ABC KLUBBEN

Rapport nr 2
Juni 1982

Fig - FORTH on ABC 80

Implemented by

Robert Johnsen

ABC-klubben, Vidängsvägen 1
161 33 BROMMA
Pg 15 33 36-3

ABC KLUBBEN

ABC-klubben

är en sammanslutning av användare av ABC-80 och dess efterföljare ABC 800. ABC-klubben är en **ideell förening** som står helt fritt från kommersiella intressen. ABC-klubben har f n (april 1982) över 2300 medlemmar spridda över hela landet. Klubbens syfte är att tillvarata medlemmarnas gemensamma intresse av datorer och datortillämpningar och verka för ökade kunskaper inom dessa områden till nytta utbildning och nöje. Ledstjärna för verksamheten är **erfarenhetsutbyte.** Lokalavdelningar av ABC-klubben bildas på orter där medlemsunderlag finns. För att underlätta kontakter mellan medlemmarna ger ABC-klubben ut en **Medlemsmatrikel** med namn, adress och telefonnummer till alla medlemmar.

ABC-klubben ger ut **ABC-bladet** som utkommer med 4 - 5 nummer om året. Tidningen speglar klubbens hela verksamhet och innehåller intressanta artiklar i skilda ämnen, bl a redovisning av tester av hård- och mjukvara som utförts i klubbens regi samt massor av tips. ABC-klubben bevakar den tekniska utvecklingen och verkar för att få fram en enhetlig standard och kompabilitet när det gäller program och utrustning.

ABC-klubben ger även ut **ABC-kassetter** fulla med nyttiga program gjorda av medlemmar och andra, c:a 3 kassetter per år sändes fritt till alla medlemmar i klubben. ABC-kassetterna har blivit en mycket uppskattad medlemsförmån.

ABC-klubben har på kansliet i Alvik ställt upp en program- och informationsbank **ABC Monitor** dit du kan ringa och själv hämta program och läsa meddelanden om du har tillgång till 300 baud modem. Du kan även sända program dit som andra medlemmar får hämta. Systemet är ständigt påkopplat och har varit i kontinuerlig drift i över ett år. ABC Monitor finns även på andra platser i landet.

ABC-klubben ger ut **ABC-Rapporter** som innehåller värdefull dokumentation. ABC-Rapport nr 1, den s k dissassemblern, innehåller en listning av programvaran i ABC-80 med kommentarer, en verklig guldgruva för den som vill dyka djupare.

ABC-klubben har påbörjat ett arbete som på sikt kommer ge stor utdelning, nämligen att gå igenom och ta fram kommentarer, köranvisningar och annan dokumentation över alla de program som klubben får tillgång till. Syftet är att skapa ett väldokumenterat programbiblotek med användarprogram. Arbetet med denna s k programscreening kommer att bli mycket omfattande men vi räknar med att få hjälp av intresserade medlemmar över hela landet.

Gå med i ABC-klubben för att få ökat utbyte av din dator och få tillgång till den stora erfarenhet som vi gemensamt bygger upp.

Medlem blir du enklast genom att betala in medlemsavgiften på ABC-klubbens postgirokonto nr 15 33 36-3. Medlemsavgiften är för år 1982 **125 kr för seniorer och 70 kr för juniorer (junior är man t o m det kalenderår man fyller 18).**

ABC-klubben
Vidängsvägen 1
161 33 Bromma

Telefon:
08-80 15 22 (automatisk telefonsvarare)
08-80 15 23 (Modem med ABC Monitor)

Postgiro 15 33 36-3

Medlemsavgifter 1982

Seniorer 125 Skr
Juniorer 70 Skr

Junior räknas man t o m det kalenderår man fyller 18 år. Ange därför personnummer när Du betalar medlemsavgiften. Medlemskapet är personligt.

Medlem blir Du enklast genom att sätta in medlemsavgiften på ABC-klubbens postgirokonto 15 33 36-3.

