P2000 FORTH

Installation Guide

P2000 FORTH INSTALLATION GUIDE: TABLE OF CONTENTS

WARRANTY NOTICE

Humans cannot guarantee their software to be bug-free. Nor can we. So we must depend on users to tell us what went wrong when using this product.
So we run an Update Service that will provide improved code on the Initializer Tape you may send us whenever we will request you to do so.
If you have found a bug, please do notify us, giving a detailed account of the circumstances under which the anomaly occurred, or that have led to the anomaly.

We also guarantee that the Source Code of this implementation will be made available to P2000 and/or Forth User Groups if the support of this product should be discontinued.

(C) 1983 - F.v.d.Markt & J.Vermeulen

For P2000 FORTH Release 1.2, the following changes are valid:

* The Initialization procedure has been changed for better security. Please replace page 2 of your Installation Guide by the new page accompanying this Update if you have Release 1.2.

* In some circumstances, a Forth Source Tape may have been made unaccessible after a MONitor I(inspect operation under Release 1.0 or 1.1. This bug has been eliminated in Release 1.2.

* A number of bugs in the Utilities (Runfile procedures, Editor, Assembler) have been eliminated. Recompile your Runfiles from the new Utility Track.

* The Boot program for Disk based systems has been modified in such a way that access to Drive 1 should be more comfortable and the motor is not turned off any more at Boot Time.

IMPORTANT NOTICE:

As it would appear that most intolerable bugs have been eliminated with the issue of Release 1.2, it is our intention NOT TO SUPPORT previous releases any more.
This implies that ANY following Releases will ONLY ACCEPT System tapes from RELEASE 1.2 for updating.

SOFTWARE ISSUES:

We are considering bringing out a Cassette with additional software in Source form to run under P2000 Forth.
Under preparation are:

* a Screen oriented editor

* a UFOCODE (Universal FOrth CODE) utility for transportation of Forth Source screens between systems by means of audio tapes.

* String Handling and Line Printer commands.

We will welcome software by other authors for inclusion in this tape, if found suitable.

You may send your tape or disk to:

Jan Vermeulen
Riouwstraat 55
1521 SC Wormerveer/NL

December 1983

The following changes and additions to Rev. 0.0 of the Installation Guide should be noted.

Page 2: Initializing your system

* There is no need for the Initializer Tape to be Write Enabled after it has been used for the first time.
* Do not open the Tape Drive unless requested.
* Do not interrupt the Initialization procedure: restart initialization after its completion if you have made an error.

Page 4: Memory Map changes for Release 1.1:

	16K Tape	32K Tape	48K Tape	48K Disk
P2000 Forth Nucleus	6800-7F8D	6800-7F8D	6800-7F8D	6800-80A2
User's Work Space	7F8D-9717	7F8D-C707	7F8D-DEFF	80A3-DBAD
Free for Compilation	5 899	18 271	24 307	23 180

Page 5: Subsection 05.01 - KEYBOARD:

The Local Keys are also available when in MON.

Page 6: TAPE: NXTBLK (see page 10) is updated.

Page 6: Subsection 05.03 - SCREEN:

For details on conversion, see Appendix 1 under your National Version.

Page 10:

On a disk based system, disk capacity tables are located in a 2 bytes wide list just below ORIGIN.
-16 +ORIGIN @ returns the number of blocks on drive # 0.
-14 +ORIGIN @ returns the number of blocks on drives 0 thru 1.
-12 +ORIGIN @ returns the number of blocks on drives 0 thru 2.
-10 +ORIGIN @ returns the number of blocks on drives 0 thru 3.
If less than four drives are available, the remaining entries contain the maximum number of blocks available.

-8 +ORIGIN @ returns the number of tracks on drive 0.
-6 +ORIGIN @ returns the number of tracks on drive 1.
-4 +ORIGIN @ returns the number of tracks on drive 2.
-2 +ORIGIN @ returns the number of tracks on drive 3.

If the value returned is over 256, the drive has two heads, and substracting 256 from the value will return the number of cylinders available on that drive.
In dual sided drives, both sides will be Formatted in the same operation.

Page 11: S(ave: Another example is found on page 27.
Use a formatted Tape.

Page 12: the 3rd line from the bottom should read:
0 BLOCK UPDATE F3 PREV @ 2+ C! FLUSH

Runfiles:

Once a Runfile has been L(oad-ed you only can discard it by re-setting the System and booting Forth again.
There is, however, another solution which is making a Runfile immediately after Forth has booted. L(oad-ing this file will restore the Nucleus (a time saver for disk based systems).

Assembler in a Runfile:

The Patch for ;CODE must be repeated as this will not have been S(ave-d to Tape or Disk.
It is best to put this and other patches in a word that is executed once after a Runfile has been L(oad-ed.

Page 29:
U* operates correctly on unsigned positive numbers only.

Page 33:
R/W for Disk returns Message #6 ('Disk Range?') if the Block is not available (Release 1.1).

Page 47: on the bottom line, read 'right' instead of 'left'.

Page 48: LABEL
This word does not quite work as advertised: a defined label must be SMUDGEd (or: delete SMUDGE in the definition) and you can only use LABEL at the start of an assembler routine, not in the middle of it.

Lost Tape Blocks:

Tape Blocks may get lost and become inaccessible because of a power failure, turning off the P2000 with a tape in the Drive, or opening the Drive during a Write operation.
You may cure this by rewinding the Tape until in front of the lost block with a non updated block in one of the buffers, then COPY it to the 'lost' block, and by FLUSH force it back on the tape. The block's contents then can be restored from a backup.

A new Utility on track 8 of Release 1.1: .BUFS

You will want to have this word resident all the time: if BLOCK executes with no block # on the stack, you may be left with an unFLUSHable screen. .BUFS will show the addresses and contents of all your buffers, and you may kill the offender with:

```
HEX 7FFF adr !
```

ADDITIONAL INFORMATION ON C o p y r i g h t s

The P2000 Forth software package is subject to our Copyright as claimed in the Installation Guide, on the Initializer Tape and on the System Tape.

From a Copyright point of view however, three groups of software must be distinguished:

(1) Software that is in the Public Domain:
- fig-FORTH for the 8080, Version 1.1
- the fig-Editor
- a number of Utilities published in 'Forth Dimensions', the journal of the U.S. Forth Interest Group

This software is included free of charge.

(2) Software published for personal use and on which copyright for commercial distribution has been reserved:

- "Pacman for the ACE", published in 'Your Computer', Volume III, #2, March 1983.
- a Forth Assembler for the 8080 processor by John Cassady (Forth Dimensions, Vol. III, # 5).
- a number of Utilities published in 'Forth Dimensions' and 'Het Vijgeblad', the journal of the Dutch HCC-Forth Interesse Groep.

This software is included free of charge, but it is our responsibility to extend the authors' claims for copyright.

(3) Software that has been created by the authors of P2000 Forth (including improvements on, and extensions to, software that belongs to one of the previous groups):

- Adapting 8080 fig-Forth to the Z80 and eliminating a number of bugs in the fig-FORTH Model.
- Improved VLIST, based on the original idea by A.C.S. van Roosmalen.
- Adapting and extending the 8080 Forth-Assembler to the Z80 and introducing Relative Labels.
- Improving the fig-Editor.
- Utilities signed by one of the Authors.
- The implementation on the P2000 as described in Sections 2 and 5 thru 10 of the Installation Guide.

If you're still 'stand-alone': join a Forth User Group.
Addresses on page 1 of the Installation Guide.

May the FORTH be with you!

October 1983

01 - INTRODUCTION TO P2000 FORTH

This Installation Guide is not intended as a Forth tutorial.

Among the good tutorials that are available we recommend the book 'Starting Forth' by Leo Brodie (Prentice-Hall, Englewood Cliffs, NJ, 1981; ISBN 0-13-842922-7). Besides being highly readable and entertaining, this book can be obtained from almost any computer book store.
Although the book teaches poly-FORTH, all differences with the 1979 Standard and fig-Forth are well explained.

Another good tutorial is 'The Forth Encyclopedia' by Mitch Derick & Linda Baker (2nd Edition, Mountain View Press, Mountain View, CA, 1982), which explains most fig-FORTH words in detail.

The present implementation is in fig-Forth; although the Forth Interest Group in the USA has adopted the 1979 Standard (only as late as early 1982!), European Forth users appear to be far more reluctant to do so, and (we think) for good reasons.

Those wishing to convert the system to 79-Standard Forth should obtain both the following publications:

a- 'Forth-79', a publication of the Forth Standards Team
(Forth Interest Group, San Carlos CA, 1980)

b- 'Forth-79 Standard Conversion' by Robert L. Smith
(Mountain View Press, Mountain View, CA, 1981).

We may provide a package for conversion to 79-Standard Forth at a later date.

Some addresses of Forth User Organizations are:

BRD:
Fig-Deutschland
Postfach 20 22 64
2000 Hamburg 20

Nederland:
HCC Forth Interesse Groep
A. v.d. Meerweg 31
1191 ED Ouderkerk a/d Amstel

UK:
Forth Interest Group
38 Worsley Road
Frimley, Camberley
Surrey GU16 5AU

USA:
Forth Interest Group
P.O. Box 1105
San Carlos, CA 94070

In addition to many Forth Chapters and machine oriented user groups in the USA, there are also Forth User Groups in Australia, Canada and Japan.

02 - INITIALIZING YOUR SYSTEM

The P2000 Forth Cassette supplied enables you to configure your system as either a tape based system or as a disk based system that is booted from the System Tape you are going to make.

You always can reconfigure or make a new System Tape from the Initalizer Tape, so there is no need for providing a back up facility.
Whenever updates are released, you can send in your Initializer Tape while continuing to work with the System Tape.

For initializing your system, proceed as follows:

1. Make sure you have an empty cassette at hand.
2. For a Disk based system, you will also need an empty disk for assessing the capacity of each disk drive.
3. Remove any Keys from slots 1 and 2 of the P2000.
4. Power up or Reset your P2000
5. Reverse the Write-Protect Plug for Track A of the Initializer Tape (it arrives both sides Write-protected).
6. Insert the Tape, Side A facing upward, into the Drive.
7. Follow the instructions displayed on the screen, alternately placing the Initializer Tape and the System Tape in the Tape Drive as prompted.
   You will be asked if the System Tape is to be used on the system it is being initialized on. If that is not the case, you must enter a few data from the keyboard.
8. When the message 'That's all' appears on the screen, extract the Initializer Tape from the Drive and write-protect it.
9. Reboot your system with the System Copy.

Whenever the Initializer Tape is used again, it will request the System Tape it has previously initialized. You then will have the option either to have it erased and to have a fresh tape initialized, or to reconfigure the System Tape.
This normally will only be necessary if the P2000 has been upgraded from Tape to Disk, or for installing an updated version of P2000 Forth.

02 - INITIALIZING YOUR SYSTEM (Rel. 1.2)

The tape supplied has the System Initializer on side A and 42 screens of Utilities on side B. Once you have a System Tape you can view the contents of side B with 0 LIST and 1 LIST.

The Initializer allows you to make two System Tapes. If you have a System Tape from Rel. 1.0 or 1.1, you should first make a second System Tape prior to overwriting your old System Tape.

Whenever updates are released, you can send in your Initialize Tape while continuing to work with your System Tapes.

For initializing your System, proceed as follows:

1. Have at least one empty cassette at hand.
2. Slot 1 must be empty.
3. Power up or Reset your P2000.
4. If you do not already have two Rel. 1.2 System Tapes, the Initializer Tape must be Write Enabled.
5. Insert the Initializer Tape, the label with 'P2000 FORTH' in green characters facing upwards.
6. Follow the instructions that will be displayed on the screen. These will vary according to the status of your System Tapes:
   a- if you are updating from a previous release, you will be requested to use a fresh tape or the oldest of your System Tapes (the release no. will be displayed and checked).
   b- if you have a new Release 1.2 Initializer, you will just be requested to insert 'the System Tape' which is a fresh tape.
   c- if you have 2 Rel. 1.2 System Tapes, the Initializer will request either tape for reconfiguration. This enables you also to replace a System Tape that is beginning to wear.
7. You must insert the System Tape and the Initializer Tape as requested. Inserting the wrong tape is nearly always fatal !! For that reason, you have one opportunity to check at each tape change before the program proceeds.
8. If you have made an error in giving the configuration of the machine your System Tape must run on, do not abort the procedure, but restart it after it has been completed.
9. The procedure is completed when the message 'That is all' appears on your screen. Now first boot your system with the first System Tape you have made, prior to making your second system tape.

If anything has gone wrong during the above procedure and if you do not feel safe about restarting the procedure on your own, you must contact us for help at phone # (Netherlands) 075-283943.

03 - FOR OLD FORTH HANDS

We have been faithful to the fig-Model for the 8080, Ver. 1.1, as implemented by John Cassady and Kim Harris.
The low level code has been adapted to the Z80's more powerful instruction set. thus increasing the speed of a number of words like FILL and CMOVE.

However. we have felt free to improve on the model by mending a number of bugs that must have been sources of irritation for at least a few of you:

FLUSH now flushes all buffers and sets their numbers to 7FFFH

EMPTY-BUFFERS now uses FLUSH. so it is now possible to List Scr# 0 after COLD or WARM.

CMOVE has been made smart. and will move either way without deleting data.

EXPECT has been rewritten in machine code and now filters out all control codes except RUBOUT and CR.

ENCLOSE will handle strings of over 256 spaces.

FORGET is smart and adjusts vocabulary pointers

VLIST gives formatted output and also lists the CFA's, thus dispensing with the need for a Decompiler unless you really must have one.

We include the fig-Editor in source form with MATCH and -TEXT written in machine language. Some useful extensions are also provided (see documentation in the source screens).
D. DELETE and COPY have been altered (see section 13).

A prototype of a Z80 Forth Assembler using Relative Jumps to declared Labels is also provided. We would like to have your comments on its quirks.

Before you start to try out the system, we recommend that you read at least the following sections:

- for a tape based system: sections 06 and 09
- for a disk based system: sections 07 and 10
- for any system: sections 04, 05, 08 and 11

Should you come across a bug. do not hesitate to notify us: you may be the only one to have noticed and we are grateful for any help in improving this product.

04 - MEMORY MAP; DON'TS

The following Memory Maps are valid for Release 1.0:

Contents of Area	16K Tape	32K Tape	48K Tape	48K Disk
P2000 System ROM	0000-0FFF	0000-0FFF	0000-0FFF	0000-0FFF
ROM Key (not used)	1000-4FFF	1000-4FFF	1000-4FFF	1000-4FFF
Video RAM (T&M)	5000-57EF	5000-57EF	5000-57EF	5000-57EF
System Use	57D0-57FF	57D0-57FF	57D0-57FF	57D0-57FF
Video Attributes (M)	5800-5FFF	5800-5FFF	5800-5FFF	5800-5FFF
System Use	6000-606E	6000-606E	6000-606E	6000-606E
Terminal & Monitor	6070-67FF	6070-67FF	6070-67FF	6070-67FF
P2000 Forth Nucleus	6800-7FAA	6800-7FAA	6800-7FAA	6800-8095
User's Work Space (includes Stack)	7FAB-9717	7FAB-C707	7FAB-DEFF	8096-DBAD
User Area and TIB	9718-97F7	C708-C7E7	DF00-DFDF	DBAE-DC8D
DOS Routines				DC8E-DFDF
Tape/Disk Buffers	97F8-9FFF	C7E8-DFFF	DFE0-FFFF	DFE0-FFFF
Number of Buffers	2	6	8	8
	16K Tape	32K Tape	48K Tape	48K Disk
Free for Compilation	5 869	18 241	24 277	23 193

The exact amount of available free memory always can be found by using the sequence: SP@ HERE - 128 (decimal) - .

