined by the x (column) and y (row) coordinate on the stack. (x,y)=(0,0) is assumed to be the upper left corner of the screen. X must be in the range 0-79, y in the range 0-23.

The file FORTHLE.COM on the disk is FORTH with a built-in fig-FORTH line editor. Use this editor to modify the terminal dependent functions in screen #8 for your terminal. Detailed instructions for the line editor are in the FORTH user's manual. Of course, you would initiate this process by typing:

```
FORTHLE  EDITOR.SCR  <return>

(after system ID and "OK", type: )

EDITOR <return>          (selects the editor vocabulary)
```

proceed using the "P" function of the line editor to modify the CLEARSCREEN and GOTOXY definitions. Be sure to type FLUSH and BYE when finished to force your changes out to the disk file.

After modifying the terminal dependent functions appropriately, you can create a customized screen editor by typing:

```
FORTH  EDITOR.SCR  <return>

3 LOAD
```

```
Wait... loading screen editor

.LINE isn't unique

Screen editor compilation completed.
12177 bytes left in dictionary.
ok:

SAVE EDITOR
```

You can now transfer the file EDITOR onto all of your FORTH working disks. You will have no further need for the files FORTHLE, EDITBASE, and EDITOR.SCR unless you change to a different type of CRT terminal or you wish to make basic changes in the operation of the screen editor program.

If the compilation process aborts, you may have made an error in editing the terminal dependent screen. The most common error made by the new user is the omission of either a semicolon ";" or a forward arrow "-->".

Using the screen editor:

First you must invoke the EDITOR load module, specifying the target screen file name in the command line. If no screen file is specified, it will default to the file FORTH.SCR. Example:

```
EDITOR  FLOAT.SCR  <return>
```

After the Editor locates and opens the screen file, it enters command mode and waits for input from the keyboard. In either command mode or edit mode, you can type control/V (^V) at any time to display the list of available command or control keys.

Control codes for the edit mode are defined by constants in screens #30-32 of the source code listing. The user can change these at will to make them more convenient for his terminal. The function code menu automatically references the constants for its display so in most cases the menu module program code will not need to be modified. The list below gives the command codes as they are defined in this source listing.

Command Mode Control Codes

C	Copy a single screen of text within the file. The user is prompted for the source and destination screen number.
D	Display directory for selected disk drive.
E	Begin editing. The user is prompted for the number of the first screen to be edited.
I	Display index to current screen file.
M	Move a set of screens within the current file. The user is prompted for the number of the first source screen, the last source screen, and the first destination screen. Depending on the direction of transfer, the routine begins at the appropriate end of the screen range so that when screen numbers overlap no data will be destroyed.
T	Set width of tab stops.
U	Change screen files. All updated buffers are written to the disk. The user is prompted for the drive assignment and the name of the new screen file. If the new file cannot be located, the previous screen file is reopened.
X	Leave the screen editor, enter normal FORTH interpretive move. This can be used to compile and test program text that is being edited.
ESC	Write all updated buffers to disk, close the screen file, and return control to MDEX

When a screen number is requested in command mode, you must enter a decimal number (any number of digits) followed by a carriage return. You may use the usual FORTH backspace code to delete incorrect digits.

Edit Mode Control Codes

[cursor commands]

^L	Move cursor left. If cursor is already at beginning of a line, move it to the end of the previous line. Contents of screen are unchanged.
^R	Move cursor right. If cursor is already at the end of a line, move it to the start of the next line. Contents of screen are unchanged.
^U	Move cursor up. If cursor is already in the top line of the screen, move it to the left margin. Contents of screen are unchanged.
^D	Move cursor down. If cursor is already in the bottom line of the screen, move it to the right margin. Contents of screen are unchanged.
^H	Move cursor to home position (upper left corner of editing area). Contents of screen are unchanged.
CR	Move cursor to beginning of next line. If cursor is already in the last line, move it to the right margin.
^I	Move cursor right to next tab position. If cursor is already in the last tab position of the current line, move it to the beginning of the next line. Contents of screen are unchanged.
^O	Move cursor left to next tab position. If cursor is already at the left margin, move it to the last tab position of the previous line. Contents of screen are unchanged.