ABC-klubbens styrelse 1982

Ordförande:	Gunnar Tidner
Vice ordförande:	Kjell-Åke Johansson
Sekreterare:	Ulf Sjöstrand
Kassör:	Marianne Forsman
Redaktör:	Claes Schibler
Ledamot:	Joe Johnsson
Suppleanter:	Allan Larsson
	Björn Sjöborg
	Göran Sundqvist

INNEHÅLLSFÖRTECKNING

Anvisningar för överföring från kassett till disk

Denna version av Fig-FORTH som ABC-klubben förvärvat rättigheterna till har utvecklats av docent Robert Johnsen vid Institutionen för Fysikalisk Kemi, Uppsala universitet. Den kan köras på en standard ABC-80 utrustad med 5-tums flexskiveutrustning. Programmet är skrivet i assembler och köres under CMDINT.SYS.

Det är vår förhoppning att programvaran skall komma att utvecklas och anpassas även till andra system såsom ABC-80 med enbart kassett, 8-tums flexskiveutrustning eller ABC-800. ABC-klubben ställer gärna källkoden till förfogande till dem som vill göra sådana utvecklingar.

Av praktiska skäl distribueras promgramvaran på kassett. På ABC-kassett nr 4 ligger FORTH.ABS för ABC-80 utan extra minne. Den kan köras även på ABC-80 med 32 K RAM men därvid utnyttjas inte hela RAM-minnet effektivt. För den utbyggda versionen finns FORT32.ABS som läggs på ABC-kassett nr 5. FORTH utnyttjar inte det vanliga filhanteringssystemet på ABC-80 utan lagrar program och data på s k screens. Varje screen upptar 3 sektorer på disketten och lagras fr o m spår 3 sektor 1 och uppåt. Man bestämmer själv hur stort utrymme man vill reservera för screens. Om man reserverar plats för 75 screens så ryms hela programbibloteket, tiotalet egna screens samt FORTH.ABS och CMDINT.SYS på en skiva med enkel densitet. Programbibloteket till FORTH distribueras i form av textfilerna SCREEN.TXT och SCREEN2.TXT. SCREEN.TXT innehåller screens nr 3 till 36. SCREEN2.TXT som läggs på ABC-kassett nr 5 innehåller screens nr 34 till 64 samt en fullständigare version av screen nr 7. Screens nr 34 till 36 i SCREEN.TXT kommer att skrivas över av andra screens från SCREEN2.TXT. Det kan vara skäl att spara screen 36 genom att flytta den till annan plats med nummer högre än 64.

För att lägga upp screens på disketten i det format de skall ligga i användes programmet DOSCREEN som ligger på ABC-kassett nr 4. Programmet överför screens från textfil SCREEN.TXT resp SCREEN2.TXT till Forth-format samt skriver ett directory med filnamnet "Forthscr.een". Läser man det med vanliga LIB så kan man se hur många sektorer dessa screens tar upp. Forthscr.een är ingen vanlig fil, man kan därför inte kopiera den med COPYLIB. För att kopiera till annan diskett måste man använda något program som kopierar från spår till spår i exakt samma form som orginalet.

Så här går det till att från de leverade filerna på kassett göra en systemskiva för FORTH:

1. överför först de aktuella programmen på ABC-kassetten till diskett med hjälp av programmet CASDISK.
2. Formatera en ny diskett och sätt den i drive 0.
3. Sätt skivan med de från kassetten överförda programmen i drive 1 och gör RUN DOSCREEN.
4. Svara på frågan om det redan finns ett screen-directory med tillräcklig plats. Svara N för nej när du gör en helt ny systemskiva.
5. Svara på frågan om det högsta screen-nummer du vill reservera plats för. RETURN sätter standardvärde (default) till 75.
6. Ange filnamn SCREEN.TXT resp SCREEN2.TXT.
7. Kopiera därefter över FORTH.ABS (eller FORTH32.ABS) samt CMDINT.SYS till den nya systemskivan med hjälp av COPYLIB.
8. Finns det plats kan du lägga in även andra filer t ex LIB, COPYLIB och COPYDISK som kan vara bra att ha på systemskivan.