The second memory bank from E000 to FFFF on 48K machines is not used and can be accessed from within an application program by using the Forth sequences:

```
1  94 ( hex)  P!   to access the bank, and
0  94 ( hex)  P!   to switch back to the other bank.
```

The Screen Area that can safely be accessed by applications is limited to hexadecimal addresses 5000 thru 57CF and 5800 thru 5FCF, respectively. Clearing the screen can be safely done by 12 EMIT (decimal). With the cursor off, this will not interfere with memory mapped display routines.

CAUTION: Section 08 lists the system variables that can be accessed and maybe modified by an application.
Do not modify any other memory location within that area, as a system crash might result.

NEVER tell US that you have NOT been warned !!!

05 - THE P2000 AS A TERMINAL

In order to provide a comfortable work space even on the smallest versions of the P2000, we have implemented the machine as a terminal comprising a Screen, a Keyboard and a Line Printer interface, as it is expected by most Forth implementations.

05.01 - KEYBOARD

On the Keyboard, we have provided a number of Local Keys that will normally not be echoed to the Forth machine.
These keys can not only be used when in command mode, but they also will function when Forth accesses the Terminal thru KEY, EXPECT, EMIT and ?TERMINAL.
For details on how to turn them off, see section 08.

Using either the SHIFT or LOCK Keys will slow down output to the screen by approximately 43 milliseconds per character.
This speed applies with a value of 32 decimal in the system variable CDELAY. By modifying the contents of CDELAY one may increase or decrease the output speed. Each step changes this speed by 1338 microseconds. A value of 1 gives the fastest output, a value of 0 results in the slowest output.

SHIFT+TAB will toggle the keyboard between two modes:
(1) the Typewriter Mode in which you must use the Shift or Lock Keys for typing Upper Case, and
(2) the Program Mode in which it is not possible to type in Lower Case characters.

SHIFT+5 on the numeric pad will stop output to the Terminal during EMIT and ?TERMINAL (the Terminal goes busy).

START cancels the Terminal's Busy Condition and output, if any, is resumed.

STOP is returned as 3 (^C) by KEY and as a true flag by ?TERMINAL. It is ignored during EMIT.

The following LOCAL KEYS will function while in Command Mode and when the Terminal has gone Busy:

← (T Model only) provides a horizontal wraparound scroll of the screen to the left.
→ (T Model) does the same to the right.

FLASH (SHIFT+2 on the numeric pad) will toggle the cursor from on to off and back. When the cursor is off, the horizontal scroll is disabled and the left side of the screen is shown

TAPE (SHIFT+7 on the pad) will rewind a cassette if a tape is present in the drive. There is no error message.

DISK (SHIFT+8 on the pad) will toggle the disk drive motor from on to off and back.

LPT (SHIFT+00 on the pad) will toggle the Line Printer Interface between on and off.
Switching the printer off-line will also disable the interface until it is toggled to 'on' again with the printer set to 'on-line'.

05.02 - KEY CODES

Appendix 1 lists codes returned for any national version for which P2000 Forth so far has been implemented. (Keys marked 00 are not echoed).
Both '0' keys return 30H, the '00' key returns B0H.
Bit 7 of all codes is reset by the headerless primitive (KEY). Applications that wish to know the particular key that has been pressed must make this reset inoperative by patching (KEY) as follows: 0 ' KEY 1+ @ 3 + !
Calling COLD will restore the original code.

05.03 - SCREEN

For portability of source text, P2000 Forth uses ASCII codes for internal storage. Characters that are sent to the screen are translated according to each version's character set.

The list of characters to be converted is located at address CRTCOD, which is found by:

```
506E   @   0B ( hex)   -   CONSTANT   CRTCOD
```

The first byte of the list gives the list's length.

The list of characters into which the above characters are converted (in reverse order!!) is located at address CRTAB (there is no length byte here):

```
CRTCOD   5   +   CONSTANT   CRTAB
```

05.04 - CONTROL CODES

The Terminal accepts the following control codes, some of which have the same function as the Local Keys discussed earlier:

01H,05H as ← key	10H as LPT-key
02H,06H as → key	11H as FLASH-key
04H as DISK key	14H as TAPE key
07H BELL: sound beeper	15H Cursor and Hor. Scroll on
08H Back Spade/Rubout	16H Cursor and Hor. Scroll off
0CH Clear Screen/Form Feed	1FH as SHIFT+TAB
0DH Enter (CRLF)	

05.05 - LINE PRINTERS

In order to accomodate printers that lack a back space feature, the Terminal's bottom line is used as a buffer, contents of which are passed to the printer only when a Carriage Return or a Form Feed is executed.

Character conversion tables are located at addresses LPTAB and LPTAB1, which can be found by:

```
    605E @ DUP   CONSTANT   LPTAB  0D  +  CONSTANT LPTAB1
```

According to the Line Printer strap on the P2000 bottom board, two conversion tables for either Matrix or Daisy Wheel printers are loaded at LPTAB and LPTAB1, respectively:

- for ASCII Matrix Printers either table consists of a length byte followed by a 2 bytes wide conversion list.
- for Daisy Wheel Printers the first byte is a zero followed by
  - a length byte and (if the byte is non zero) a three byte wide conversion list for double printing characters.
  - a length byte and a list for single printing characters.

The table loaded at LPTAB converts to US-ASCII Matrix Printers or to the BIL Print Wheel.

The table loaded at LPTAB1 converts to the P2123 Matrix Printer or to the ESA Print Wheel.
If either of the latter is connected, the user should adapt the System Variable PRTABL with: LPTAB1 60JE !

The only exception to this is, that a French P2123 connected to a Swins P2000M requires the table loaded at LPTAB.

Transparent output to the printer should go through the P2000 Forth word LPT which uses no conversion tables.

06 TAPE OPERATING SYSTEM (T.O.S.)

Most Digital Mini Cassettes will hold two tracks of 42 1K Forth screens each. The screens are numbered consecutively from 0 to 41 by the Formatting Routine provided in the Monitor.

The TOS always will calculate tape block access from the last block accessed, and only check the current block number after the read/write operation has been performed, UNLESS the Forth word RESET has been executed prior to the access.

RESET is automatically executed whenever a Tape Error occurs.

CAUTION: have RESET execute after any tape change in order not to overwrite data in a wrong block.

Error codes returned by the TOS are:

'A' - Drive not ready	'I' - Rewind time out
'B' - Beginning of tape	'J' - Tape torn
'C' - Checksum Error	'K' - Invalid function/Block no.
'D' - Mark checksum error	'L' - End of tape (reading)
'E' - End of tape (writing)	'M' - No mark found
'F' - Tape full	'N' - No record found
'G' - Tape write protected	'W' - Not a Forth Source Tape

Normally, the TOS will take the proper action if any of errors B, E, M or N are encountered and not issue an error message. Errors E or L may occur when writing to or reading from a short tape.

07 - DISK OPERATING SYSTEM (D.O.S.)

Storage is in 4 1K Forth Screens to a disk track. There is no interleaving as there is with CP/M and P2000 DOSes.

The available functions are: Read a Block, Write a Block and Reset Current Drive, all three accessed through Forth with:
1 R/W 0 R/W RESET respectively.

RESET is used internally by R/W in order to reset a drive after a Disk I/O Error has occurred.

I/O-error codes returned by the DOS (or the Monitor) are:

'0' - System Disk (e.g. a 24K Basic System Disk)	
'1' - Drive not ready	'3' - Drive not available
'2' - Disk write protected	'4' - Block not available

08 - SYSTEM VARIABLES

The System Flag Byte is located at address 6013H.

The following flags may be read but should not be changed by an application program:

Bit #:	Designates:	set:	not set:
0	Model	M	T
1	Forth version	Tape	Disk
2	Lower case	Enabled	Disabled
3	Cursor/Horizontal Scroll	OFF	ON
5	Line Printer Interface	ON	OFF
6	Disk Motor	ON	OFF

To influence the status reflected by bits 2, 3, 5 or 6, the application should EMIT the associated Control Codes listed in Section 05.04.

The following two flags may be directly modified by the Application. in order to adapt the Terminal to the requirements of the program:

Bit #:	Designates:	set:	not set:
4	Terminal Busy	Disabled	Enabled
7	Local Keys	Disabled	Enabled

Of the other System Variables, only two may be modified by the User or the Application:

CDELAY at hexadecimal location 602E, which controls the output speed to the Console when the Shift or Shift-Lock key is pressed.
Default setting of CDELAY is 20H, which slows down the output by 43 milliseconds per character.
Setting CDELAY to 1 makes the output fastest: the delay is reduced to 1.3 milliseconds per character.
Storing a zero in CDELAY increases the delay to 343 ms per character, which is intolerably slow.

PRTABL see Section 05.05

A few other System Variables may be read, but should NOT be modified as this may cause the system to crash:

Loc.	Name	Contents
6010	CLOCK1	20 ms System Timer
6011	CLOCK2	5.12 sec System Timer
		These are stopped during Tape/Disk Input/Output.

Loc.	Name	Contents
6029	NXTBLK	Number of next tape block to be accessed. If -1, a Reset has occurred.
602A	TAPLEN	Number of blocks on a tape. Set to 42 decimal.
602C	CURPOS	Current cursor position.
602E	CDELAY	See above.
6057	MAXDSK	Number of disk drives connected to the system.
6058	CURDSK	Current (selected) disk drive.
605E	PRTABL	Address of Line Printer Character Conversion Table (see Section 05.05).

```
**************************************************************************
*                                                                        *
*  W A R N I N G:                                                        *
*                                                                        *
*  Modifying other Variables or Flags than those indicated               *
*  may lead to unexpected system crashes !!!!                            *
*                                                                        *
**************************************************************************
```

09 - THE MONITOR FOR TAPE BASED SYSTEMS

The Monitor is called by the Forth word MON .
It provides the following utilities:

Q(uit brings you back to Forth

F(ormat allows formatting P2000 Forth Source Tapes.
Formatting unused tapes or unused portions of tapes takes considerably more time than reformatting tapes that have been used before.
Shorter tapes will be formatted without an error message being issued (see section 06).

I(nspect returns the ASCII cassette type byte:

B = a Basic Tape	P = a program tape
F = a Forth Source Tape	V = a Viewdata tape
f = a Forth Runfile	W = a Word Processing Tape

If no type is shown, an unprintable code was found in the correspondino header location.

S(ave allows to save an application that has been placed in the Protected Forth Dictionary (See the example in Scr# 1 on track 8 of the Initializer Tape).
The Runfile thus created must be given a file name of one character in length. 'F' type blocks are overwritten and the cassette type is changed to 'f'.
There is no check against overwriting following files or Forth Source Blocks.
The intended use is to save loading and compiling tested application programs or utilities, by loading the compiled version, which then is immediately executed from COLD.

R(un loads and runs a file that was saved by a one character file name.
Search for the file starts at the next Tape Block.
Tape Error '?' is returned if the block type is 'F'.
If successful, the file is loaded and the routine jumps to COLD, thus erasing any buffers that might be in memory.

CAUTION: R(un-ning a file that was not properly installed may lead to a system crash.
Also keep Source Backups of your Run-files as you may need these for recompiling after a system reconfiguration or after your Initializing Tape has been updated.

10 - THE MONITOR FOR DISK BASED SYSTEMS

The Monitor is called by the Forth word MON .

It provides the following utilities:
DISK: F(ormat,C(onvert,S(ave,L(oad,R(eset
T(APE format,I(nspect,Q(uit

The T(ape format and I(inspect Routines have been described in Section 09.

Q(uit brings you back to Forth

S(ave and L(oad are nearly identical to the S(ave and R(un routines described in Section 09. which please consult. The difference is. that the files are stored on a disk by number in the first 35 tracks, as follows:

File:	#1	#2	#3	#4	#5	#6	#7
Track:	00-04	05-09	0A-0E	0F-13	14-18	19-1D	1E-22

F(ormat will format a disk to the capacity of the drive used if that drive was connected at initialization time. It will refuse to format a system disk. The routine will verify the accessability of all tracks and return Disk Error 1 if not successful.

C(onvert will convert disk with Philips/CPM interleaving to the P2000 Forth format or back (except System Disks): 1=Philips to Forth, 2=Forth to Philips

CAUTION: L(oad-ing a non existing file will crash the system.

WARNING: L(oad-ing files from either a 35/40 track disk in a 77 track drive or from a 77 track disk in a 35/40 Track Drive will also crash the system.

In all the above routines, when prompted for a Drive or File number, pressing the STOP key will bring you back to the start of the Monitor.

A disk may be protected against unintentional C(onvert-ing or F(ormat-ting by the following procedure:
0 LIST UPDATE F3 PREV @ 2+ C! FLUSH
It then will be treated as if it were a System Disk.
Using F0 (hex) instead of F3 will un-protect the disk.

11 - NON fig-FORTH WORDS USED IN P2000 FORTH

Implementing fig-Forth on the P2000 machine called for a number of special routines.

Most of these have been written in machine language or header-less code, but a few were interesting enough to be made available to the user:

In both the Tape and Disk versions:

LPT	for direct output to the Line Printer
MON	accesses the Monitor (see sections 09 and 10).
RESET	for resetting the current tape or disk drive.

In the Disk Version only:

TAPE	for selecting the tape drive on a disk based system

These words are explained in the Glossary in Section 12.

12 - Fig-FORTH GLOSSARY

This glossary groups the Forth words by function rather than in sorted order. An index is provided at the end of this document.

Throughout the glossary, the following abbreviations are used:

For numbers:

'n'	for signed single length numbers	(15 bits)
'u'	for unsigned single length numbers	(16 bits)
'd'	for signed double numbers	(31 bits)
'ud'	for unsigned double numbers	(32 bits)

For other values:

'addr' to indicate an address in memory
'c' to indicate a character
'b' to indicate a byte
'f' to indicate a Boolean false or true flag
(0=false, non-zero=true)
'cccc' for a sequence of one or more non blank characters forming a name

Pronunciations given between [] in accordance with the 'Forth Encylopedia' may be different from those given in 'Starting Forth'. Dialects even there!

Stack representation is: b3 b2 b1 ----- a2 a1
[b3 under b2 under b1, leaves a2 under a1], where b3..b1 are the values on the stack before execution, and a2..a1 those after execution of the word.
a1 and b1 are the values on the top of the stack.

12.01 - MANIPULATING THE COMPUTATION STACK

DROP n ----
Drop the number from the stack.

SWAP n1 n2 ---- n2 n1
Exchange the top two values on the stack.

DUP n ---- n n [dupe]
Duplicate the value on the stack.

OVER n1 n2 ---- n1 n2 n1
Copy the second stack value, placing it on top.

ROT n1 n2 n3 ---- n2 n3 n1 [rote]
Rotate the top three values on the stack, bringing the third to the top.

2DUP d ---- d d [two-dupe]
Duplicate the double number value on the stack.

-DUP n1 ---- n1 (n1) [dash-dupe]
Reproduce n1 only if non-zero. Usually used with IF to avoid an ELSE part to drop the value if zero.

S0 (---- addr) [s-zero]
Contains the initial value of the stack pointer.

SP@ (---- addr) [s-p-fetch]
Leaves the address of the stack position as it was before SP@ was executed.

SP! initializes the stack pointer from S0. [s-p-store]

0 is a constant leaving the value 0 on the stack.