[word commands]

^F	Move cursor forward (right) to beginning of next word.
^B	Move cursor back (left) to beginning of previous word.
^C	Delete the word to the right of the cursor. Discard any trailing spaces and bring the next word and the remainder of the line up to the cursor.

[line commands]

^X	Erase current line (the line containing the cursor).

^K — Delete current line (the line containing the cursor), moving all subsequent text lines up one position and making line 15 blank.

^S — Insert a blank line at the cursor, moving all lines down, contents of previous line 15 are lost.

^Y — Copy current line (i.e. the line containing the cursor) to separate holding area. Once ^Y has been used, the contents of the extra line buffer are displayed at the bottom of the screen between a pair of braces: { ..... }. The contents of the screen are unchanged.

^T — Get contents of extra line buffer to current line (the line containing the cursor). The contents of the extra line buffer are unchanged, the previous contents of the current line are lost.

[character commands]

^G — Delete the character under the cursor, moving the rest of the line left. Other lines are not affected.

DEL — Delete the character to the left of the cursor, moving the cursor and the remainder of the line to the left. Other lines are not affected.

^A — Insert a space at the cursor, moving the rest of the current line to the right. Characters shifted off the right end of the line are lost. Other lines are not affected.

^Q — Enter insert mode. All subsequent characters are inserted at the current cursor position and the rest of the line is moved to the right. Characters shifted off the right end of the line are lost. Insert mode terminates on a carriage return or any control code.

[screen commands]

^E	Erase entire screen and leave cursor in the upper left corner of the screen.
^N	Write current screen to disk and go to next screen.
^P	Write current screen to disk and go to previous screen.
^W	Force the current screen contents to be written to disk, then return for further editing.
^Z	Discard all changes and restore screen as it was before any editing.

[miscellaneous commands]

^V	Display menu of control codes. Screen contents are unchanged.
ESC	Leave edit mode, return to command mode.

Disk File Interface

Source code for a disk file and record interface is found in screens #23-30 of the distribution file FORTH.SCR . This set of extensions is provided to give the 9900 FORTH user complete control over MDEX standard disk files and access to any record within such files, bypassing the FORTH internal disk drivers and virtual memory management. The functions are compatible on the high level with the disk file interface provided with Laboratory Microsystems PC/FORTH and 8086 FORTH packages. The operations have minimal error and syntax checking and must be used with caution; they are not recommended for use by beginners in FORTH.

As each file control block is defined, it is allocated a special control byte and a dedicated disk buffer. When record operations are performed referencing a given file control block, the buffer address is automatically passed to and provided back to the user if the operation is successful. The number of file control blocks which may be defined (and consequently the maximum number of files which may be open at any one time) is limited only by the amount of dictionary space available.

Complete descriptions of each file and record operation are provided below in the same format used for the manual's Glossary section. The status-code returned by all file operations is in the range 0-3 if operation successful, and 255 if operation failed. The status-code returned by all record operations is either the disk buffer address if operation successful, or zero if operation failed.

?BUFFER-ADDR fcb-address --- disk-buffer-address
Given the address of a file-control-block, return the address of its dedicated disk transfer area.

CLOSE-FILE fcb-address --- status-code
Close an MDEX disk file, updating the disk directory if necessary. Release the file control block.

DELETE-FILE fcb-address --- status-code
Delete the MDEX disk file named in the file control block from the disk directory.

FCB ---
Used in the form:
FCB fcb-name
Allocate and initialize a file control block and disk buffer. Subsequent execution of the control-block name will leave the address of the file control block on top of the stack (see examples).

FILENAME fcb-address ---

Used in the form:

(fcb-address placed on stack) FILENAME file-spec

Format a file specification of the form unit:filename.ext into the indicated file control block. If the drive assignment is not included, the default drive will be used. If the extension is not specified it is set to all blanks.