Om du redan har en systemskiva med tillräklig plats bör du svara J för ja i punkt 4. Frågan i punkt 5 hoppas då över och övriga programfiler kommer att vara oförändrade varför du inte behöver göra kopieringen enligt punkterna 7 och 8. Vill du utöka utrymmet för screens kommer alla gamla screens som ej skrivs över av nya att vara kvar oförändrade men du måste kopiera in övriga programfiler enligt punkt 7 och 8.

För att köra FORTH sättes systemskivan i drive 0 och du skriver BYE. Därefter ger du kommandot FORTH (eller FORTH32 om du har extra minne och vill använda den versionen. Det kan i så fall vara praktiskt att döpa om den versionen till FORTH.ABS och den gamla till FORTH16.ABS.)

FORTH är nu klar att använda. Lycka till

Gunnar Tidner

FOREWORD

The information presented on the following pages is the basic instruction manual for running FORTH on the ABC80. The GLOSSARY has been taken from the fig-FORTH INSTALLATION MANUAL, provided through the courtesy of the FORTH INTEREST GROUP, PO Box 1105, San Carlos, CA 94070. Here you will find a description of each word in the standard core vocabulary, with notes on which parameters are expected to be on the stack when the WORD (routine) is called, and which values are found on the stack on return from the routine.

An introduction to FORTH on the ABC80 is also presented. Here you will find information about additional WORDS available in the core in this ABC80 implementation, thus adding to the basic glossary. An explanation of SCREENS and how to edit them is included, with comments on the various editors available on this system. And in conclusion, some remarks are made about the FORTH assembler, contributed through the courtesy of John J. Cassady of the FORTH INTEREST GROUP.

How does one learn to write programs in FORTH? BYTE magazine devoted an entire issue to FORTH in August, 1980. Until recently, that was probably the best starting point for learning the language. Last year Leo Brodie of FORTH, Inc. published a book entitled "Starting FORTH". The book is filled with examples, it is written in a pleasant, light-hearted, often joking, style, and it presents almost everything you need to know to use and enjoy FORTH. Another source book for FORTH is the instruction manual for PET-FORTH. It is written in Swedish, and is concerned with an implementation of fig-FORTH on the Commodore PET. Most of the information applies as well to ABC80-FORTH. Some of the PET enhancements are not (yet) found on the ABC80, others are found under another name (and some ABC80 enhancements are not found on the PET). The manual is well-written, contains some good examples and many valuable comments and explanations.

And finally, the FORTH INTEREST GROUP in California produces FORTH DIMENSIONS, six issues per year. These newsletters contain articles for both beginners and experienced FORTH programmers. (A one year membership in the FORTH INTEREST GROUP, together with air mail subscription to FORTH DIMENSIONS costs 27 US dollars).

Every FORTH system should contain a WARNING:

THIS PRODUCT IS HABIT-FORMING!

You may never write another program in BASIC!

Uppsala, 1982.06.07
Bob Johnsen

FORTH on the ABC80

An implementation of fig-FORTH on the ABC80

By Robert Johnsen
Uppsala
June 6, 1982

1 Introduction

1.1 General

FORTH was created by Charles Moore in the early 70´s. It´s prominent features are compactness, speed and good structure. Since the noticeable deficiencies of microcomputers lie in their limited memory capacity and limited speed, the FORTH language is especially appreciated by microcomputer enthusiasts. The FORTH language is especially useful for instrument and process control, and games, is quite good for data-bases and is weakest in pure "number-crunching" applications.

The FORTH language consists of a collection of subroutines. Any subroutine can be called and executed by merely writing its name. In FORTH, the subroutines are called WORDS. A new WORD is constructed by starting its definition with a colon (:) followed by a space, then the name of the new WORD. We then list the WORDS called by the new definition, and end the definition with a semicolon (;). Parameters are passed to and from subroutines on the stack. (There is a separate stack for return addresses.)