1 is a constant leaving the value 1 on the stack.

2 is a constant leaving the value 2 on the stack.

3 is a constant leaving the value 3 on the stack.

BL Leaves the ASCII value for a "blank" (20H). [b-l]

?STACK see 2.15

12.02 - MANIPULATING THE RETURN STACK.

>R n ---- [to-r]
Remove a number from the computation stack and place it on top of the Return Stack. Use should be balanced with R> in the same definition.

R ---- n
Copy the top of the Return Stack to the computation stack

R> ---- n [r-from]
Remove the top value from the Return Stack and leave it on the computation stack.

RP@ ---- addr [r-p-fetch]
Leaves the current value in the return stack pointer.

RP! A procedure to initialize the return stack pointer from user variable R0 [r-p-store]

R0 (---- addr) [r-zero]
A user variable containing the initial location of the Return Stack.

12.03 - NUMBER BASES

BASE (---- addr)
Contains the current number base for input and output conversion (Default at power up is decimal).

HEX Set the numeric conversion base to 16 (hexadecimal)

DECIMAL Set the numeric conversion base for decimal I/O.

12.04 - COMPARISON

= n1 n2 ---- f [equal]
f is true if n1=n2.

< n1 n2 ---- f [less-than]
f is true if n1<n2.

U< u1 u2 ---- f [u-less]
Used for comparing addresses.

> n1 n2 ---- f [greater-than]
f is true if n1>n2.

0= n1 ---- f [zero-equal]
f is true if n1=0 (Effect as Logical NOT

0< n1 ---- f [zero-less-than]
Leave a true flag if n1 is negative.

MAX n1 n2 ---- max
Leave the greater of two numbers

MIN n1 n2 ---- min
Leave the smaller of two numbers.

12.05 - MACHINE MEMORY

! n addr ---- [store]
Store 16 bits of n at addr and addr+1.

@ addr ---- n [fetch]
Leave the 16 bit contents of addr and addr+1.

+! n addr ---- [plus-store]
Increment contents at addr and addr+1 by signed number n

C! n addr ---- [c-store]
Store 8 bits at addr.

C@ addr ---- byte [c-fetch]
Leave the 8 bit contents of memory address.

2! d addr ---- [two-store]
Store a 32 bit number at addr thru addr+3.

2@ addr ---- d [two-fetch]
Leave a 32 bit number from addr thru addr+3

? addr ---- n [question-mark]
Print the 16 bit value at addr and addr+1 in free format
according to the current base.

CMOVE addr1 addr2 u ---- [c-move]
Move the specified quantity of u bytes beginning at addr1 to addr2. P2000 Forth features an overwrite security so that moves may be safely executed in both directions (addr2=addr1+1 etc.).

FILL addr count byte ----
Fill memory at the address with the specified quantity of the byte. Will not fill if count is zero.

ERASE addr n ----
Clear a region of memory to zero from addr over n addresses.

BLANKS addr n ----
Fill an area of memory beginning at addr with blanks.

12.06 - STRUCTURE CONTROL (DO ... LOOP)

DO n1 n2 ----
Removes loop limit n1 and loop index n2 from the stack and repetitively executes the following words under control by and until LOOP or +LOOP.

LOOP selectively branches back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit.
The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead.

+LOOP executes in the same way as LOOP, but requires a signed increment value n3 just before +LOOP, which is added to the index prior to comparison of the index to the limit.

LEAVE forces termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index remains unchanged and execution continues until LOOP or +LOOP.

I copies the loop-index to the computation stack.

(LOOP) compiled by LOOP ; increment the loop index by 1 and test for loop completion.

(+LOOP) compiled by +LOOP ; increment the loop index by the number on the stack and test for loop completion.

(DO) Compiled by DO. Moves the loop control parameters to the return stack.

12.07 - STRUCTURE CONTROL (IF ...)

IF f ----
Occurs in a colon-definition in the form:
IF (true) .... THEN ....
or IF (true) .... ELSE .... THEN ....
After each part, execution resumes after THEN. If ELSE is missing, false execution skips to just after THEN.

ELSE is the header of the optional false part of IF.

THEN marks the conclusion of the conditional IF ... (ELSE) structure.

ENDIF is an alias for THEN

-DUP see 12.01
OBRANCH and BRANCH see 12.26

12.08 - STRUCTURE CONTROL (BEGIN .... REPEAT)

BEGIN marks the start of a sequence in the form:
BEGIN ... UNTIL. BEGIN ... AGAIN or
BEGIN ... WHILE ... REPEAT
that may be repetitively executed.

UNTIL controls the conditional branch back to the corresponding BEGIN as long as the flag on the stack is true. If f is false, execution returns to just after BEGIN.?

END is an alias for UNTIL

AGAIN forces execution to return to just after the corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop unless R> DROP is executed one level below. Rarely used.

WHILE selects conditional execution based on the flag on the top of the stack. If the flag is true WHILE continues execution of the true part through to REPEAT, which then branches back to BEGIN. If the flag is false, execution branches to just after REPEAT.

REPEAT forces an unconditional branch to just after the corresponding BEGIN.

12.09 - TERMINAL: INPUT

KEY ---- c
Wait for a key stroke and leave its ASCII value.

EXPECT addr count ----
Transfer characters from the keyboard to address until a "Return" or the count of characters have been received. One or more nulls are added at the end of the text.

WORD c ----
Read the next text characters from the input stream until a delimiter c is found, storing the character string beginning at the dictionary buffer HERE.
WORD leaves the character count in the first byte, then the characters, and ends with 2 or more blanks.
Leading occurrences of c are ignored.
If BLK is zero, text is taken from the terminal input buffer, else from the Disk or Tape Block stored in BLK.

QUERY Input 80 characters of text (or until a "return") from the keyboard. Text is placed at the address contained in TIB with IN set to zero.

IN ---- addr
A User Variable containing the byte offset within the current input text buffer (terminal or disk) from which the next text will be accepted.

TIB [t-i-b]
contains the address of the terminal input buffer.

?TERMINAL ---- f [question-terminal]
A true flag indicates that the STOP key has been struck. When accessed, the Terminal may go busy. See Section 05.

12.10 - INPUT FORMATTING

ENCLOSE addr1 c ---- addr1 n1 n2 n3
The text scanning primitive used by WORD. From the text address addr1 and an ASCII delimiting character c, it determines the byte offset to the first non-delimiter character (n1), the offset to the first delimiter after the text (n2) and the offset to the first character not included (n3). Will not process past an ASCII "null".

DIGIT c n1 ---- (n2) f
Converts ASCII character c using base n1 to its binary equivalent n2, with a true flag. Otherwise leaves only a false flag.

-TRAILING addr n1 ---- addr n2 [dash-trailing]
Adjusts the character count n1 of a text string beginning at addr to suppress output of trailing blanks.

NUMBER addr ---- d
Convert a character string left at addr with a preceding count to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given in DPL but no other effect occurs. If numeric conversion is not possible, an error message will be given.

(NUMBER) d1 addr1 ---- d2 addr2 [paren-number]
Convert the ASCII text beginning at addr+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.

DPL ---- addr [d-p-l]
A User Variable containing the number of digits to the right of the decimal point on double integer input. It may also hold the output column location of a decimal point in user generated formatting. The default value on single number input is -1.

12.11 - TERMINAL: NUMERIC OUTPUT

. n ---- [dot]
Print a signed number, converted according to the current value in BASE. A blank follows.

U. u ---- [u-dot]
Print an unsigned number, converted according to the the current value in BASE. A blank follows.

D. d ---- [d-dot]
Print a signed double number, converted according to the current value in BASE. A blank follows.

.R n1 n2 ---- [dot-r]
Print a signed number n1 right aligned in a field n2 characters wide. No blank follows.

D.R d n ---- [d-dot-r]
As .R, but for a signed double number.

12.12 - TERMINAL: TEXT OUTPUT

EMIT n ----
Transmit n as an ASCII character to the terminal. OUT is incremented for each character that is output.
The P2000T will accept color and graphics codes provided bit 7 of the code is set (add 128 decimal to the code). On the P2000M, this bit will often result in a flashing underline under a character or symbol.

CR [carriage-return]
Transmit a carriage return and line feed to the terminal

SPACE Transmit an ASCII blank to the terminal.

SPACES n ----
Transmit n spaces to the terminal

." Used in the form ." cccc" [dot-quote]
When used inside a definition, cccc compiles as an in-line string literal that at execution time will be sent to the terminal.
When used outside a definition, ." will execute immedia-tely.

C/L ---- n [c-slash-l]
A constant leaving the number of characters per line.

COUNT addr1 ---- addr2 n
Leave the byte addr2 and byte count n of a message text beginning at addr1. It is presumed that the first byte at addr1 contains the text byte count and that the actu-al text starts with the second byte.
Typically, COUNT is followed by TYPE or LPT.

TYPE addr count ----
Transmit (count) characters from addr to the Console.

LPT addr count ---- flag [l-p-t]
A P2000 Forth word transmitting (count) characters from addr to the line printer, bypassing the console.
If the printer is not on line, the procedure aborts and a true flag is placed on the stack.
Else a false flag (0) is placed on the stack.

MESSAGE n ----
In the tape version, print on the console message #n as stored in RAM.
In the disk version, print the text of line n relative to screen 4 of drive 0. n may then be positive or nega-tive.
If WARNING=0, the message will be printed as a number. A list of messages is given in Appendix 7.

12.13 - OUTPUT FORMATTING

<# [less-than-sharp]
Setup for pictured numeric output formatting using the words: <# # #S SIGN #>
The conversion is done on a double number producing text from just before PAD towards lower memory.

d1 ---- d2 [sharp]
Generates from a double number d1 the next ASCII character which is placed in an output string. Result d2 is the quotient after division by BASE, and is maintained for further processing.

#S d1 ---- d2 [sharp-s]
Generates ASCII text in the output buffer, by the use of #, until a zero double number d2 results.

HOLD c ----
Inserts an ASCII character into a pictured numeric output string. E.g. 2E HOLD will place a decimal point.

#> d ---- addr count [sharp-greater-than]
Terminates numeric output conversion by dropping d, leaving the text address and the character count suitable for TYPE.

SIGN n d ---- d
Stores an ASCII "-" sign at one end of a converted numeric output string when n is negative. n is discarded but d is maintained.

HLD ---- addr [h-l-d]
A user variable that holds the address of the latest character of text during numeric output conversion.

FLD ---- addr [f-l-d]
A user variable for control of number output field width that is presently not used in fig-FORTH.

OUT ---- addr
A user variable that contains a value incremented by EMIT. The user may examine and alter OUT to control display formatting. OUT is not set to 0 by CR.

PAD ---- addr
Leaves the address of the text output buffer, which is a fixed offset above HERE.

R# ---- addr [r-sharp]
A user variable which may contain the location editing cursor, or other file related function.

12.14 - COMPILER SECURITY

?COMP [question-compile]
Issues an error message if not compiling.

?CSP [question-c-s-p]
Issues an error message if the stack position differs from the value saved in CSP.

?ERROR f n ---- [question-error]
Issues an error message number n, if f is true.

?EXEC [question-execute]
Issues an error message if not executing.

?LOADING [question-loading]
Issues an error message if not loading.

?PAIRS n1 n2 ---- [question-pairs]
Issues an error message if n1<>n2, i.e. compiled conditionals do not match.

?STACK [question-stack]
Issues an error message if the stack is out of bounds.

ERROR line ---- in blk
Execute error notification and restart of Forth.

!CSP Save the stack pointer in CSP. Used as part of the compiler security. [store-c-s-p]

CSP ---- addr [c-s-p]
A user variable temporarily storing the stack pointer position, for compilation error checking.

SMUDGE see 12.26

WARNING ---- addr
A user variable containing a value controlling messages. If WARNING=1 and disk is present, screen 4 of drive 0 is the base location for messages. In a tape based system, the messages are stored in RAM.
If WARNING=0, messages will be presented by number.
If WARNING=-1, (ABORT) is executed for a user specific procedure.
Default power-up values in P2000 Forth are:
on a disk based system: 0
on a tape based system: 1

12.15 - DICTIONARY

' ---- addr [tick]
Used in the form: ' nnnn
Leaves the parameter field address of the dictionary word nnnn. As a compiler directive, it executes in a colon-definition to compile the address as a literal. If nnnn is not found after a search of CONTEXT and CURRENT, an appropriate message (?, #0) is given.

DP ---- addr [d-p]
A user variable, the dictionary pointer, which contains the address of the next free memory byte above the dictionary. May be read by HERE and altered by ALLOT.

HERE ---- addr
Leave the address of the next available dictionary location.

ALLOT n ----
Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or reorganize memory. n = the number of bytes to be ALLOTed.

FORGET is executed in the form FORGET cccc.
Deletes definition cccc from the dictionary with all entries physically following it, unless cccc is in the protected dictionary. Vocabulary pointers are adjusted.

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

TASK A no-operation word which can mark the boundary between applications. By forgetting TASK and recompiling, an entire application can be discarded.

SMUDGE see 12.26

TRAVERSE addr1 n ---- addr2
Move across a name field starting from addr1 which is the address of either the length byte or the last letter. If n=1 the motion is towards high memory; if n=-1, the motion is towards lower memory.

-FIND ---- (pfa) (b) f [dash--find]
Accepts the next text word (delimited by blanks) in the input stream to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, leaves a true flag, the length byte, and the parameter field address. Else leaves only a false flag.

(FIND) addr1 addr2 --- (pfa) (b) f [paren-find]
Searches the dictionary starting at the name field address addr2, matching to the text at addr1. If found, leaves a true flag, the length byte of the name field, and the parameter field address. Otherwise leaves a false flag.

CFA pfa ---- cfa [c-f-a]
Converts the parameter field address of a definition to its code field address.

LFA pfa ---- lfa [l-f-a]
Converts the parameter field address of a definition to its link field address.

NFA pfa ---- nfa [n-f-a]
Converts the parameter field address of a definition to its name field address.

PFA nfa ---- pfa [p-f-a]
Converts the name field address of a compiled definition to its parameter field address.

ID. addr ---- [i-d-dot]
Prints a definition's name from its name field address.

VLIST [v-list]
Lists the names of the definitions in the context vocabulary. In the P2000 Forth implementation, they are preceded by the definition's code field address.
Hitting STOP will terminate the listing.

WIDTH ---- addr
In fig-FORTH, a user variable containing the maximum number of letters saved in the compilation of a definition's name. It must be 1 .. 31, with a default value of 31. The name's full character count is saved in the length byte which is not included in the field specified by WIDTH.
Characters not saved are represented by dashes during ID. and VLIST.

+ORIGIN n ---- addr [plus-origin]
Leaves the memory address relative by n to the origin parameter area. n is expressed in bytes.
This definition is used to access or modify the boot-up parameters at the origin area.
The parameters are (with n expressed in hexadecimal):
n = 0C : Topmost word in the Dictionary
n = 0E : The rubout character emitted by the Terminal
n = 10 : Location of User's Area
n = 12 : Initial Computation Stack Pointer Value
n = 14 : Initial Return Stack Pointer Value

n = 16 : Terminal Input Buffer
n = 18 : Initial value of WIDTH
n = 1A : Initial value of WARNING
n = 1C : Initial Value of FENCE
n = 1E : Initial Value of DP
n = 20 : Initial Value of VOC-LINK

All these values should only be changed with the utmost caution, as most of them are used by COLD and the system may crash or behave in an unpredictable way if the pointers are not correctly set.