INPUT-FILENAME fcb-address ---

Queries the console operator for input of a drive name, file name, and extension identifying a specific MDEX disk file. Formats the supplied information into the indicated file control block.

OPEN-FILE fcb-address --- status-code

Find and make available the file named in the indicated file control block for further record operations. The filename must have been previously formatted into the control block using FILENAME or INPUT-FILENAME.

PARSE-FILENM fcb-address file-spec-address ---

Parse a file specification of the form unit:filename.ext into the indicated file control block. Initialize all reserved areas. If a drive specification is not included, the "current" drive will be used.

READ-RANDOM fcb-address record-number --- status-code

Read the designated record from the disk into memory, returning either the memory address of the record or zero if the record does not exist.

READ-SEQ fcb-address --- status-code

Read the next sequential record in the file into memory, returning either the memory address of the record, or zero if end of file has been reached.

WRITE-RANDOM fcb-address record-number --- status-code
Write the designated record from memory to the disk, returning the memory address of the record if operation successful, or zero if no room left on the disk.

WRITE-SEQ fcb-address --- status-code
Write the sequential record from memory to the disk, returning the memory address of the record if operation successful, or zero if no room left on the disk.

A simple example using the interface functions. Open a file named USER.TMP, read record zero into memory, then print word zero of the record on the terminal.

```
FCB FILE1                       ( establish file control block )
FILE1 FILENAME USER.TMP         ( place filename in block )
: DEMO    FILE1 OPEN-FILE
          255 = IF ." File not found " ABORT
          THEN  FILE1 0 READ-RANDOM
          DUP 0= IF ." Record does not exist " ABORT
          THEN   @ .
          FILE1 CLOSE-FILE ;
```

"return" or the count of characters has been received. One or more nulls are added at the end of the text.

FENCE --- addr U

A user variable containing an address below which FORGETting is trapped. To forget below this point the user must alter the contents of FENCE.

FILL addr quan b ---

Fill memory at the address with the specified quantity of bytes b.

FIRST --- n

A constant that leaves the address of the first (lowest) block buffer.

FLD --- addr U

A user variable for control of number output field width.

FLUSH

Write all updated disk buffers to the disk. Should be used after editing, before dismounting a disk, or before exiting FORTH.

FORGET E

Executed in the form:

FORGET cccc

Deletes definition named cccc from the dictionary with all entries physically following it. In 9900 FORTH and fig-FORTH, an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same.

FORTH P

The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. Until additional user vocabularies are defined, new user definitions become a part of FORTH. FORTH is immediate, so it will execute during the creation of a colon definition, to select this vocabulary at compile-time.

HERE --- addr

Leave the address of the next available dictionary location.

HEX

Set the numeric conversion base to sixteen (hexadecimal).

HLD --- addr

A user variable that holds the address of the latest character

SCR --- addr U

A user variable containing the screen number most recently referenced by LIST.

SCREEN-FCB --- addr Z

Leave the address of the standard file control block used by FORTH to manipulate the screen file.

SEC/BLK --- n

Constant giving the number of physical disk sectors per FORTH standard 1024 byte block.

SIGN n d --- d

Stores an ASCII "-" sign just before a converted numeric output string in the text output buffer when n is negative. n is discarded, but double number d is maintained. Must be used between <# and #>.

SMUDGE

Used during word definition to toggle the "smudge bit" in a definition's name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without an error.

SP@ --- addr

A computer dependent procedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. E. g. 1 2 SP@ @ . . . would type 2 2 1 .

SP!

A computer dependent procedure to initialize the stack pointer register from S0.

SPACE

Transmit an ASCII blank to the output device.