1.2 poly-FORTH and fig-FORTH

There are two main "dialects" of FORTH. poly-FORTH is the product of Charles Moore´s company, FORTH, INC. This is available for the ABC80 in a PROM version.

The second "dialect" is produced by the Forth Interest Group in California, and is called fig-FORTH. This implementation is derived from fig-FORTH. The fig-FORTH editor enhancements are taken from Kim Harris´ contribution to FORTH DIMENSIONS II/6 and the poly-FORTH-type editor, from a contribution by S.H.Daniel to FORTH DIMENSIONS, III/3. The 8080 assembler was contributed by John J. Cassady (the author of the 8080 implementation of fig-FORTH).

There are very few differences between poly-FORTH and fig-FORTH. Both allow subroutine names (called WORDS) of up to 31 characters. In poly-FORTH the number of characters in the name is recorded along with the first 3 characters, while fig-FORTH records the number of characters in the name and ALL characters in the name. This means that WORDS of the same length, beginning with the same three letters, will confuse poly-FORTH, but not fig-FORTH. Compilation time will be longer for fig-FORTH, but execution times will be uneffected.

In fig-FORTH variables must be given values when defined. For example, the variable SIZE, in fig-FORTH would be defined:

```
0 VARIABLE SIZE
```

while in poly-FORTH one would define

```
VARIABLE SIZE
```

1.3 Deviations from fig-FORTH

Although FORTH does not require the use of floppy discs, it would be inconvenient to use it without discs for any but the simplest applications. The discs are divided into BLOCKS of 1024 bytes (the appropriate number of sectors are logically "blocked together"). On a conventional large screen, a BLOCK is displayed on 16 lines of 80 characters. The limited size of the ABC80 screen makes this impossible, so that a BLOCK in this implementation is defined as 768 bytes.

A BLOCK containing program text is commonly called a SCREEN. An application is commonly edited onto one or more SCREENS and then loaded when the program is to be run. In the present implementation, a SCREEN is displayed on 21 lines of 36 characters, making a total of 756 bytes used on a SCREEN. Although this arrangement was forced on us due to the small size of the ABC80 screen, it has actually proved to be an advantage. A WORD definition may be written as one long string of WORDS, just as a program may be written in PASCAL as one long string of commands. In order to make the program easier to understand, we prefer to break it up into shorter lines, indenting DO loops and IF and ELSE statements. The same approach is used in FORTH. The resulting "program" is thus "tall and slim" rather than "short and broad". In many cases you will find that a program that occupies a conventional 1024 byte screen will fit just as well on the ABC80 756 byte screen!

When text (or numbers) are entered directly from the keyboard, you may enter up to 80 characters without pressing RETURN, just as you may enter lines up to 120 characters in BASIC.

1.4 Additions to fig-FORTH

A few WORDS have been defined identically under two different names: the fig-FORTH name and the poly-FORTH name. For example, the fig. word 0= is identical to the poly. name NOT, and the fig. DP is identical to poly.´s H. Either word may be used in this implementation.

The Swedish keyboard for the ABC80 lacks some characters used in FORTH, noteably the square brackets. The three words, Ä, Å and ÄCOMPILEÅ have a strange appearance on the ABC80. These words have also been given the names LB (Left Bracket), RB (Right Bracket) and /COMPILE/.

ABC80 enhancements

In addition to the above pseudonyms which have been encorporated for the user´s convenience, many words not contained in the original core vocabulary have been added to the core.

FORTH words are generally of three kinds: constants, variables and colon definitions. When a CONSTANT is named, its value will be put on the stack. When a VARIABLE is named, its address will be put on the stack. When a COLON defined word is named, it will be executed.

CONSTANTS

XCUR returns the value 65012, which you may recognize as the address of the byte containing the column position of the cursor.

YCUR returns the value 65011 (row position of the cursor).

CLOCK returns 65008, the address of the lowest byte of the ABC80 three byte clock.