The following example shows, how the Editor is placed in the protected dictionary by modifying the boot-up parameters:

```
FORTH DEFINITIONS DECIMAL
LATEST          12  +ORIGIN  !    ( Top Name Field Address)
HERE            28  +ORIGIN  !    ( FENCE)
HERE            30  +ORIGIN  !    ( Dictionary Pointer)
' EDITOR  6  +  32  +ORIGIN  !    ( VOC-LINK)
```

When COLD is executed, the Editor will not be discarded.

HERE FENCE ! must be added in order to insure protection until the first time that COLD is executed.

12.16 - VOCABULARIES

VOCABULARY Used in the form VOCABULARY cccc.
Creates a vocabulary definition cccc. Subsequent uses of cccc make it the CONTEXT vocabulary which is searched first by INTERPRET.
The sequence "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary in which new definitions are placed. cccc will be so chained as to include all definitions of the vocabulary in which cccc itself is defined.
By convention, vocabulary names are to be declared IMMEDIATE.

CONTEXT ---- addr
A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

CURRENT ---- addr
A user variable containing a pointer to the vocabulary to which definitions are "currently" appended.

HERE see 12.15

LATEST ---- addr
Leaves the name field address of the topmost word in the CURRENT vocabulary.

VOC-LINK ---- addr [voke-link]
A user variable containing the address of a field in the definition of the most recently created vocabulary.

FORTH 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.

DEFINITIONS is used in the form cccc DEFINITIONS.
Sets the CURRENT vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.

12.17 - ARITHMETIC (SINGLE LENGTH

+	n1 n2	(n1+n2)	[plus]
-	n1 n2	(n1-n2)	[subtract]
*	n1 n2	(n1*n2)	[times]
/	n1 n2 ----	int(n1/n2)	[divide]

Leave the signed quotient of n1/n2.

/MOD n1 n2 ---- rem quot [divide-mod]
Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

MOD n1 n2 ---- mod
Leave the remainder of n1/n2. with the same sign as n

*/ n1 n2 n3 ---- n4 [times-divide]
Leave n4=n1*n2/n3. Retention of an intermediate 31 bit product permits greater accuracy than would be available with: n1 n2 * n3 /

*/MOD n1 n2 n3 ---- n4 n5 [times-divide-mod]
Leave quotient n5 and remainder n4, using the procedure described for */.

U* u1 u2 ---- ud [u-star]
Leave the unsigned double number product (32 bits) of two unsigned numbers.

U/ ud u1 ---- u2 u3 [u-slash]
Leave the unsigned remainder u2 and the unsigned quotient u3 from ud/u1.

1+ n1 n2 : increment n1 by 1. [one-plus]

2+ n1 n2 : leave n1 incremented by 2. [two-plus]

ABS n ---- u [absolute]
Leaves the absolute value of n as u.

+- n1 n2 ---- n3 [plus-minus]
Applies the sign of n2 to n1, which is left as n

MAX see 12.04
MIN see 12.04

12.18 - ARITHMETIC (DOUBLE LENGTH)

D+ d1 d2 ---- dsum [d-plus]
Leave the double number sum of d1 and d2.

D+- d1 n ---- d2 [d-plus-minus]
Apply the sign of n to d1, leaving it as d2.

DABS d ---- ud [d-absolute]
Leave the absolute value of a double number.

S->D n ---- d [s-to-d]
Sign extend a single number to form a double number

12.19 - ARITHMETIC (MIXED LENGTH)

M* n1 n2 ---- d [m-star]
Leave the double number signed product of two signed numbers.

M/ d n1 ---- n2 n3 [m-slash]
Leave the signed remainder n2 and signed quotient n3 from d/n1.

M/MOD ud1 u2 ---- u3 ud4 [m-slash-mod]
Leave a double unsigned quotient ud4 and remainder u3 from ud1/u2.

12.20 - LOGICAL

AND n1 n2 --- n3
Leave the bitwise logical and of n1 and n2 as n3.

OR n1 n2 --- n3
Leave the bitwise logical or of n1 and n2 as n3.

XOR n1 n2 --- n3 [x-or],[exclusive-or],[x-o-r]
Leave the bitwise logical exclusive-or of n1 and n2 as n3.

MINUS n1 ----- n2 : Leave the two's complement of n1

DMINUS d1 ---- d2 [d-minus]
Convert d1 to its double number two's complement d2.

TOGGLE addr b ----
Complement the contents of addr by bit pattern b.

12.21 - VIRTUAL I/O: MEMORY ALLOCATION

EMPTY-BUFFERS clears all block-buffers and marks them as empty (in such a way that Scr # 0 can be listed right away).

UPDATE marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disk or tape should its buffer be required for the storage of a different block.

BLOCK n ---- addr
Leave the memory address of the block buffer containing block n. If the block is not ready in memory it is transferred from disk or tape. Any updated block occupying that buffer is first saved to the selected drive.

BUFFER n ---- addr
Obtain the next memory buffer, assigning it to block n. First save any updated contents of the buffer. The block is not read from disk or tape.

+BUF addr1 ---- addr2 f [plus-buf]
Advance the buffer address addr1 to the address of the next buffer addr2. f is false when addr2 is the buffer presently pointed to by PREV.

FLUSH Saves any updated screens onto tape or disk. Used by EMPTY-BUFFERS.
Contrary to the Fig-FORTH model, FLUSH sets all block numbers to 7FFF, thus enabling Screen # 0 to be LISTed.

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

LIMIT ---- n
A constant leaving the address just above the highest memory available for disk bufers. Usually this is the highest system memory.

PREV ---- addr [prev]
A variable containing the address of the disk buffer most recently referenced. The UPDATE command marks this buffer to be later written to disk or tape.

USE ---- addr
A variable containing the address of the block buffer to use next, as the least recently written.

12.22 - VIRTUAL I/O: INTERPRETATION & DOCUMENTATION

LIST n ----
Display the ASCII text of screen n of tape/disk on the Console. SCR contains the screen number during and after this process.

LOAD n ----
Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S.

--> Continue interpretation with the next disk or tape screen. [next-screen]

LINE line scr ---- [dot-line]
Print on the terminal a line from the storage device by its line and screen number.

(LINE) n1 n2 ---- addr count [paren-line]
Convert the line number n1 and the screen number n2 to the buffer address containing the data. A count of 64 indicates the full line text length.

INDEX from to ----
Prints the first line of each screen over the range from.to. This is used to view the comment lines of an area of text on disk or tape screens.

TRIAD scr ----
Display on the terminal the three screens which include that number scr, beginning with a screen number divisible by three: 3 TRIAD, 4 TRIAD and 5 TRIAD will all list screens 3 thru 5.
Output of the screens is terminated by a reference line taken from Message # 15.

12.23 - VIRTUAL I/O: FORTH DISK/TAPE I/O

R/W addr blk flag ---- [r-slash-w]
The fig-FORTH standard disk read-write linkeage.
addr : specifies the source or destination block buffer.
blk : is the sequential number of the referenced block.
flag : is 0 for a write operation and 1 for reading.
R/W determines the location on disk or tape, performs the read-write and performs any error checking.
In P2000 Forth, an error message is issued in case the operation was not successfull (see sections 06 and 07) and the system will wait for either:
the RETURN key for a retry, or
the STOP key as a command to abort the operation.
RESET is used internally both in case of a Disk and a Tape R/W Error.

SCR ---- addr [s-c-r]
A user variable containing the last screen number referenced by LIST.

BLK ---- addr [b-l-k]
A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer. (As a consequence, it is not possible to LOAD from Scr # 0).

OFFSET ---- addr
A user variable which may contain a block offset to disk drives. BLOCK adds the contents of OFFSET to the number on the stack. Messages by MESSAGE are independent from OFFSET.

B/BUF ---- n [bytes-per-buffer]
This constant leaves the number of bytes per disk or tape buffer, in P2000 Forth 1024 decimal.

B/SCR ---- n [blocks-per-screen]
A constant leaving the number of disk/tape blocks per editing screen (1 in P2000 Forth).
By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each.

#BUF ---- n [sharp-buf],[number-buf]
A constant returning the number of disk/tape buffers allocated.
In P2000 Forth, the following numbers of buffers are allocated depending on the RAM size:
16 K : 2 Buffers
32 K : 6 Buffers
48 K : 8 Buffers

12.24 - P2000 TAPE I/O

RESET is a P2000 Forth word that is used to signal to the Tape Operating System that the cassette in the tape drive has been changed or turned over.
It is ESSENTIAL to use this word, as the TOS itself does a minimum of checking in order not to further decrease the speed of an operation that is rather slow in itself. NOT USING this word after a tape change may cause data to get lost.

12.25 - P2000 DISK I/O

RESET is also used internally by R/W for resetting disk drives having an I/O error. Direct use is rare.

DR0
DR1
DR2
DR3 are words selecting disk drives by presetting OFFSET and resetting the flag that would have been set by the word TAPE. The cumulative capacities of the drives lower than the selected drive are stored in OFFSET.
Selecting a non existent or non implemented drive will generate the message ' I/O Error 3 ' meaning that the drive is not available.
[d-r-zero],[d-r-one],&c.

TAPE is a P2000 Forth word selecting the Tape Drive as the current drive on disk based systems.
It had to be implemented in order to avoid generating messages from the tape drive.

12.26 - COMPILER DEFINING WORDS

[colon]
Used in the form called a colon-definition:
: cccc .... ;
Creates a dictionary entry defining cccc as equivalent to the following sequence of Forth word definitions .... until the next ';' or ':CODE'.

[semicolon]
Terminate a colon-definition and stop further compilation. Compiles the run-time ;S

(Used in the form (cccc) [left-paren]
Ignore a comment that will be delimited by the first occurence of a right parenthesis in the input stream.
A blank after (is required.

;S [semicolon-s]
Stop interpretation of a screen. Do not continue with the next screen. It is also the run-time word compiled at the end of a colon-definition which returns execution to the calling procedure.

;CODE [semicolon-code]
Used in the form : cccc .... ;CODE Assembler Mnemonics.
Stop compilation and terminate a new defining word cccc by compiling (;CODE) and assemble the following mnemonics to machine code. When cccc later executes in the form 'cccc nnnn' the word nnnn will be created with its execution procedure given by the machine code following cccc (when nnnn is executed it jumps to the code after nnnn).
An existing defining word must exist in cccc prior to ;CODE. Also, the Forth Assembler must be resident when ;CODE is being executed.
A low level equivalent of <BUILDS ... DOES>.

CONSTANT Used in the form n CONSTANT cccc to create a word cccc that, when it is executed, will push the value n onto the stack.

VARIABLE Used in the form n VARIABLE cccc.
When cccc is later executed, the address of its parameter field containing n is pushed on top of the stack so that a fetch or store may access this location.

CREATE is used in the form CREATE cccc by such words as CODE and CONSTANT to create a dictionary header for a Forth definition. The code field contains the address of the word's parameter field.
The new word is created in the CURRENT Vocabulary.

, n ---- [comma]
Store n into the next available dictionary memory cell (16 bits), advancing the dictionary pointer by two.

C, n ---- [c-comma]
Store 8 bits into the next available dictionary memory byte, advancing the dictionary pointer by one.

<BUILDS Used within a colon-definition:
: cccc <BUILDS .... DOES> .... ;
Each time cccc is executed, <BUILDS defines a new word with a high-level execution procedure.
" cccc nnnn " uses <BUILDS to create a dictionary entry for nnnn. When nnnn is executed, it will execute the words after DOES> in cccc.

DOES> defines the run-time action within a high-level defining word. Typical uses include the Forth Assembler, multi-dimensional arrays and compiler generation.

LIT is, within a colon-definition, automatically compiled before each 16 bit literal number encountered in input text. Later execution causes the contents of the next dictionary address to be pushed on the stack.

LITERAL n ----
Compile the stack value n as a 16 bit literal. LITERAL is immediate. The intended use is:
: xxx [calculate] LITERAL ;

DLITERAL [d-literal] is the double number version of LITERAL.

IMMEDIATE Following a definition, compels it to be executed when encountered during compilation.
Such a definition can be forced to compile by preceding it with [COMPILE].

COMPILE When the word containing COMPILE executes, the address of the word following COMPILE is copied (compiled) into the dictionary.

[COMPILE] is used in a colon-definition in the form:
: cccc [COMPILE] FORTH ;
and will force the compilation of an immediate definition. The above example will select the FORTH vocabulary when cccc executes, rather than at compile time.
[bracket-compile]

[Used in a colon definition. The words after [are executed, not compiled. [left-bracket]

] Resume compilation to the completion of a colon-definition. [right-bracket].

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 error.

STATE ---- addr
A user variable containing the compilation state. A non-zero value indicates compilation.

BACK addr ----
Calculates the backward branch offset from HERE to addr and compiles it into the next available dictionary memory address.

OBRANCH Compiled by IF, UNTIL and WHILE [zero-branch]
If the flag on top of the stack is false, the following in-line parameter is added to the interpretitive pointer to branch ahead or back.

BRANCH Compiled by ELSE, AGAIN, REPEAT
An in-line offset is added to the interpretitive pointer to unconditionnally branch back or ahead. [zero-branch]

(;CODE) compiled by ;CODE : rewrite the code field of the most recently defined word to point to the following machine sequence. [paren-semicolon-code]

(.") [paren-dot-quote]
The run-time procedure compiled by ."
Transmits the following in-line text to the Terminal.

12.27 - SYSTEM WORDS

QUIT Clears the return stack, stops compilation and returns control to the operator's terminal. No message is given. QUIT is an endless loop and the primary loop of the Forth System.

ABORT clears the stacks and enters the execution state. Return control to the operators terminal, printing a message appropriate to the installation.

(ABORT) executes after an error when WARNING is -1. Normally executes ABORT but may be altered (with care) to a user's alternative procedure. [paren-abort]

EXECUTE addr ----
Executes the definition whose code field address is on the stack. The code field address is also called the compilation address.

USER n ----
Used in the form n USER cccc
Creates a user variable cccc. The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable.
cccc later executes as a variable name, placing the storage address on the stack.
The User Variable space provided allows for 7 more user variables in addition to those provided by fig-Forth, at offsets 32H through 3EH.

MON [m-o-n] accesses the Monitor. See sections 09 and 10.

NOOP [no-op] is the Forth no-operation word, mostly used to allow for later linking an application by replacing the reference to NOOP by the code field address of the application.

SP! see 12.01
RP! see 12.02

WARM executes a restart of the application. Disk buffers are erased.

COLD adjusts the dictionary pointer to the minimum loaded and restarts via ABORT. May be called from the terminal to remove application programs and restart Forth.

12.28 - HARDWARE

P! n1 n2 ---- [p-store]
Output byte n1 to port n2.

P@ port ---- n [p-fetch]
Leaves the value stored at port.

.CPU [dot-c-p-u]
Prints the processor name from the system parameters.

12.29 - OUTER INTERPRETER

INTERPRET is the outer (text) interpreter which sequentially executes or compiles text from the input stream. Text input will be taken according to the convention for WORD. If a decimal point is found as part of a number, a double number value will be left. See NUMBER.

NULL is pronounced [null] and sometimes called 'X'. It is a zero byte compiled as a legal Forth word and causes an exit from the endless loop in INTERPRET.

QUERY see 12.09
NUMBER see 12.10
QUIT see 12.27
STATE see 12.26
[and] see 12.26

12.30 - INNER INTERPRETER

NEXT is the inner interpreter of Forth that uses the interpretitive pointer IP to execute compiled definitions. It is not directly executed but is the return point for all code procedures. It acts by fetching the address pointed to by IP, storing this value in the Forth register W. It then jumps to the address stored at the location pointed to by W.