FORTH GRAPHICS EXTENSIONS

Rev 1.0

by RICHARD ROBERTS

May 1984

MicroProcessor Engineering Ltd
21 Hanley Road Shirley
Southampton SO1 5AP
Tel: 0703 780084

MPE-FORTH graphics extensions Rev 1.0 May 1984

CONTENTS

Microprocessor Engineering Limited
21, Hanley Road, Shirley, Southampton, SO1 5AP
0703-780084

MPE-FORTH graphics extensions

1. FORTH Graphics extension

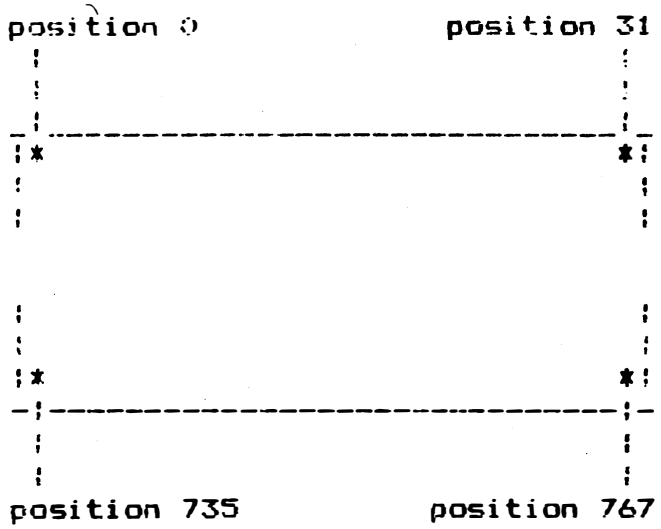
The graphics extension on the 9900 FORTH running on the CORTEX allows full use of the the 9928/29 graphics ability. The MDEX file GFORTH contains a revised version of the 9900 FORTH which supports a full set of graphic commands, including line, arc and circle drawing words, full sprite manipulation etc. Along with the graphics version of the FORTH you will find the file GDEMO.SCR, which contains a demonstration of most of the graphics commands available, (This can be run by typing GFORTH GDEMO.SCR 7 load).

All graphics words exist in the vocabulary GRAPHICS. It will also be noticed while using GFORTH that it is case insensitive, i.e. case=CASE etc.

2. Graphic modes. Features and video memory map.

2.1. Graphics 1 (Mode 0)

In this mode the pattern plane is divided into 32*24 character positions as follows:



It is possible to have full use of sprites (q.v.) in this mode and every seventh character beginning a different colour. The only software supplied for use in this mode is G1COL this word sets all the colours to the current text colours as it is assumed that the user will use mode 1 (Graphics 2) because of it's greater possible resolution rather than mode 0. The FORTH word GRAPH1 places the system into this mode.

Memory map for Mode 0

0000	Sprite pattern generator table
0400	Name table
0700	Colour table
0780	Sprite name table
0800	Pattern generator table

2.2. Graphics 2 (Mode 1)

This mode is the high-res graphics mode allowing the user to have full control over a 256*192 bit mapped screen. It is possible to mix graphics and text in this mode with the text having a resolution of 32*24 characters, the character positions are the same as for mode 0.

All future references to positions on the screen when in this mode are refered to x and y, x being across y being down, with location 0 0 as the top left hand corner.

Colour definition in this mode is any two colours per eight horizontal pixels. The FORTH word GRAPH2 places the system in this mode. Software is supplied to draw lines, circles, arcs, points, move sprites, change colour of new points, move blocks of 8*8 characters about the screen, etc.

Memory map for Mode 1

0000	Pattern generator table
1800	Pattern name table
1B00	ASCII table lower
2000	Pattern colour table
3800	Sprite pattern table
3C00	ASCII table upper
3F80	Sprite attribute table

2.3. Multicolour (Mode 2)

In this mode the screen is split into 64 blocks by 24 blocks, providing an unrestricted colour display with sprites. The memory map is the same as mode 0 and again no software is supplied for operation in this mode as it assumed mode 1 will be used instead.

2.4. Text (Mode 3)

In text mode the screen is divided into 40 characters by 24 lines, two colours only are displayed at any one time. One for the text colour the second for the background. No sprites are available in this mode, and the memory map is the same as that for mode 0. It is possible to have user-defined characters when in this mode.