?PR returns 0 if the printer is not activated, 1 if a CENTRONICS interfaced printer is activated and 2 if a serial interfaced printer is activated.

NEXT, RXD and TXD are start addresses for machine code routines which you might want to use in assembler coded definitions (more will be said about this later).

VARIABLES

OUT is a standard fig-FORTH variable which is incremented for each character that is out-put. This can be used as a tabulator to produce an attractive layout for the computer output. A new variable has been added to this implementation, called #CR. This variable is incremented for each carriage return output, and is intended for use in formatting the page length of the output.

PR-TYPE has been added to this implementation, and contains, by default, the value 1, indicating that a CENTRONICS interfaced printer is indicated when printing is requested. This variable should be set to 2 if a serial interfaced printer is to be used.

DISK-ERROR is a variable which contains the error code returned by the disc controller when an error is encountered in disc access. This is automatically converted to an error message which will appear on the screen when an error occurs.

COLON DEFINITIONS

New WORDS are defined in FORTH with colon definitions (WORDS defined with the assembler begin with the word CODE, rather than a colon). Since parameters are passed to and from routines via the stack, it is common practice to illustrate the top-most elements on the stack prior to calling the routine, as well as on return from the routine. A colon definitions is thus appears as:

: HEJ (n-3, n-2, n-1 -- m-2, m-1)

n-1, n-2 and n-3 are parameters passed to the routine, with n-1 top-most on the stack. m-1 and m-2 are the values returned by the routine, with m-1 top-most on the stack. Although the numbering may vary from author to author, the order is always the same: the top-most stack element is to the right in the group of parameters.

VEDIT (screen# --)
is an elementary video editor. See below (under Editing) for more details.

CLEAR (screen# --)
fills a block with blanks. See Editing.

PAGE (--)
will clear the screen and home the cursor without issuing a formfeed to the printer.

BINARY and OCTAL (--)
change the calculation base to base 2 or base 8, just as the standard words DECIMAL and HEX change the base to 10 or 16.

<CMOVE (source-addr, destination-addr, number --)
is a block move similar to CMOVE, but it moves the last byte first. It is a stantard poly-FORTH word.

CURADDR (X, Y -- addr)
assumes that the X and Y screen coordinates (column and row) are on the stack, and converts these to the video memory address for that coordinate.

PR-ON (--)
informs the system that all future output should go both to the screen and to the printer.

PR-OFF (--)
negates the PR-ON, and all future output goes only to the screen.

BAUD (baud-rate --)
uses the value on the top of the stack to set the baud rate for serial output or input through the V24 contact.

ASCII-OUT (char --)
sends out the character on the top of the stack in serial form via pin 2 of the V24 contact.

ASCII-IN (-- char)
receives a character on pin 3 of the V24 contact. You can exit this routine by pressing any key.

2DROP (n-2, n-1 --)
will drop the top two stack members.

2SWAP (n-4, n-3, n-2, n-1 -- n-2, n-1, n-4, n-3)
will swap stack members 1 and 2 with stack members 3 and 4.

0> (number -- true/false)
returns TRUE (1) if the top stack member is greater than 0, otherwise, FALSE (0).

I´, J and J´ (-- number)
are standard poly-FORTH words. They do not belong to the core vocabulary in fig-FORTH, but they have been entered in the core in this implementation.

CCONSTANT and CVARIABLE (usage: 0 CCONSTANT name)
define byte constants and variables in the same way that CONSTANT and VARIABLE define cell values (2 byte values).

2 Starting up on the ABC80

Insert the program disc in drive 0 and type BYE to come into the disc operative system. Next, type FORTH. When FORTH has finished loading (it takes about 8 seconds) the screen will be cleared and the message Z80 FIG-FORTH 1.1 will appear. Replace the program disc in DR0 with the FORTH SCREENS disc and write 6 LOAD (FORTH sees the difference between capitals and small letters, so you must write LOAD and not load). You have now added some new words to your vocabulary, and you are informed that you can add some new vocabularies by writing EDIT or POLYED for editors, or ASM for an 8080 assembler.