Jumping to NEXT-1 pushes the contents of the HL register pair on the stack; jumping to NEXT-2 first places the contents of DE, then the contents of HL, on the stack.
In 8080/Z80 implementations, the address of NEXT is placed on the stack by the sequence:

```
' ;S 0B + @
```

IP [i-p] is the Interpretative Pointer, which is contained in the BC register pair.
When using machine language routines within Forth code, great care must be taken to save the contents of the BC register pairs, as otherwise programs will behave in an erratic way, if they do not crash altogether.
Do not use the EXX instruction unless you are sure that the primitives called do not use this same instruction.

W [w] is a logical Forth entity used for "threading" Forth words. It is contained in the DE register pair but needs not to be saved prior to calling a machine language routine.

13 - THE fig-FORTH EDITOR

The Forth Editor and Extensions to the Editor are supplied in source form on track 8 of the P2000 Forth Initializer Tape. SCR # 0 lists the contents of the track.

13.01 - SCREEN SELECTION AND TEXT INPUT

LIST (scr# ----) lists a screen and selects it for editing. This is the standard Forth word LIST.

CLEAR (scr# ----) clears a screen and selects it for editing

P (line# ----) Put following text, separated from P by a space, in line# of selected screen.
Example: 0 P This is how to input
1 P Text to lines 1, 2 and 3
2 P of the selected Screen.

13.02 - LINE EDITING COMMANDS

H (line# ----) holds the line at PAD (useful for transferring lines between screens).

D (line# ----) deletes the line but holds it at PAD. Line 15 becomes blank and lines below the deleted line move up one line.

T (line# ----) types the line and holds it at PAD.

R (line# ----) replaces the line with the line at PAD.

I (line# ----) inserts the line at PAD at line#, moving down all following lines. Line 15 is lost.

E (line# ----) erases the line with blanks.

S (line# ----) inserts a line of blanks, moving down the subsequent lines. Line 15 is lost.

- CURSOR POSITIONING

TOP positions the cursor at the start of the screen.

M (n ----) moves the cursor by the signed amount of n positions. The position of the cursor on its line is shown by a '↑'.

WHERE is used when an error occurs at loading time. It takes the scr# and the offset into the screen from the stack and uses these to print the screen and line numbers and a picture of where the error occured.

13.04 - STRING EDITING

F cccc searches forward from the current cursor position until the string cccc is found. The cursor is left at the end of the string and the cursor line is printed.
If the string is not found Error Message [huh?] is given and the cursor is repositioned at the top of the screen.

B is used to back up the cursor by the length of the text at PAD. Can be used after F, C &c.

N finds the next occurrence of the string at PAD. Mostly used after F but may take an argument left by C or X.

X cccc finds and deletes the first occurrence of cccc.

C cccc copies in text cccc at the current cursor position, moving the remainder of the line. Text beyond the 64th position of the line is lost.
Typing C with no text will copy a null into the text at the cursor position. This will abruptly stop later compiling. To delete the error type TOP X without text.

TILL cccc deletes on the cursor line from the cursor till the end of the string cccc.

DELETE n ----
Deletes n characters in front of the cursor, moving forward, the remainder of the line. It will not move any characters into the previous line, but it will erase characters from the end of the previous line forward if n is greater than the cursor offset into the line where deletion started.
Also used by X and TILL within the limits of 1 line.

13.05 - EDITING FORTH SCREENS

COPY (from to ----) copies the contents of one screen to the other by replacing the 'from' screen number by the 'to' screen number and setting the Update Flag.
A screen can be renumbered as long as it has not been FLUSHed. Multiple copies from one screen to several new screens must be separated by FLUSH commands.
This implementation differs from the fig-Forth Editor COPY that calls FLUSH. This has been intentional in order to avoid multiple tape movements.

L (scr# ----)
Lists the current screen and the cursor line.

13.06 - EDITOR GLOSSARY

Words not described in earlier subsections:

LINE n ---- addr
Leave the address of line n of current screen. This will be an address in the disk/tape buffer area.

TEXT c ----
Accept following text to PAD. c is the text delimiter.

R# is a Forth user variable which contains the offset of the editing cursor from the start of the screen.

#LOCATE ---- n1 n2 [sharp-locate]
Translates R# into line no. n2 and offset into the line n1.

#LEAD ---- line-address offset-to-cursor [sharp-lead]

#LAG --- cursor-address count-till-end-of-line [sharp-lag]

-MOVE addr line-no ---- [dash-move]
Move a line of text from addr to line-no in the current screen.

1LINE ---- f [one-line]
Scan the cursor line for a match to the text in PAD. If f=false, R# points to the beginning of the next line.

FIND searches for a match to the string in PAD, from the cursor position till the end of the screen. If no match is found, it issues error message 0 and repositions the cursor at the top of the screen.

MATCH adr1 n1 adr2 n2 ---- f n3
Matches the string at adr2 with all strings on the cursor line forward from the cursor address adr1.
n1 is the number of bytes left till the end of the cursor line and n2 is the string count.
If f is true, n3 allows the cursor R# to be updated to the end of the matching text.
If f is false, n3 contains the number of bytes left until the start of the next line.
May also be used by other words for scanning blocks of data, up to 65536 bytes in length. The block length then must be in n1, and adr1 is the address where the search is to start.
n3 will contain the offset to the end of the block from where the search ended.

14 - A FORTH ASSEMBLER FOR THE Z80

There has been no official release of a fig-Forth Assembler for the 8080/Z80 family of microprocessors. John Cassady has written an Assembler for the 8080 that was published in Vol.III of 'Forth Dimensions', the journal of the US Forth Interest Group.

The Assembler provided with P2000 Forth reproduces most of the features of John Cassady's Assembler. however it has been extended by a number of definitions that allow forward label references for relative condtional and unconditional jumps.

The intended use for the Assembler is to enable users to write routines in machine language that later can be_incorporated as "Comma-ble" code in load screens without having the Assembler resident. thus making the best possible use of available memory even on a 16k system.

A demonstration of the Assembler is given on track 8 of the Initializing Tape (see Entry "Assembler Demo" in Scr# 0) with the resulting code in the Low Level Match Screen of the Editor, where it was put by the word MEM, of the Editor Extensions.

No structure controls are provided as this would get in the way of the principle of creating comma-ble relocatable machine code executing as fast as possible.

We hope you can get used to its quirks and would like to have your comments on how it works.

14.01 - INSTRUCTION SET

Remember, Forth is a stack oriented language, and so is an Assembler written in Forth.
The whole of Forth is at your disposal when writing an Assembly routine, so you can have Forth put an address on the stack or calculate a value and put it on the stack, where it will be taken by the Assembler for loading a register.

The rather incoherent mnemonics used for the Z80 made it necessary to 'invent' a number of new mnemonics. For that reason the instruction set will be treated by groups of functions.

```
Z80 mnemonic:       Forth Assembler:

LOAD Instructions:

LD  reg,byte        byte reg  LDRI (Load Register Immediate)
LD  dest,src        src  dest  LD
LD  A,(addr)        addr  LDA      (Load A)
LD  (addr),A        addr  STA      (Store A)
LD  A,(rp)          rp  LDAX       (an 8080 mnemonic)
LD  (rp),A          rp  STAX       (an 8080 mnemonic)

LD  SP,rp           not implemented. Comma in!
LD  rp,data         data rp  LDPI  (Load Pair Immediate)
LD  rp,(addr)       addr rp  LDPM  (Load Pair from Memory)
LD  (addr),rp       addr rp  STPM  (Store Pair to Memory)

POP rp              rp  POP
PUSH  rp            rp  PUSH

EX  DE,HL           HL  DE  EX
EX  AF,AF'          AF' AF EX
EXX                 EXX
```

Data Block Instructions:

LDI, LDIR, LDD, LDDR, CPI, CPIR, CPD and CPDR take no arguments and are the same in the Forth Assembler.

Logic Instructions:

```
AND  reg            reg   AND
AND  byte           byte  ANDA
```

CP, OR and XOR follow this pattern.

Arithmetic Instructions:

```
DEC  reg            reg   DEC
DEC  rp             rp   DECP    (Decrement Pair)
ADD  A,reg          reg  ADD
ADD  A,byte         byte  ADDA
ADD  rp,rp          rp  rp  ADDP    ( Add Pairs)
```

INC, ADC, SUB and SBC follow the same pattern.

DAA, CPL, NEG, CCF and SCF remain unchanged.

Shift and Rotate Instructions:

```
RLC  reg            reg  RLC
```

RRC, RL, RR, SLA, SRA and SRL follow this pattern.
RLCA, RRCA, RLD and RRD take no argument and are unchanged.

Bit Manipulation:

```
BIT  n,reg          reg  n  BIT
```
SET and RES follow the same pattern.

Jumps and Calls:

```
CALL  addr          addr  CALL
JP    addr          addr  JP
```
Conditional JP and CALL have not been implemented.

```
JP    (HL)          JPHL
JP    (IX)          JPIX
JP    (IY)          JPIY
```

Relative Jumps:

```
JR    addr          labelname JR
JR    Z,addr        labelname JRZ
```

JRNZ, JRC, JRNC follow the same pattern.
'labelname' may be a signed offset or a forward reference to label that has been previously declared by RLABEL: .

RET, RETI and RETN are unchanged.

Restart:

```
RST  n              n  RST
```

Addressing Ports:

```
IN   A,(port)       port  IN
IN   reg,(C)        reg   CIN
OUT  A,(port)       port  OUT
OUT  (C),reg        reg   COUT
```

INI, OUTI, INIR, OTIR, IND, OUTD, INDR, OTDR are unchanged.

Interrupt Modes:

```
IM  byte            byte IM
```

NOP, HALT, DI and EI are unchanged

Displacements with IX and IY register pairs:

The syntax is as follows:
```
LD   A,(IX+disp)    disp (IX  A  LD
BIT  n,(IX+disp)    disp (IX  n  BIT
```
It is important to omit the left parenthesis.

14.02 - GLOSSARY

CODE used in the form CODE cccc ..... C;
Invokes the Assembler which assembles the mnemonics after cccc until C; is encountered.

C; [c-semicolon] terminates assembly of the preceding mnemonics.

LABEL used in the form LABEL cccc
Places the dictionary address of cccc in a variable for later reference. When cccc executes, the address is placed on the stack.

RLABEL: is used in the form RLABEL: cccc
It declares a label cccc for relative jumps prior to it's use. The number of allowed forward references is contained in the constant #REF.
When cccc later executes in a forward reference, the location of reference is stored for later use by RLABEL.
After RLABEL has resolved all forward references, executing cccc will place the backwards displacement on the stack for use by JR, DJNZ, etc.
Error message # 9 will be issued if the displacement is out of the range -129<disp<128.
It is possible to use cccc more than once, as declaring a label that has been declared before will only clear the contents of the array.

RLABEL is used in the form RLABEL cccc to mark the address of cccc .
It resolves all forward declarations for cccc and places the address of cccc in a variable for ulterior relative backward references.

#REF ---- n [sharp-r-e-f]
A constant containing the number of forward references allowed. If the number of forward references exceeds the value, Error message # 10 is issued.

NEXT is an assembly macro executing a jump to NEXT.

PUSH1 is an assembly macro executing a jump to NEXT-1, pushing the contents of HL onto the stack.

PUSH2 is an assembly macro executing a jump to NEXT-2, pushing the contents of DE, and then the contents of HL onto the stack.

KEYBOARDS AND CHARACTER GENERATORS

For some versions of the P2000, we had to compromise between key top inscriptions and the US-ASCII codes used by P2000 Forth for internal storage.
The following pages list the differences, not only for the Keyboards but also for the Character Generators and the Printers.

Codes returned by the Keypad:			Relative positions in the Keyboard Tables:		
AF	AA	0C	73	72	70
AD	AB	18	2B	2A	28
14	04	19	7B	7A	78
B7	B8	B9	33	32	30
1D	13	1C	8B	8A	88
B4	B5	B6	43	42	40
17	11	12	83	82	80
B1	B2	B3	3B	3A	38
1A	10	03	5B	5A	58
30	B0	AE	13	12	10

The codes that have bit 7 set correspond to ASCII characters that appear more than once on the keyboard. As the 7th bit is reset by (KEY), this is transparent to the user unless (KEY) has been patched as described in section 05.02.
The other codes are control codes, and of these the following are free for use by an application:
0C, 17, 19, 1A, 1C and 1D.

Relative positions for main keyboard:

```
 68  76  87  4C  4F  4D  49  4E  7E  71  75  77  8C  74
 20  2E  3F  04  07  05  01  06  36  29  2D  2F  44  2C

  50  4B  6B  6C  6F  6D  69  6E  8E  79  7D  7F  84  ****
  08  03  23  24  27  25  21  26  46  31  35  37  3C  *7C*
                                                      ***
    *** 6A  43  54  57  55  51  56  86  89  8D  8F  5C  34*
    *** 22  0B  0C  0F  0D  09  0E  3E  41  45  47  14  ***

*** 62  52  63  64  67  65  61  66  5E  81  85   ***
*** 1A  0A  1B  1C  1F  1D  19  1E  16  39  3D   ***

     48  4A  ------------ 59 ------------ 5D  5F
     00  02  ------------ 11 ------------ 15  17
```

The keyboard Table is located at LPTAB - 9B (hexadecimal)

Version: United Kingdom and Netherlands

Keyboard

Where possible. ASCII characters returned are shown, otherwise the hexadecimal code that is returned.

The keyboard has key top inscriptions for all characters used in P2000 Forth.

```
 9B      "  00  $  %  &  '  (  )  =  00  00  88
 1B      2  3  4  5  6  7  8  9  0  -  00  08

  1F  Q  W  E  R  T  Y  U  I  O  P  00  [  ****
  09  q  w  e  r  t  y  u  i  o  p  @  ]  *BD*
                                          ***
   ***  A  S  D  F  G  H  J  K  L  +  *  00  0D*
   ***  a  s  d  f  g  h  j  k  l  ;  :  #  ***

***     Z  X  C  V  B  N  M  AC  AE  ?  ***
***     z  x  c  v  b  n  m  ,  .  /  ***

    05  15  ------------ A0 ------------ 0B  06
    01  ↑  ------------ 20 ------------ 0A  02
```

Screen

On the P2000T, the brackets [and] are shown as left and right arrows.
In all other cases, the Forth character set is displayed.

Matrix Printers

On standard US-ASCII printers all characters are printed as per the P2000 Forth character set.

On the P2123 printer, [and] are printed as left and right arrows.

Philips Daisy Wheel Printer

With a BIL printing wheel, the entire set is printed.

With an ESA wheel, [and] are printed as { and } and @ is printed as ° (degree sign).

Version: Germany and Austria

Keyboard

Where possible, ASCII characters returned are shown, otherwise the hexadecimal code that is returned.

For a number of characters used in P2000 Forth, keys with different inscriptions had to be used.

```
 9B   !   "  00   $   %   &   /   (   )   =   ?  00  88
 1B   1   2   3   4   5   6   7   8   9   0  00  00  08

  1F   Q   W   E   R   T   Z   U   I   O   P  00   *  ****
  09   q   w   e   r   t   z   u   i   o   p   Ø   +  *8D*
                                                      ***
   ***  A   S   D   F   G   H   J   K   L  00  00   '  0D*
   ***  a   s   d   f   g   h   j   k   l   Ä   Ö   #  ***

***   >   Y   X   C   V   B   N   M   ,   :  00  ***
***   <   y   x   c   v   b   n   m   .   -  00  ***

     05  15  ---------- A0 ---------- 08  06
     01   ↑  ---------- 20 ---------- 0A  02
```

Screen

The following three Forth characters are replaced by characters proper to the German version of both the P2000T and the P2000M:
Ø becomes Ü [becomes Ö] becomes Ä

Matrix Printers

On standard US-ASCII printers all characters are printed as per the P2000 Forth character set.