3. Graphics 2 Hi-res,

As has been mentioned prior to this mode 1 (Graphics 2) has the capacity to be a 256*192 bit-mapped graphic mode.

During the following there will be references to 'the current plot position' this is simply the last point plotted. Note this will be 0 0 to start with.

3.1. Plotting modes

In graphics 2 mode four plotting modes are available:

Pmode	Effect
OVERPLOT	In this plotting mode any point/line plotted will always plot a point/line in the foreground colour.
TOGGLEPLOT	When a line is plotted in this mode any point will toggle the condition of the point currently in that position.
DESTROY	A line plotted in this mode will plot in the background colour.
NULLPLOT	This is the no action plotting mode i.e. any point will not change when it is plotted.

A plotting mode is entered by the FORTH words OVERPLOT, TOGGLEPLOT, DESTROY and NULLPLOT.

3.2. Points

Single points are plotted on the screen by using the word POINT, the stack requirements for POINT are (x y ---)

e.g. 100 40 POINT
will plot a point at 100 across 40 down.
N.B. remember what plotting mode you are in.

3.3. linear lines

The are a number of ways to draw a line, as follows:-

FORTH word	Stack comment & Effect
PLOT-ABS	(x1 y1 x2 y2 ---) Draw a line from x1 y1 to x2 y2
PLOT-REL	(dx dy ---) Draw a line from the last point plotted cx cy to cx+dx cy+dy
PLOT-TO	(x1 y1 ---) Draw a line from the last point plotted to x1 y1

Each word produces a line that is as linear as possible, note that if you require to destroy a line it is often better to plot in the same direction as you originally plotted it.

3.4. Arcs and circles

It is possible within the graphics vocabulary to plot circles and arcs as follows:-

FORTH word	Stack comment & Effect
ARCX	(start-x start-y centre-x centre-y endpoint-x cw/ccw ---) Draw an arc from start-x start-y, centred at centre-x centre-y, in a direction determined by cw/ccw, cw/ccw=0 produces a clockwise arc else an anticlockwise arc. The position at which the arc will stop is when the x value of the arc crosses the endpoint-x line.
ARCY	(start-x start-y centre-x centre-y endpoint-y cw/ccw ---) This is the same as ARCX except that the endpoint is determined by the arc crossing endpoint-y line.
CIRCLE	(x y r ---) Draw a complete circle centre x y radius r.

As with linear lines it is advised to unplot arcs in the same direction as you plotted them.

3.5. Point movement

There exist two words which move the plot position without plotting a line these are:

MOVE-REL	(dx dy ---) Move the plot position by dx dy
MOVE-TO	(x y ---) Move the plot position to x y

3.6. Colour changes

There are a possible 16 colours displayed at anyone time on the graphics two screen, with the maximum colour definition of two colours per eight horizontal pixels.

Colours are changed by the following methods:-

n1 n2 COLOUR	Changes the foreground colour to n1 and the background colour to n2
n1 BACKGROUND	Changes the background colour to n1
n1 FOREGROUND	Changes the foreground colour to n1

Constants are provided for each colour code 0..15.

Code	Colour constant
0	TRANSPARENT
1	BLACK
2	MED-GREEN
3	LIGHT-GREEN
4	DARK-BLUE
5	LIGHT-BLUE
6	DARK-RED
7	CYAN
8	MED-RED
9	LIGHT-RED
10	DARK-YELLOW
11	LIGHT-YELLOW
12	DARK-GREEN
13	MAGENTA
14	GRAY
15	WHITE