FORTH has a very primative disc system. There is no file directory in the system. In order to review the contents of a disc, you should use the word INDEX. By writing, for example, 3 20 INDEX, you will list on the screen the first line of each of screens 3 through 20. It is a FORTH convention that each screen begins with a comment line which should indicate the contents of that screen. If you write, instead, PR-ON 3 20 INDEX PR-OFF you can make a hard copy on your printer.

3 Editing

At first, it will be fun to try the various FORTH commands directly at the keyboard. You can enter definitions directly on the keyboard, and use them until you turn off the ABC80. But you cannot list a definition, if you forgot exactly how you wrote it. It exists in the computer in compiled form, which is only a list of addresses to the routines called by the definition. You will soon find that it is much more convenient to edit the definitions onto a screen, and then load the screen. If it is a good definition, you´ll want to use it again sometime. If it´s a bad definition (it doesn´t work the way you intended) you can easily change it with the editor and try again.

VEDIT

An elementary video editor has been included in the core vocabulary. It is intended primarily for the convenience of new users who do not have the SCREENS disc available, as well as for those who choose to change the screen format. (If you have expanded the screen to 80 characters, you might like to try the conventional 64 character, 16 line screen). VEDIT should adapt itself automatically to any new format implemented in the core.

5 VEDIT will initialize editing of screen 5. The existing screen 5 is listed with line numbers, and a white border appears on the right of the screen, to show the editing limits. The cursor is moved around the screen by the use of keys left-arrow (left), Ü (up), right-arrow (right) and RETURN (down). Any printable characters entered are placed on the screen. ctrl-É (with an accent) will cause you to leave the editor.

It is best to list a screen (5 LIST, for example) before editing it. If you are working with a newly formated disc, the screen will be filled with ascii character 96. Blank the screen by typing 5 CLEAR, and list it again. Then edit it with VEDIT.

The FORTH code for VEDIT and its related routines is included on the SCREENS disc. It is constructed around a CASE statement, and was authored by Major Robert A. Selzer. It was first published in FORTH DIMENSIONS II/3. A greatly expanded version, called FEDIT, appeared in FORTH DIMENSIONS II/5 (by Edgar H. Fey), with important errata in FORTH DIMENSIONS III/5.

Fig-FORTH editor

The fig-FORTH editor is rather primative, but two commands have been added to it which make it very easy to use. Write EDIT and this editor will be loaded from the disc. Write EDITOR and you will be able to use the editor vocabulary. Find an unused screen and list it. Screen 80 is probably free, so write 80 LIST. If it is filled with rubbish, write 80 CLEAR. L will list the screen again, and show that it is now clear. Lines are numbered from 0 to 20.

NEW

Enter new text on a line by entering 0 NEW (for line 0). Write in what you want (up to 36 characters) and RETURN. You can continue directly now, entering text on line 1, or you can press RETURN to leave the NEW mode. NEW replaces existing text on a line, unless you press RETURN at the very beginning of the line.

The first line of a screen should be a comment line, which starts with (followed by a blank, then the comment text, and ended with a).

UNDER

This command will allow you to squeeze in a new line under an existing line, without replacing existing text. The last line (the old line numbered 20) will disappear forever. 0 UNDER will let you enter a new line 1: the old line 1 will be moved down to line 2, etc. After RETURN you can continue to squeeze in more lines. An immediate RETURN will cause you to leave the UNDER mode.

Ending the editing session

On the first free line after your definitions on a screen, write ;S. When the screen is later loaded, this FORTH word, ;S, will cause loading to cease. After this word, you can write anything you like on the screen, and it will not be loaded. A few explanatory comments and the date might be nice to have there, for instance. Now write FLUSH and the newly edited screen will be copied onto the disc.