On the P2123 printer:
Ø becomes Ü [becomes Ö] becomes Ä

Philips Daisy Wheel Printer

With a BIL printing wheel, the entire set is printed.

With an ESA wheel, [and] are printed as { and } and Ø is printed as ° (degree sign).

Version: France

Keyboard

Where possible, ASCII characters returned are shown, otherwise the hexadecimal code that is returned.

For a number of characters used in P2000 Forth, keys with different inscriptions had to be used.

9B	1	2	3	4	5	6	7	8	9	0	[	]	88
1B	$	00	"	'	(	-	00	00	00	ə	)	!	08
1F	A	Z	E	R	T	Y	U	I	O	P	00	&	$$$$
09	a	z	e	r	t	y	u	i	o	p	00	#	$8D$
													$$$
$$$	Q	S	D	F	G	H	J	K	L	M	%	00	0D$
$$$	q	s	d	f	g	h	j	k	l	m	00	*	$$$
$$$	>	W	X	C	V	B	N	?	.	/	+	$$$	
$$$	<	w	x	c	v	b	n	,	;	:	=	$$$	

05 15 ---------- A0 ---------- 0B 06
01 ↑ ---------- 20 ---------- 0A 02

Screen (P2000M only)

The left bracket [is represented by ° (degree sign)
The right bracket] is represented by § (section sign)
In all other cases, the Forth character set is displayed.

Matrix Printers

On standard US-ASCII printers all characters are printed as per the P2000 Forth character set.

On the P2123 printer:
ə becomes à [becomes °] becomes §

Philips Daisy Wheel Printer

With a BIL printing wheel, the entire set is printed.

With an ESA wheel, [and] are printed as { and } and ə is printed as ¨ (tréma).

Version: **Italy**

Keyboard

Where possible, ASCII characters returned are shown, otherwise the hexadecimal code that is returned.

For a number of characters used in P2000 Forth, keys with different inscriptions had to be used.

9B	1	2	3	4	5	6	7	8	9	0	00	+	88
1B	$	00	"	'	(	[	00	00	]	@	)	-	08
1F	Q	Z	E	R	T	Y	U	I	O	P	=	&	$$$$
09	q	z	e	r	t	y	u	i	o	p	00	#	$8D$
													$$$
$$$	A	S	D	F	G	H	J	K	L	M	%	00	0D$
$$$	a	s	d	f	g	h	j	k	l	m	00	$	$$$
$$$	>	W	X	C	V	B	N	?		/	!	$$$	
$$$	<	w	x	c	v	b	n	.		:	00	$$$	
	05	15	----------			A0	----------			0B	06		
	01	↑	----------			20	----------			0A	02		

Screen (P2000M only)

The entire Forth character set is displayed.

Matrix Printers

On standard US-ASCII printers all characters are printed as per the P2000 Forth character set.

On the P2123 printer, [and] are printed as left and right arrows.

Philips Daisy Wheel Printer

With a BIL printing wheel, the entire set is printed.

With an ESA wheel, [and] are printed as (and) and @ is printed as ° (degree sign).

Version: Spain

Keyboard

Where possible, ASCII characters returned are shown, otherwise the hexadecimal code that is returned.

For a number of characters used in P2000 Forth, keys with different inscriptions had to be used.

9B		&	00	$	%	00	#	*	(	)	00	+	88
1B		2	3	4	5	6	7	8	9	0	-	=	08
1F	0	W	E	R	T	Y	U	I	O	P		>	****
09	q	w	e	r	t	y	u	i	o	p		<	*8D*
													***
***	A	S	D	F	G	H	J	K	L	00	[	00	0D*
***	a	s	d	f	g	h	j	k	l	@	]	00	***
***	:	Z	X	C	V	B	N	M	AC	AE	?	***	
***	:	z	x	c	v	b	n	m	.	.	/	***	

05	15	----------	A0	----------	0B	06
01	↑	----------	20	----------	0A	02

Screen (P2000M only)

The entire Forth character set is displayed.

Matrix Printers

On standard US-ASCII printers all characters are printed as per the P2000 Forth character set.

On the P2123 printer, [and] are printed as left and right arrows.

Philips Daisy Wheel Printer

With a BIL printing wheel, the entire set is printed.

With an ESA wheel, [and] are printed as { and } and @ is printed as ° (degree sign).

Version: Sweden and Finland

Keyboard

Where possible, ASCII characters returned are shown, otherwise the hexadecimal code that is returned.

For a number of characters used in P2000 Forth, keys with different inscriptions had to be used.

9B		"	#	$	%	&	/	(	)	=	?	00	88
1B		2	3	4	5	6	7	8	9	0	+	00	08
1F	Q	W	E	R	T	Y	U	I	O	P	00	00	****
09	q	w	e	r	t	y	u	i	o	p	@	00	*8D*
													***
***	A	S	D	F	G	H	J	K	L	00	00	*	0D*
***	a	s	d	f	g	h	j	k	l	[	]	'	***
	Z	X	C	V	B	N	M	;	:	00	***		
	z	x	c	v	b	n	m	,	.	-	***		
05	15	----------					A0	----------		0B	06		
01	↑	----------					20	----------		0A	02		

Screen

On the P2000T, the brackets [and] are shown as Ö and Ä
On both models, @ is displayed as Å.
In all other cases, the Forth character set is displayed

Matrix Printers

On standard US-ASCII printers all characters are printed as per the P2000 Forth character set.

On the P2123 printer:
@ becomes Å [becomes Ö] becomes Ä.

Philips Daisy Wheel Printer

With a BIL printing wheel, the entire set is printed.

With an ESA wheel, [and] are printed as (and) and @ is printed as ° (degree sign).

Version: Switzerland

Keyboard

Where possible, ASCII characters returned are shown, otherwise the hexadecimal code that is returned.

For a number of characters used in P2000 Forth, keys with different inscriptions had to be used.

98	+	"	$	00	%	&	/	(	)	=	?	00	88	
18	1	2	3	4	5	6	7	8	9	0	'	00	08	
1F	Q	W	E	R	T	Z	U	I	O	P	00	!	****	
09	q	w	e	r	t	z	u	i	o	p	@	#	*8D*	
													***	
***	A	S	D	F	G	H	J	K	L	00	00	00	0D*	
***	a	s	d	f	g	h	j	k	l	[	]	~	***	
	Y	X	C	V	B	N	M	;		00	***			
	y		c	v	b	n	m	.		-	***			
05	15	----------					A0	----------			0B	06		
01	↑	----------					20	----------			0A	02		

Screen (P2000M only)

@ is displayed as a.
In all other cases, the Forth character set is displayed.

Matrix Printers

On standard US-ASCII printers all characters are printed as per the P2000 Forth character set.

On the P2123 printer, representation and the table used depend on the version of the printer that is used:
German printers use the table at LPTAB1 and will print:
@ as Ü, [as Ö and] as Ä.
French printers use the table at LPTAB and will print:
@ as à, [as ° (degree sign) and] as § (section sign).

Philips Daisy Wheel Printer

With a BIL printing wheel, the entire set is printed.

With an ESA wheel, [and] are printed as { and } and @ is printed as ° (degree sign).

FORTH MESSAGES

For tape based systems, messages are stored in RAM in order to prevent multiple tape movements when the Forth system wishes to send a message.

Users of a disk based system should edit the following messages to Screens 4 and 5 of one or more of their disks. When LISTing them, these screens should look like:

```
Scr # 4
  0
  1 Empty Stack
  2 Dictionary Full
  3 Incorrect Address Mode
  4 isn't unique
  5
  6 Disk range?
  7 Full Stack
  8 I/O Error
  9 Displacement Error
 10 Forward Reference Overflow
 11
 12
 13
 14
 15 P2000 Forth Rel. 1.0

Scr # 5
  0
  1 In Definition Only
  2 Execution Only
  3 Conditionals not paired
  4 Definition not finished
  5 In protected dictionary
  6 Use only when loading
  7 Off current editing screen
  8
  9
 10
 11
 12
 13
 14
 15
Ok
```

Disk users can add more messages to those listed above

ALPHABETICAL INDEX

The numbers refer to the pages of this Guide.

[Ed] refers to the Editor. [Asm] refers to the Assembler.

SUPPORT AND UPDATES:

This product will be supported by Frans van der Markt and Jan Vermeulen.

Known owners of a P2000 Forth System will be given notice whenever a new Release will be available.

Questions and notifications about bugs are requested in writing and should be accompanied by as much evidence as possible (e.g. program listings and screen dumps) to:

Jan Vermeulen
Rioumstraat 55
1521 SC WORMERVEER (Netherlands)

```
 0 ( P2000 FORTH  Utility  Package  for  Release 1.2       dec 1983)
 1 ( P2000 FORTH  Utility  Package  for  Release 1.2       dec 1983)
 2 ( Make Runfile "E": Dump,Tools,Editor                JV-1983oct16)
 3 ( Dump Routine -1-                                     JV 82apr12 )
 4 ( Dump Routine -2-                                     JV 83jul01 )
 5 ( Tools 1:  Stack,Base,Vocabulary -   Kim Harris, FD III/1 p 10)
 6 ( Tools 2:  Number Output -           Kim Harris, FD III/1 p 10)
 7 ( Tools 3:  .VOCS   .BUFS                              - JV/83sep11)
 8 ( Install P2000 fig-Editor                JV-83apr18  &  WFR-79mar23)
 9 ( P2000 fig-Editor  -1-       JV/FLM - 83 apr 17  &  WFR - 1979)
10 ( P2000 fig-Editor  -2-                        WFR - 1979 may 03)
11 ( P2000 fig-Editor  -3-                       FvdM - 1983 may 25)
12 ( P2000 fig-Editor  -4-                  JV-83jun01/WFR-1979may03)
13 ( P2000 fig-Editor  -5-  Low level MATCH       JV  -  83 apr 17)
14 ( P2000 fig-Editor  -6-                        WFR - 1979 mar 24)
15 ( P2000 fig-Editor  -7-                         WFR - 79 mar 24)
16 ( Make Runfile "A": Dump,Tools,Editor,Extensions & Assembler   )
17 ( Install P2000 fig-Assembler                          JV 83 apr 18)
18 ( fig-FORTH Z80 Assembler -1-          JV-82nov21 & JJC-81aug17)
19 ( fig-FORTH Z80 Assembler -2-   Forward references   JV-83jun30)
20 ( fig-FORTH Z80 Assembler -3-   Forward References   JV-83jun30)
21 ( fig-FORTH Z80 Assembler -4-          JV-82nov21 & JJC 81aug17)
22 ( fig-FORTH Z80 Assembler -5-  Mnem.    JV-82nov21 & JJC 81aug17)
23 ( fig-FORTH Z80 Assembler -6-  Mnem.    JV-82nov21 & JJC 81aug17)
24 ( fig-FORTH Z80 Assembler -7-  Mnem.    JV-82nov21 & JJC 81aug17)
25 ( fig-FORTH Z80 Assembler -8-  Mnem.    JV-82nov21 & JJC 81aug17)
26 ( fig-FORTH Z80 Assembler -9-  Mnem.    JV-83jun30 & JJC 81aug17)
27 ( Editor extensions  -1-  Comma'ble Code A10 .        JV-83mar20 )
28 ( Editor extensions  -2-  Comma'ble Code A20          JV-83mar20 )
29 ( Trace Colon Words -            Paul van der Eijk-FD III/2,p.58)
30 ( Low-Level MATCH for the Editor  -1-                  JV-82dec31)
31 ( Low-Level MATCH for the Editor  -2-                  JV-82dec31)
32 ( Low-Level MATCH for the Editor  -3-                  JV-82dec31)
33 ( String Handling Routines                            JV-83may27 )
34 ( Screen output routines                  83jul27 F.van der Markt )
35 ( Packman -1- 83apr08 F.van der Markt )   CR ." Packman "  6 .
36 ( Packman -2- 83apr08 F.van der Markt )        5 .
37 ( Packman -3- 83may12 F.van der Markt )        4 .
38 ( Packman -4- 83may12 F.van der Markt )        3 .
39 ( Packman -5- 83may12 F.van der Markt )        2 .
40 ( Packman -6- 83may12 F.van der Markt )        1 .
41 ( Packman -7- 83may12 F.van der Markt )        0 .
```

```
Scr # 0
  0 ( P2000 FORTH  Utility  Package  for  Release 1.2      dec 1983)
  1
  2 Screens    Contents
  3
  4 00 - 01    Directory
  5
  6 02 - 02    Make Runfile "E" with scr#:
  7            03-07, 09-15, 29
  8 03 - 04    Dump Routine
  9 05 - 07    Software Development Tools
 10 08 - 08    Install Editor
 11 09 - 15    fig-Forth Editor
 12
 13 16 - 16    Make Runfile "A" with scr#:
 14            03-07, 09-15, 18-29 ( 32K min.
 15 -->


Scr # 1
  0 ( P2000 FORTH  Utility  Package  for  Release 1.2      dec 1983)
  1
  2 Screens    Contents
  3
  4 17 - 17    Install Assembler
  5 18 - 26    Z80 Forth Assembler
  6 27 - 28    Editor Extensions
  7 29 - 29    Trace Utility
  8
  9 30 - 32    Assembler Demo: Low Level Match
 10 33 - 33    String Handling Routines
 11
 12 34 - 34    Screen output routines
 13 35 - 41    Pacman  (Loads Scr #34)
 14
 15 ;S


Scr # 2
  0 ( Make Runfile "E": Dump,Tools,Editor              JV-1983oct16)
  1
  2  3  LOAD  ( Dump)     5  LOAD  ( Tools)
  3  9  LOAD  ( Editor)  29  LOAD  ( Trace)
  4
  5 ( Install loaded screens: )
  6
  7 FORTH DEFINITIONS DECIMAL
  8 LATEST        12  +ORIGIN  !   ( Top NFA)
  9 HERE          28  +ORIGIN  !   ( Fence)
 10 HERE          30  +ORIGIN  !   ( DP)
 11 ' EDITOR 6 + 32  +ORIGIN  !   ( VOC-LINK)
 12 HERE FENCE !
 13
 14 MON    ( Now save to a runfile)
 15 ;S

P2000 Forth Rel. 1.2
```

```
Scr # 3
  0 ( Dump Routine -1-                                        JV 82apr12 )
  1
  2 FORTH  DEFINITIONS  HEX
  3
  4 : BYTE
  5    HEX  0  <# BL HOLD  # # #> TYPE  ;
  6 : ADDRESS ( n --- )
  7    HEX  0 <# BL HOLD # # # # #> TYPE   ;
  8 : CHAR  ( n --- )  7F AND ( Reset bit 7)
  9    DUP  BL - 0<  IF  DROP  2E  ( a dot )
 10                  ELSE  7A OVER -
 11                     0<  IF  DROP  2E THEN
 12                  THEN  EMIT  ;
 13 : BYTES  ( addr  n ---- )
 14    0 DO DUP I + C@ BYTE LOOP  DROP  ;          ( hex output)
 15 -->

Scr # 4
  0 ( Dump Routine -2-                                        JV 83jul01 )
  1 : CHARS  ( addr  n  ---- )
  2    0 DO DUP I + C@ CHAR LOOP  DROP  ;          ( ascii output)
  3 6013 C@ 1 AND 1+ 8 * VARIABLE LLEN
  4 : DUMP ( from  count  ---- )
  5   CR  16  EMIT
  6   BASE  @  >R  OVER  -1  +  +  >R
  7   LLEN @  /MOD  SWAP  0= 0=  OVER  0<  AND  -  LLEN @  *
  8   BEGIN  DUP  ADDRESS                              ( ---- addr addr' )
  9          DUP  LLEN @  BYTES  DUP  LLEN @  CHARS
 10          LLEN @ 8 = IF CR THEN    ( New line if count is only 8 )
 11          LLEN @ +  R  OVER  -          ( Check for last address )
 12          0<  ?TERMINAL  OR       ( End reached or STOP pressed?! )
 13   UNTIL                             ( If so, leave the structure )
 14   CR 15 EMIT   DROP  R>  DROP  R>  BASE  !  ;
 15 DECIMAL    ;S

Scr # 5
  0 ( Tools 1:  Stack,Base,Vocabulary -   Kim Harris, FD III/1 p 10)
  1
  2 : DEPTH           ( leaves the number of 16 bit stack cells used)
  3     SP@  S0 @  SWAP  -  2  /  ;
  4 : .S            ( print stack contents, top last; stack unchanged)
  5     DEPTH  IF  SP@  2  -  S0  @  2  -
  6                DO  I  @  -2  +LOOP
                ELSE  ." Empty "  THEN  ;
      BASE                          ( Prints current base in decimal !)
          10  :  .  DUP  DECIMAL  .  BASE  !  ;
 11   /OR                           ( prints CONTEXT Vocabulary name )
 12                        NFA  ID.  ;
 13
 14
```

```
Scr # 6
  0 ( Tools 2:  Number Output -              Kim Harris, FD III/1 p 10)
  1
  2 ( Base-specific stack-print operators )
  3 :  BASED.  <BUILDS  ,
  4            DOES> @  BASE  @  SWAP  BASE  !  SWAP  U.  BASE
  5
  6 16  BASED.  H.     ( print top of stack in hex)
  7  8  BASED.  O.     ( print in octal)
  8  2  BASED.  B.     ( print in binary)
  9
 10 ( Number bases: )
 11 : OCTAL   8  BASE  !  ;
 12 : BINARY  2  BASE  !  ;
 13 : ALPHA   36  BASE  !  ;  ( Uses 0..9 and A..Z)
 14
 15 -->


Scr # 7
  0 ( Tools 3:  .VOCS   .BUFS                              - JV/83sep11)
  1
  2 : .VOCS   ( List vocabularies )
  3     VOC-LINK  @
  4     BEGIN   CR  DUP  11  -  -1  TRAVERSE
  5             COUNT  31  AND  TYPE  @  DUP  0=
  6     UNTIL   CR  DROP  ;
  7 : .BUFS  ( List buffers)
  8     CR  BASE  @  FIRST  ." Addr Screen "   CR
  9     BEGIN  DUP  @  OVER  HEX  U.  DECIMAL
 10            32767  OVER  AND  32767  OVER  =
 11              IF  DROP  6  SPACES  ELSE  6  .R  THEN
 12            32768  AND  IF  ." Upd"  THEN
 13            CR  1028  +  DUP  LIMIT  =
 14     UNTIL  DROP  BASE  !  ;
 15 ;S


Scr # 8
  0 ( Install F2000 fig-Editor            JV-83apr18  &  WFR-79mar23)
  1
  2 BLK  @  1+  LOAD  ( Loads Editor from following 7 screens )
  3
  4 ( Following commands install Editor in protected Dictionary;
  5   it may be then saved to a Runfile with MON)
  6
  7 FORTH DEFINITIONS DECIMAL
  8 LATEST        12  +ORIGIN  !   ( Top NFA)
  9 HERE          28  +ORIGIN  !   ( Fence)
 10 HERE          30  +ORIGIN  !   ( DP)
 11 ' EDITOR 6 + 32  +ORIGIN  !   ( VOC-LINK)
 12 HERE FENCE !
 13
 14 EDITOR   CR  ." EDITOR Resident and Current "  CR
 15 ;S

F2000 Forth Rel. 1.2
```

```
Scr # 7
  0 ( P2000 fig-Editor  -1-          JV/FLM - 83 apr 17  &  WFR - 1979)
  1
  2 FORTH DEFINITIONS HEX                  ( Text support)
  3 : TEXT                                 ( Accept following text to Pad)
  4       HERE  C/L  1+  BLANKS  WORD  HERE  PAD  C/L  1+  CMOVE  ;
  5 : LINE                      ( Relative to SCR, Leave address of line)
  6       DUP  FFF0  AND  17  ?ERROR                ( Keep on this screen.)
  7       SCR @  (LINE)  DROP ;
  8
  9 VOCABULARY  EDITOR  IMMEDIATE  HEX
 10
 11 : WHERE                                ( Print Screen # and image of error)
 12       DUP   DUP   SCR   !   ." Scr # " DECIMAL .
 13       SWAP  C/L  /MOD  C/L  *  ROT  BLOCK  +  CR  C/L  TYPE
 14       CR HERE  C@ - SPACES 5E EMIT  ACOMPILED  EDITOR QUIT ;
 15 -->
```

```
Scr # 10
  0 ( P2000 fig-Editor  -2-                                WFR - 1979 may 03)
  1 EDITOR DEFINITIONS
  2 : #LOCATE   ( ---- cursor.offset  line )
  3       R# @ C/L /MOD ;
  4 : #LEAD     ( ---- line.addr  offset.to.cursor )
  5       #LOCATE LINE SWAP ;
  6 : #LAG      ( ---- cursor.addr  count.after.cursor)
  7       #LEAD DUP >R + C/L R>  - ;
  8 : -MOVE     ( Move in block buffer:  addr.from  line.to  ---- )
  9       LINE C/L CMOVE UPDATE ;
 10 : H         ( n  ---- ;                    Hold numbered line at Pad )
 11   LINE PAD 1+ C/L DUP PAD C! CMOVE ;
 12 : E   LINE C/L BLANKS UPDATE ; ( n --- ; Erase line with blanks)
 13 : S         ( n ---- ;          Spread at line, making line blank)
 14       DUP 1 - 0E  DO I LINE I 1+ -MOVE -1 +LOOP E ;
 15 -->
```

```
Scr # 11
  0 ( P2000 fig-Editor  -3-                                FvdM - 1983 may 25)
  1
  2 : D                                    ( Delete line but hold it in Pad
  3     DUP  H  ( hold in Pad)
  4     BEGIN  DUP  0F  <
  5     WHILE  DUP  1+  DUP  LINE  ROT  -MOVE
  6     REPEAT  E  ;  ( Erase line 15)
  7
  8 : DELETE  ( count ---- )
  9     DUP  #LEAD  ROT  OVER  MIN  >R  ( No move to previous line)
 10     +  #LAG  SWAP  R  -  SWAP  CMOVE  ( Move and overwrite )
 11     #LAG  +  R  -  R  BLANKS  ( Fill to end of line)
 12     DUP  R>  -  #LEAD   DROP  OVER
 13     -  SWAP  BLANKS      ( Delete any chars on previous line)
 14     MINUS  R#  +!  UPDATE  ;   ( Back up cursor )
 15 --

 ok forth Rel. 1.2
```

```
Scr # 12
  0 ( P2000 fig-Editor  -4-                    JV-83jun01/WFR-1979may03)
  1
  2 : M              ( Move cursor by signed amount & print its line)
  3       R#  +!  CR  SPACE   #LEAD  TYPE  5E  EMIT
  4                           #LAG  TYPE  #LOCATE  .  DROP  ;
  5 : T   DUP  C/L  *  R#  !  H  0  M  ;                ( Type the line)
  6 : L   SCR  @  LIST  0  M  ;                         ( Relist screen)
  7 : R   PAD  1+  SWAP  -MOVE   ;  ( Replace line by text from Pad)
  8 : P   1  TEXT  R  ;                  ( Put following text on line)
  9 : I   DUP  S  R  ;                 ( Insert text from Pad at line)
 10 : TOP  0  R#  !  ;          ( Place cursor to beginning of screen )
 11 : CLEAR  SCR !  10 0 DO FORTH I EDITOR E LOOP ;    ( Clear screen)
 12 : COPY ( from.scr#  to.scr#  ----- ;          Copy screen contents )
 13       OFFSET  @  +  SWAP  BLOCK  2  -  !  UPDATE      ;
 14 -->
 15


Scr # 13
  0 ( P2000 fig-Editor  -5-  Low level MATCH        JV  -  83 apr 17
  1 ( Replaces high-level words MATCH and -TEXT )
  2 CREATE  MATCH         ( Comma'd in to avoid loading Assembler)
  3 ( Set up registers for text search, saving IP in BC')
  4  D1 C, D5 C, D9 C, C1 C, E1 C, E5 C, 9 C, 36 C, 0 C,
  5  E1 C, D9 C, E1 C, D9 C, D1 C, EB C, E5 C, D9 C, E5 C, B7 C,
  6  ED C, 52 C, E5 C, D9 C, C1 C, 3 C, 3B C, 23 C,
  7 ( Outer search loop: )  1A C, ED C, B1 C, 20 C, 1E C,
  8 ( Compare strings:) E5 C, D5 C, 13 C, 1A C, B7 C, 28 C, A C,
  9  BE C, 20 C, 3 C, 23 C, 18 C, F5 C, D1 C, E1 C, 18 C, EA C,
 10 ( Match:)
 11  F1 C, F1 C, F1 C, D1 C, B7 C, ED C, 52 C, E5 C, D9 C, D1 C,
 12  37 C, 18 C, 4 C, ( No match:) D9 C, D1 C, F1 C, B7 C,
 13 ( Exit:) 21 C, 00 , ED C,  6A C, EB C,
 14 ( Jump to DPUSH:) C3 C,  ' ;S  0B + @ 2 -  ,    SMUDGE
 15 -->


Scr #. 14
  0 ( P2000 fig-Editor, -6-                          WFR - 1979 mar 24)
  1 : 1LINE  ( Scan cursor line for match to Pad text and update   )
  2     #LAG PAD COUNT MATCH R# +! ;      (   cursor, return Boolean)
  3 : FIND   ( String at Pad over full Screen Range; else Error)
  4     BEGIN  3FF R# @ <
  5         IF  TOP  PAD  HERE  C/L  1+  CMOVE  0  ERROR  THEN
  6         1LINE UNTIL ;
  7 : N   FIND 0 M ;          ( Find next occurrence of previous text)
  8 : F  1 TEXT  N ;          ( Find 1st occurrence of following text)
  9 : B  PAD C@ MINUS M ;     ( Backup cursor by text in Pad)
 10 : X                       ( Delete following text)
 11     1  TEXT  FIND  PAD  C@  DELETE  0  M  ;
 12 : TILL          ( Delete on cursor line, from cursor to text end)
 13     #LEAD  +  1  TEXT  1LINE  0=  0  ?ERROR
 14     #LEAD  +  SWAP  -  DELETE  0  M  ;
 15 -->

P2000 Forth Rel. 1.2
```

```
Scr # 15