3.7. Plotting examples

Below is an example of how the above words can be used to produce a coloured display.

```
GRAPHICS DEFINITIONS
WHITE BLACK COLOUR          Set background colour to black,
                            foreground to white
GRAPH2                      Place system in mode 1 and
                            clearscreen
0 0 255 0 PLOT-ABS          Draw a line across the top of the
                            screen
-128 191  PLOT-REL          Draw a line to the bottom centre of
                            the screen
0 0       PLOT-TO           Draw a line to the top left of the
                            screen
128 95 95 CIRCLE            Draw a circle in the centre of the
                            screen
DARK-RED FOREGROUND         Change foreground colour to dark-red
100 120 128 96 156 1 ARCX   Draw an arc
100 120 128 86 156 1 ARCX   Draw a second arc
5 BACKGROUND                Change background colour to colour
                            5, (light-blue.)
148 40 15 CIRCLE            Draw a circle
108 40 15 CIRCLE            Draw a circle
128 96 15 CIRCLE            Draw a circle
```

From the last three commands it becomes very noticeable about the lack of colour definition.

4. Sprites

Sprites are graphics that can occupy space on the screen independently and in addition to the characters which normally make up the screen. Thirty two sprites are available on the highest priority video planes. Sprites are only visible in graphic 1, graphic 2 and multicolour modes.

4.1. Sprite animation

The following words allow animation of sprites in the above modes:-

IS-SPRITE (colour sprite-shape£ x-pos y-pos plane£ ---)
Places a new sprite on the screen at position x-pos y-pos in colour 'colour', the plane£ is reference to the sprite priority, 0 being the highest 31 being the lowest.

MAG (n1 ---) n1 = 0..3 n1 defines the size of all the sprites on the screen as follows:

n1	effect
0	8*8 sprite
1	8*8 sprite mapped onto a 16*16 set of pixels
2	16*16 sprite
3	16*16 sprite mapped onto a 32*32 set of pixels

RSPRITE (---) Resets all sprites from the screen.

SHAPE (n0..n7 sprite£ ---) Defines a new sprite pattern.

SPRITE-POS (x y plane£ ---) Moves the sprite on plane£ to position x y

4.2. Sprite examples

Below follows an example of the use of sprites, it can also be found on screen 18 of GDEMO.SCR

```
GRAPHICS DEFINITIONS BINARY
  : SPRITE-S1       00011011
                    00010001
                    00010001
                    00010001
                    00010001
                    00010001
                    00010001
                    00010001  00 SHAPE  ;

  : SPRITE-S2       00001001
                    01101001
                    10010110
                    00010000
                    00010000
                    10010000
                    01100000
                    00000000  10 SHAPE  ;

  : SPRITE-S3       00000000
                    00000000
                    00000000
                    01000001
                    10100001
                    10010001
                    00010001
                    00010101 01 SHAPE   ;

  : SPRITE-S4       00000000
                    00000000
                    00000000
                    00000110
                    00001000
                    00001000
                    00001110
                    00001001 11 SHAPE  ;
```

```
DECIMAL

: ON-SPRITES      SPRITE-S1 SPRITE-S2
                  SPRITE-S3 SPRITE-S4
                  1  0 00 00 0 IS-SPRITE
                  3  0 00 00 1 IS-SPRITE
                  9  0 00 00 2 IS-SPRITE
                  13 0 00 00 3 IS-SPRITE  ;

: SDEMO           GRAPH2 RSPRITE 3 MAG ON-SPRITES
                  5000 0 DO
                         I  I  0 SPRITE-POS
                         0  I  1 SPRITE-POS
                         60 I + I  2 SPRITE-POS
                         200  I 50 / 10 + 3 SPRITE-POS
                  LOOP  ;

SDEMO
```

Note. No more that four sprites can appear in a row on the screen at any one time, if this happens then the lower priority sprite(s) will become invisible and the Fifth sprite flag in the status register will be set to '1', this is cleared ONLY when the status register is read or the system reset. The number of the fifth sprite is placed into the lower five bits of the status register. The status register can be read by VDP-STAT. Coincidence of sprites can be detected by reading the Coincidence flag in the VDP-STAT register, the coincidence flag is again only cleared by reading the VDP-STAT register or by a system reset.

5. User Defined Characters

It is possible in TEXT mode (mode 3) to have up to 256 user-defined characters, each character being a 6*8 block of pixels. Thus giving an effective possible pixel definition of 240*192.