The poly-FORTH editor

This editor has many useful functions not found in the fig-FORTH editor. You can Find a given string directly on a screen: you can Delete a string, or Replace a string, and even Search through the current screen and subsequent screens after all occurances of a given string. Its functions and uses are well described in Brodie´s excellent book, Starting FORTH.

4 8080 assembler

An assembler is included, which uses the 8080 mnemonics, and the typical FORTH post-fix notation. An example of its use is included on the FORTH SCREENS disc. An assembler definition of a WORD must end in a jump to the inner interpreter. The address of the inner interpreter is NEXT, which is a CONSTANT in this implementation. RXD and TXD are CONSTANTS which are equal to the start addresses of the routines for serial input and output. In an assembler definition you can put the ASCII character to be sent into register E and write TXD CALL. Or write RXD CALL and you will find the ASCII character received in the E register. AN ASSEBLER ROUTINE MUST PRESERVE THE BC REGISTER PAIR.

5 Additional comments

Disc formatting

The discs used on this system should be formatted with DOSGEN,F. A single density disc has tracks 0 to 39, each track containing sectors 0 to 7. The BLOCK, track, sector organization is as follows:

BLOCK	DRIVE	TRACK	SECTOR
1	0	3	1
			2
			3
2		3	4
			5
			6
3		3	7
		4	0
			1
			2
98	0	39	4
			5
			6
99	1	3	1
			2, etc.

A double density disc contains 80 tracks, with 8 sectors per track. Blocks 1 to 205 are laid out consecutively. The system assumes single density discs by default. A double density FORTH SCREENS disc will be read properly until you address screen 99 or greater. At that time the system will fetch the block from DR1. In order to prevent this you must enter 1 DENSITY ! before addressing a screen number greater than 98.
Notice that in order to avoid splitting a screen between two discs one sector (for single density) is unused at the end of each disc.

Serial printer connection

The cord connecting the V24 contact on the ABC80 and a serial interfaced printer should have the following connections:

```
V24                    printer
pin 2 ------------> pin 3      transmission
pin 5 <------------ pin 20     printer ready
pin 7 ------------- pin 7      signal ground
```

For serial output (to HIPLOT, for example) transmission is from pin 2 and signal ground on pin 7. No check for device ready is made in the ASCII-OUT routine.

For serial input (from HIPAD, for example) reception is on pin 3, with signal ground on pin 7. ASCII-IN waits for the start bit, then counts in 8 serial bits and pushes the value onto the stack. If something goes wrong, and no start bit arrives, you can interrupt the routine by pressing any key.

Ports may be addressed in FORTH with the words
PÉ (port# -- n)
P! (n, port# --)

Screen	Description
3	Title screen must be included
4 - 5	Error messages - reported when error occurs if WARNING is set to 1, as it is by default. (Set WARNING to 0 if a disc without error messages on blocks 4 and 5 is placed in DR0.) Line 15 of screen 4 is printed by writing 15 MESSAGE.
6 - 7	My standard starting screens. Initializes things I usually want available. If error occurs when loading a screen, write WHERE. Notice the defining word LOADED-BY. If you define a number of words using FORTH assembler, it would be a good idea to load the assembler when starting up the system. Notice also the definition of the word ASCII. It allows you to write in a program, for example, ASCII A, and to get the value 65 on the stack.
8	Print a copy of the screen on a printer. The name of the colon definition on line 4 is ascii character 127 (ctrl-≷), which is a non-printing character. Illustrates temporary storage on return stack (line 3) and temporary use of current dictionary space for storage.
9	A printing version of INDEX.
10	List screens on printer, 2 per page with message.
11	DR0->DR1 is used to produce a back-up disc of screens and DR1->DR0 has a similar function. Used in the form 10 20 DR0->DR1 to copy screens 10 to 20 from DR0 to DR1.
12 - 14	Memory dump routines. 32769 20 DUMP will dump the first 20 bytes, starting at location 32769, on the screen, 8 locations per line. 32769 20 DUMPA will also dump the first 20 bytes, but will start at 32768, which is the first lower address that is divisible by 8.
15	Loading screen for the poly-FORTH editor. This program was submitted to FORTH DIMENSIONS (III/3, page 80) by S.H. Daniel
16 - 30	poly-FORTH editor definitions. K on screen 30 is taken from BRODIE's book.)! puts) at next-to-last place on the current line (see the index listing).
31	3 38 READ.SECTOR will copy sector 3, track 38 to the 256 byte buffer starting at 62720. The address, 62720, will be left on the stack. Inspect a sector by using READ.SECTOR, followed by 256 DUMP.