  0 ( P2000 fig-Editor  -7-                              WFR - 79 mar 24
  1
  2 : C,                ( Spread at cursor and copy in following text)
  3     1  TEXT  PAD  COUNT
  4     #LAG  ROT  OVER  MIN  >R
  5     FORTH  R  R#  +!              ( Bump cursor)
  6     R  -  >R                      ( Characters to save)
  7     DUP  HERE  R  CMOVE           ( From old cursor to HERE)
  8     HERE  #LEAD  +  R>  CMOVE     ( HERE to cursor location)
  9     R>  CMOVE  UPDATE             ( Pad to old cursor)
 10     0  M  :                       ( Look at new line)
 11
 12 FORTH DEFINITIONS DECIMAL
 13
 14 :S
 15


Scr # 16
  0 ( Make Runfile "A": Dump,Tools,Editor,Extensions & Assembler    )
  1 ( You will need a 32K Machine for this!!)
  2  3  LOAD  ( Dump)      5  LOAD  ( Tools)
  3  9  LOAD  ( Editor)   18  LOAD  ( Assembler)
  4 27  LOAD  ( Ed.Ext)   29  LOAD  ( Trace)
  5 ( Install loaded screens: )
  6
  7 FORTH DEFINITIONS DECIMAL
  8 LATEST        12  +ORIGIN  !    ( Top NFA)
  9 HERE          28  +ORIGIN  !    ( Fence)
 10 HERE          30  +ORIGIN  !    ( DP)
 11 ' ASSEMBLER 6 + 32  +ORIGIN  !    ( VOC-LINK)
 12 HERE FENCE !
 13
 14 MON    ( Now save to a runfile)
 15 :S


Scr # 17
  0 ( Install P2000 fig-Assembler                    JV 83 apr 18)
  1
  2 BLK  @  14  LOAD  ( Loads Assembler from following 9 screens )
  3
  4 ( Following commands install Assembler in protected Dictionary;
  5   it may be then saved to a Runfile with MON)
  6
  7 FORTH DEFINITIONS DECIMAL
  8
  9 LATEST          12  +ORIGIN  !    ( Top NFA)
 10 HERE            28  +ORIGIN  !    ( Fence)
 11 HERE            30  +ORIGIN  !    ( DP)
 12 ' ASSEMBLER 6 + 32  +ORIGIN  !    ( VOC-LINK)
 13 HERE FENCE !
 14 CR  ." ASSEMBLER Installed "   CR
 15 ;S

  ok Forth Rel. 1.2
```

```
Scr # 18
  0 ( fig-FORTH Z80 Assembler -1-          JV-82nov21 & JJC-81aug17)
  1
  2 HEX VOCABULARY ASSEMBLER IMMEDIATE
  3
  4 : 2*    DUP  +  ;
  5 : 8*    2*  2*  2*  ;
  6
  7 ' ASSEMBLER CFA ' ;CODE 8 + !          ( Patch ;CODE in nucleus)
  8
  9 : CODE  ?EXEC  CREATE  ACOMPILEU  ASSEMBLER  !CSP  :  IMMEDIATE
 10 : C;    CURRENT @ !  CONTEXT  !
 11         ?EXEC ?CSP  SMUDGE  ;  IMMEDIATE
 12 : LABEL  ?EXEC  0  VARIABLE  SMUDGE  -2  ALLOT
 13         ACOMPILEU ASSEMBLER  !CSP  ;  IMMEDIATE
 14 -->     [        ]
 15
```

```
Scr # 19
  0 ( fig-FORTH Z80 Assembler -2-   Forward references   JV-83jun30)
  1 3  CONSTANT  #REF            ( No. of forward references allowed )
  2 : ?RELAT                     ( Test for displacement limits)
  3     DUP  ABS  OVER    0<  7F  +  >  9  ?ERROR  ;
  4 : RESOLVE  ( addr  pointer  ----     ; Resolve forward reference)
  5     2*  +  @  1+  DUP                  ( Get reference location)
  6     HERE  ( CPU's PC)  1 -  SWAP  -  ?RELAT       ( Check offset)
  7     SWAP  C!  ;                  ( Store offset at ref. location)
  8 ( Relative Label address declaration:)
  9 : RLABEL  ACOMPILEU  '  2+  DUP  @  ( pointer)  1 -
 10     IF ( not 1) DUP  @  1  DO  DUP  I  RESOLVE  LOOP  THEN
 11     0  OVER  !  2+  HERE  SWAP  !  ;
 12 : FWDREF  ( addr ---- ) DUP  @  ( item#)  #REF 1+  =  0A  ?ERROR
 13       DUP  @  2*  OVER  +  ( reference location)
 14       HERE  SWAP  !  1  SWAP  +!  0  ;  ( Leave a zero )
 15 -->
```

```
Scr # 20
  0 ( fig-FORTH Z80 Assembler -3-   Forward References   JV-83jun30)
  1 ( Resolve a backward relative label reference,leave offset )
  2 : BKWREF  2+  @  HERE  2+  -  ?RELAT  ;
  3
  4 ( Label declaration: already declared labels are cleared and can
  5   be used again)
  6 : (RLABEL:)
  7       <BUILDS   1  ,  ( item pointer)  #REF  2  *   ALLOT
  8       DOES>  DUP  @  ( no of items)
  9                 IF    FWDREF              ( make an entry)
 10                 ELSE  BKWREF THEN  ;      ( resolve )
 11 : RLABEL:  IN  @  -FIND
 12       IF ( existing) DROP  1  SWAP  2+  !  ( reset pointer)
 13       ELSE  IN  !  (RLABEL:)  THEN  ;  ( dictionary entry)
 14 -->
 15
```

```
Scr # 21
  0 ( fig-FORTH Z80 Assembler -4-                JV-82nov21 & JJC 81aug17)
  1
  2 ASSEMBLER DEFINITIONS                        ( Constants and Variables)
  3
  4  7 CONSTANT A    0 CONSTANT BC  3 CONSTANT E    5 CONSTANT L
  5  6 CONSTANT AF   1 CONSTANT C   6 CONSTANT F    6 CONSTANT SP
  6 1D CONSTANT AF' 2 CONSTANT D   4 CONSTANT H    6 CONSTANT (HL)
  7  0 CONSTANT B    2 CONSTANT DE  4 CONSTANT HL  FA CONSTANT (SP)
  8
  9 ' ;S 08 + @ CONSTANT (NEXT)          ( Inner Interpreter Address)
 10
 11 0 VARIABLE &XYU                      ( Index Register Pair Handlers)
 12 : XY?   &XYU @ IF  C,  0  &XYU  !  THEN ;
 13 : IX    DD  C,  HL ;      : (IX    DD C,  1 &XYU !  (HL)  ;
 14 : IY    FD  C,  HL ;      : (IY    FD C,  1 &XYU !  (HL)  ;
 15 -->

Scr # 22
  0 ( fig-FORTH Z80 Assembler -5-  Mnem.   JV-82nov21 & JJC 81aug17)
  1
  2 : MI1   <BUILDS  C,   DOES>  C@ C, ;
  3 00 MI1 NOP   07 MI1 RLCA  0F MI1 RRCA  17 MI1 RLA
  4 1F MI1 RRA   27 MI1 DAA   2F MI1 CPL   37 MI1 SCF
  5 3F MI1 CCF   76 MI1 HALT  C9 MI1 RET   E9 MI1 JPHL
  6 F3 MI1 DI    FB MI1 EI    D9 MI1 EXX
  7
  8 : MI2   <BUILDS  C,   DOES>  C@ + C, XY? ;
  9 80 MI2 ADD   88 MI2 ADC   90 MI2 SUB   98 MI2 SBC
 10 A0 MI2 AND   A8 MI2 XOR   B0 MI2 OR    B8 MI2 CP
 11
 12 : MI3   <BUILDS  C,   DOES>  C@ SWAP 8* + C, XY? ;
 13 02 MI3 STAX  03 MI3 INCP  04 MI3 INC   05 MI3 DEC
 14 0A MI3 LDAX  0B MI3 DECP  C1 MI3 POP   C5 MI3 PUSH
 15 --

Scr # 23
  0 ( fig-FORTH Z80 Assembler -6-  Mnem.   JV-82nov21 & JJC 81aug17)
  1
  2 : MI4   <BUILDS  C,   DOES>  C@ C, C, ;
  3 C6 MI4 ADDA  CE MI4 ADCA
  4 D3 MI4 OUT   D6 MI4 SUBA  DB MI4 IN    ODE MI4 SBCA
  5 E6 MI4 ANDA  EE MI4 XORA  F6 MI4 ORA    FE MI4 CPA
  6 10 MI4 DJNZ  18 MI4 JR    20 MI4 JRNZ   28 MI4 JRZ
  7 30 MI4 JRNC  38 MI4 JRC
  8
  9 : MI5   <BUILDS  C,   DOES>  C@ C, , ;
 10 32 MI5 STA   3A MI5 LDA   CD MI5 CALL  C3 MI5 JP
 11
 12 : MI7   <BUILDS  C,   DOES>  ED C, C@ C, ;
 13 44 MI7 NEG   45 MI7 RETN  4D MI7 RETI  67 MI7 RRD
 14 A0 MI7 LDI   A1 MI7 CPI   A2 MI7 INI   A3 MI7 OUTI
 15 --

   ok Forth Rel  1.2
```

```
Scr # 24
  0 ( fig-FORTH Z80 Assembler -7-  Mnem.   JV-82nov21 & JJC 81aug17
  1
  2 A8 MI7 LDD   A9 MI7 CPD   AA MI7 IND   AB MI7 OUTD
  3 B0 MI7 LDIR  B1 MI7 CPIR  B2 MI7 INIR  B3 MI7 OTIR
  4 B8 MI7 LDDR  B9 MI7 CPDR  BA MI7 INDR  BB MI7 OTDR
  5 6F MI7 RLD
  6
  7 : MCB0  + &XYU $ IF SWAP C, 0 &XYU ! THEN C, ;
  8
  9 : MCB1  <BUILDS  C,   DOES>  CB C, C$ MCB0 ;
 10 00 MCB1 RLC  08 MCB1 RRC  10 MCB1 RL   18 MCB1 RR
 11 20 MCB1 SLA  28 MCB1 SRA  38 MCB1 SRL
 12
 13 : MCB2  <BUILDS  C,   DOES>  CB C, C$ SWAP 8* + MCB0 ;
 14 40 MCB2 BIT  60 MCB2 RES  C0 MCB2 SET
 15 -->

Scr # 25
  0 ( fig-FORTH Z80 Assembler -8-  Mnem.   JV-82nov21 & JJC 81aug17
  1 : ADDP   ( rp2  rp1  ----)  5  +  SWAP  8*  +  C,  ;
  2
  3 : DOPM   DUP 4 =  IF DROP 22  ELSE 8* 43 + ED C.  THEN
  4 : LDPM   ( adr  rp  ----)  DOPM  8  +  C,  .  ;
  5 : STPM   ( adr  rp  ----)  DOPM  C,  ,  ;
  6
  7 : DOED   ED  C,  SWAP  8*  +  C,  ;
  8 : SBCP   ( rp2  rp1  ----)  +  22  DOED ;
  9 : ADCP   ( rp1  rp2  ----)  +  2A  DOED  ;
 10 : CIN    ( reg  ACU  ----)   40  DOED  ;
 11 : COUT   ( ACU  reg  ----)   41  DOED  ;
 12
 13 : IM     ED  C,  8*  DUP  IF  8  +  THEN  46  +  C,
 14 : EX     +  E5  +  C,  ;
 15 -->


Scr # 26
  0 ( fig-FORTH Z80 Assembler -9-  Mnem.   JV-83jun30 & JJC 81aug17)
  1
  2 : RST    ( n  ----)  C7  +  C,  ;
  3 : JPIX   DD  C,  JPHL  ;
  4 : JPIY   FD  C,  JPHL  ;
  5 : LDRI   ( byte  reg  ----)   8*  6  +  C,  XY?  C,  ;
  6 : LDPI   ( word  rp  ----)    8*  1+  C,  .  ;
  7 : LD     ( src  dest  ----)   8*  40  +  +  C,  XY?  ;
  8
  9 ( Macros for Compiling Inner Interpreter Return Addresses)
 10 : NXT    C3  C,  (NEXT)  ;
 11 : PUSH1  NXT  1 -  ,  ;
 12 : PUSH2  NXT  2 -  ,  ;
 13 : NEXT   NXT  ,  ;
 14
 15 DECIMAL   ;S

P2000 Forth Rel. 1.2
```

```
Scr # 27
  0 ( Editor extensions  -1-  Comma'ble Code A1Ü          JV-83mar20 )
  1 ( Copy a tested machine language routine into a text screen
  2   for later C, loading without resident Assembler.
  3   Code must be relocatable.
  4   PREV must point to first screen to use and R# must point to
  5   first line to use. Following screens may be overwritten)
  6
  7 EDITOR DEFINITIONS HEX
  8
  9  : ?FIT  ( n  ----  n  ; check whether next entry fits on cur- )
 10     DUP R# § 40 /MOD SWAP ROT + 40 SWAP - 0<       (   rent line )
 11     IF ( line overflow) 1+ DUP 10 - 0=             (  and screen )
 12        IF ( screen overflow)  DROP UPDATE SCR § 1+ CLEAR 40
 13        ELSE  40 *  THEN
 14        R# !
 15     ELSE DROP THEN ;    -->

Scr # 28
  0 ( Editor extensions  -2-  Comma'ble Code A2Ü          JV-83mar20 )
  1
  2 ( Build one  'byte C,'  entry, taking input from addr1)
  3 : A,  ( addr1 ----)
  4     <#  2C  HOLD  43  HOLD  BL  HOLD
  5         C§  0  #S  BL  HOLD  #>   ?FIT
  6     PREV  §  2+  R#  §  +  SWAP
  7     DUP  R#  +!  CMOVE  ;
  8
  9 : MEM,  ( from count ----)
 10     0  DO  DUP  A,  1+  LOOP  DROP
 11     UPDATE ;
 12
 13 DECIMAL
 14
 15 ;S

Scr # 29
  0 ( Trace Colon Words -              Paul van der Eijk-FD III/2,p.58)
  1
  2 FORTH  DEFINITIONS  DECIMAL
  3
  4 VARIABLE TFLAG              ( Controls Trace Output)
  5
  6 : (TRACE)  ( give trace output, to be inserted as first word)
  7     TFLAG  §
  8     IF CR  R  2  -  NFA  DUP  ID.
  9        C§  31  AND  32  SWAP  -  SPACES
 10        -2  4  DO  SP§  I  +  §  8  .R  -2  +LOOP  ( show stack)
 11     THEN  ;
 12          ( Redefined to insert trace word after colon )
 13     ?EXEC  'CSP  CURRENT  §  CONTEXT  !  CREATE
 14        (TRACE)  CFA  DUP  §  HERE  2  -  !  ,  0  ;  IMMEDIATE

 15
with Ed
```

```
Scr # 30
  0 ( Low-Level MATCH for the Editor  -1-                JV-82dec31)
  1 ASSEMBLER DEFINITIONS
  2 ( Abbreviations used:  cur=cursor address  byt=Bytes to EOL
  3   adr=string address   cnt=string count  IP=Interpretive ptr
  4   ptr=pointer into text being searched)
  5 ( Label declarations: )  RLABEL: $1  RLABEL: $2
  6 RLABEL: $4   RLABEL: $7  RLABEL: $9  RLABEL: $X
  7
  8 CODE MATCH ( cur  byt  adr  cnt  ----  flag  cursor.movement)
  9 ( Set up search parameters)
 10 DE POP  DE PUSH  EXX  BC POP  ( BC'=IP, BC=DE'=cnt)  HL POP
 11 HL PUSH  BC HL ADDP  0 (HL) LDRI  ( 0 at end of string)  HL POP
 12 EXX  HL POP  EXX  ( HL=adr, HL'=byt)  DE POP  DE HL EX  HL PUSH
 13 EXX  HL PUSH  ( ---- cur byt)  A OR  DE HL SBCP  HL PUSH  EXX
 14 BC POP ( Search counter)  BC INCP  ( Limit range to cnt-byt)
 15 -->


Scr # 31
  0 ( Low-Level MATCH for the Editor  -2-                JV-82dec31
  1
  2 ( Search start: BC=search range, DE=adr, HL=ptr, BC'=IP)
  3 $9  JRC  ( If range<0)
  4
  5 RLABEL $1  ( Find next occurrence of 1st character)
  6   DE LDAX  CPIR  ( HL=ptr, BC=counter)  $9  JRNZ
  7   HL PUSH  DE PUSH  ( Save ptr, adr)
  8
  9 RLABEL $2  ( Compare remainder of argument with text)
 10   DE INCP  DE LDAX  A OR ( zero?)  $7  JRZ
 11   (HL) CP  $4  JRNZ  ( no match)  HL INCP  $2  JR
 12
 13 RLABEL $4  ( Try again from next character onward)
 14   DE POP ( adr)  HL POP ( ptr)  $1  JR
 15 -->


Scr # 32
  0 ( Low-Level MATCH for the Editor  -3-                JV-82dec31)
  1
  2 RLABEL $7  ( Exit if match is found)
  3   AF POP  AF POP  AF POP ( Drop adr,ptr,byt) DE POP ( cur
  4   A OR  DE HL SBCP  HL PUSH  EXX  DE POP  SCF  $X JR
  5
  6 RLABEL $9  ( Exit if no match found)
  7   EXX  DE POP ( byt) AF POP ( cur)  A OR  ( carry=0)
  8
  9 RLABEL $X  0 HL LDRI  HL HL ADCP  HL DE EX  PUSH2
 10
 11 C;
 12
 13 DECIMAL
 14 :S
 15

(c) Forth Rel. 1.2
```

```
Scr # 33
 0 \ String Handling Routines                                  JV-83may27 )
 1 FORTH  DEFINITIONS  DECIMAL
 2 ( String declaration: length  STRING  name ; filled with blanks)
 3 : STRING  ( len ---- )
 4     <BUILDS  DUP  C,  HERE  OVER  BLANKS  ALLOT
 5     DOES>  COUNT ( ---- addr len )  ;
 6 ( Direct load:  STRING: name  "ccccc" ; maximum 255 bytes)
 7 : STRING:
 8     <BUILDS  -1  IN  +!  34  ( ")  WORD  34  WORD
 9         HERE  C@  1+  ALLOT
10     DOES>  COUNT  ( ---- addr  len )  ;
11 : FILL$  ( list.on.stack  addr  count  ---- )
12     SWAP  1  -  SWAP  OVER  +  DO  ?STACK  I  C!  -1  +LOOP ;
13 :S      Use FILL$ for graphics and control codes:
14         E.g.:  2 STRING CRLF$  13 10 CRLF FILL$
15         CRLF$ LPT will then send a CR and a LF to the printer


    screen output routines                    83jul27 F.van der Markt )

  PUT                          dest  from  count          endaddress
     ROT  OVER  OVER  +  >R  SWAP  CMOVE  R>

   (PUT")     R>  COUNT  DUP  1+  R>  +  >R  PUT  ;
   PUT"              ( put inline string:   dest  ----  new-address)
     22  STATE  @
     IF  COMPILE  (PUT")  WORD  HERE  C@  1+  ALLOT
     ELSE  WORD  HERE  COUNT  PUT  ENDIF  ;  IMMEDIATE
   PUT#              ( number output:   dest  n  ---- new-address )
         #  BL  HOLD  #S  #>  PUT  ;
: PUTC ( address  code  ---- new-address )  OVER  C!  1+
: NL ( addressfrom ---- new-address )  50  /  1+  50  *  ;
: XY ( row  column  ----  address )  SWAP 50  *  +  5000 +
DECIMAL ;S
```