To define a character the word CHAR is used. CHAR is very similar to SHAPE, except that the shapes defined are the normal ASCII character set. For example

```
BINARY
  00000000
  11111000
  01001000
  01111000
  01000000
  01000000
  01000000
  00000000
DECIMAL
98 CHAR
```

Will define the 'big' lower case 'b' into a rather poor proper lower case b. Note that the last two bits of each number is a zero, as a switch from text to graph2 mode will display any character as an 8*8 character block. CHAR should NOT be used in mode 1, as the character set is split to allow for the graphics pattern and colour screen.

6. Graphics glossary

6.1. Definitions

ARCX (start-x start-y centre-x centre-y endpoint-x cw/ccw ---) Draw an arc from start-x start-y, centre centre-x centre-y, in a direction determined by cw/ccw zero equals clockwise none zero equals counter clockwise, until the arc crosses a line determined by endpoint-x.

ARCY (start-x start-y centre-x centre-y endpoint-y cw/ccw ---) This is the same as ARCX except that the endpoint is determined in the Y plane.

BACKGROUND (back-ground-colour ---) Sets a new back-ground colour (see colour)

CHAR (n0..n7 char£ ---) Redefines the character shape when in TEXT mode (see shape)

CIRCLE (centre-x centre-y radius ---) Draws a circle in the current colour.

CLS	(---) Clearscreen for any mode.
COL	(x y --- colour) Returns colour of pixel at the point x y when in graph2 mode.
COLOUR	(text-colour back-ground-colour ---) In text mode sets foreground and background colours. Text and background colours must be in range 0..15. In graph2 mode text-colour represents the colour that a new pixel will be drawn in.
COS	(n1 --- (cos-n1)*1000) Returns the cosine of n1*1000
DESTROY	(---) Sets current plotting mode to destroy i.e. any new pixel will be plotted as the background colour.
FOREGROUND	(text-colour ---) Sets new foreground colour (see colour)
GEMIT	(n1 ---) Graphic2 version of emit, transmit ASCII character to graphic screen.
GRAPH1	(---) Changes mode to graphic1 and clears screen. In graphic1 mode screen is 32 characters by 24 lines.
GRAPH2	(---) Changes mode to graphic2 and clears screen. In graphic2 mode screen is 256 pixels by 192 pixels with possible 32*24 characters.
GRAPHICS-TEXT	(---) Sets output device as graphic2 video, and CORTEX/PP95 keyboard. (See NORMAL-TEXT)
GTYPE	(address count ---) Graphic2 version of TYPE (see GEMIT)

IS-SPRITE (colour sprite-shape£ x-pos y-pos plane£ ---) Places new sprite on screen at position x-pos y-pos. Sprites are only visible in GRAPH1 GRAPH2 and MULTI modes.

MAG (n1 ---)
n1 = 0..3 n1 defines the size of the sprites displayed on the screen as follows :-

n1	effect
0	8*8 sprite
1	8*8 sprite mapped onto a 16*16 set of pixels
2	16*16 sprite
3	16*16 sprite mapped onto a 32*32 set of pixels

MOVE-REL (dx dy ---) Moves current plot position by dx dy without affecting the graphics screen.

MOVE-TO (x y ---) Moves plot position to x y without affecting the current graphics screen (See MOVE-REL)

MULTI (---) Sets graphics mode to MULTIcolour and clears the screen, in Multicolour mode the possible resolution is 64*48 with no colour restrictions.

NORMAL-TEXT Set the i/o device to a terminal (see GRAPHICS-TEXT)

NULLPLOT (---) Changes the plotting mode to no plot, in this mode any line drawn will not change the graphics screen in any way.

OVERPLOT (---) Changes the plotting mode to over plot, i.e. a line will always be drawn no matter what was on the graphics screen prior to the line being placed.

PLOT-ABS (x1 y1 x2 y2 ---) Draw a line from x1 y1 to x2 y2 in the current foreground colour.

PLOT-REL (dx dy ---) Draw a line from the current plot position cx cy to cx+dx cy+dy

PLOT-TO (x1 y1 ---) Draw a line from the current plot position to x1 y1

POINT (x y ---) Plots a single pixel at the point x y in the current foreground colour.

REG! (n1 reg£ ---) Stores n1 in VDP register reg£

RESEED (n1 ---) Resets the 'random' number seed to n1

RND (n1 --- value) Produces a 'random' number between 0 and n1-1

RSPRITE (---) Resets all sprites. i.e. all sprites will disappear from the screen.

SGET (cell --- data ; OR source-cell dest-cell ---)
Text mode :-
cell = 0..959 Data returned will be the ascii value of the shape currently at that position on the screen.

Graphics2 mode :-
source-cell = 0..767
dest-cell = 0..767 Transfers the contents of the source-cell to that of the dest-cell.

Graphics1 and Multicolour mode :- In these modes the message "Command Illegal in current mode" will be displayed N.B. In multicolour mode unless you are using a terminal you will not be able to read this message and it will appear as a string of

random colours.

SHAPE (n0..n7 sprite£ ---) Defines a sprite pattern in the shape of n0..n7. Sprite£ is in the range 0..255 e.g. the following will define sprite shape number 0 as a space invader.

```
BINARY
10000001
01000010
00111100
01100110
10111101
01111110
00111100
00011000   0 SHAPE
```

SIN (n1 --- (sin-n1)*1000) Returns the sine of n1*1000

SPRITE-POS (x y plane£ ---) Moves the sprite on plane£ to position x y (See IS-SPRITE)

SPUT (data cell£ ---) Text mode :-
cell = 0..959 Places ascii character data on the screen at position cell£

Graphics2 mode :-
cell = 0..767 Places ascii character data on the screen at position cell£ N.B. in this mode eight bytes go to make up a single character on the screen.

Graphics1 and Multicolour modes :- In these modes the message "Command Illegal in current mode" (See SGET)

TEXT (---) Sets graphic mode to text giving a resolution of 40*24 characters with user redefinable characters, but only one text and one

background colour.

TOGGLEPLOT (---) Changes plotting mode to toggle plot, that is if a point is plotted over another that point will be turned off, and vice a versa.

V-FILL (v-address count data ---) Video RAM version of FILL i.e. fill the v-ram from v-address to v-address+count with data.

V-MOVE (v-source v-dest count ---) Video RAM version of MOVE i.e. move contents of v-ram from v-source to v-source+count to v-dest. Note that this operation uses memory starting at HERE as a work space, and does NOT check for avaiable space.

VDP-STAT (--- value) Returns the status of the VDP. This contains the interrupt pending flag, the sprite coincidence flag, the fifth sprite flag. The format is as below.

BIT	MSB							LSB
MEANING	F	5S	C	Fifth Sprite Number				

VRAM! (data v-address ---) Video version of C!

VRAM@ (v-address --- data) Video version of C@

6.2. Constants

The below are used in conjunction with COLOUR and IS-SPRITE for setting new colours i.e DARK-RED CYAN COLOUR sets the foreground colour to dark-red and the background colour to cyan.

TRANSPARENT (--- 0)
BLACK (--- 1)
MED-GREEN (--- 2)
LIGHT-GREEN (--- 3)
DARK-BLUE (--- 4)
LIGHT-BLUE (--- 5)

```
DARK-RED          ( --- 6 )
CYAN              ( --- 7 )
MED-RED           ( --- 8 )
LIGHT-RED         ( --- 9 )
DARK-YELLOW       ( --- A )
LIGHT-YELLOW      ( --- B )
DARK-GREEN        ( --- C )
MAGENTA           ( --- D )
GRAY              ( --- E )
WHITE             ( --- F )
```

6.3. Variables

CURCOLOUR The value held in the upper nibble of this variable contains the current text colour, and in the lower nibble is the background colour.

EDOM Contains the current value of the current mode as follows:

0 Graphics1 mode
1 Graphics2 mode
2 Multicolour mode
3 Text mode