32 3 38 WRITE.SECTOR will copy the 256 bytes in the buffer starting at 62720 onto sector 3 of track 38. Experiment (carefully) with the directory tracks or bit map, if you like.

33 An example of how a program may ask for an input number (interactive input).

34 The loading screen for the fig-FORTH editor.

35 - 40 The fig-FORTH editor definitions as taken directly from the Installation Manual, with a few changes appropriate to the ABC80 screen size.

41 fig-FORTH editor extensions, NEW and UNDER, as proposed by Kim Harris in FORTH DIMENSIONS II/6.

42 Loading screen for the 8080 assembler. Notice the "dummy" definition of ASM. It is a good habit to always start a session with FORTH by loading screen 6 (which continues onto screen 7). As mentioned previously, these screens contain additional conveniences which I generally want available. Among other things, the defining word, LOADED-BY is constructed there, and used to define the word ASM. When I later type ASM I cause screen 42 to be loaded. This screen re-defines ASM, and also loads the assembler. When I write a new definition, using assembler mnemonics, I always start the screen with ASM. When that screen is later loaded, it will cause the assembler to be loaded, before the new word is defined. If the assembler has already been loaded, the definition of ASM on screen 42 will prevent an additional loading of the assembler.

43 - 46 The 8080 assembler, as contributed by John Cassady of the FORTH Interest Group.

47 Examples of the use of the 8080 assembler. CODE is used to start the definition (instead of the colon) and C; ends the definition (instead of the semi-colon). The first example is the definition of CSWAP, which fills the same function as the BASIC instruction SWAP: it swaps the first and second bytes of a two-byte integer. For example, HEX 1234 CSWAP . will produce 3412 as output.

The second example is for the word LCFOLD, which converts lower case letters to upper case. Try this on the text on screen 62. 62 BLOCK 756 LCFOLD 62 LIST will display screen 62 in only capital letters. Notice the use of FORTH program structure together with assembler mnemonics.

48 Another example of the use of the assembler. The word STATUS is defined. The hex value 80 is sent to the command port with hex address F0. Then the status port F1 is read until the value received matches the numerical value that was on the stack when STATUS was called. The subroutine DELAY is never called from FORTH, only from another assembler routine. It therefore concludes with RET. STATUS, on the other hand, is called from FORTH, and is therefore concluded with C;.

49 - 50 An example of the use of ASCII-OUT to drive the plotter HIPLOT. The baud rate is 9600, which is set on screen 50. Words defining one step movement of the pen are defined on the first half of screen 49. The second half of this screen contains words for longer pen movements. 5 NS will move the pen 5 steps North, for example.

CIRCLE, on screen 50, draws an octagon of any desired size, and DRAW draws the cirle, then moves the pen to the far-side of the circle.

51 - 52 This is a direct translation from BASIC to FORTH of the algorithm to produce the best "staight" line with a displacement X, Y. The word, BESTLINE, has such a long definition, that it stretches over two screens. This is very poor FORTH style! A FORTH word should rarely be more than a few lines long.

53 Using ASCII-IN to drive the Houston digitizer, HIPAD. 15 ascii characters are accepted from HIPAD, and stored on the stack, in the word HIPAD, and they are printed out by the word PRPAD.

54 - 55 Routines for setting and reading the ABC80 clock. A temporary copy of the clock is saved, and manipulated, in the four byte variable MYCLOCK. INVERT.CLOCK corrects the default in the clock decrementing routine in the BASIC rom, and inverts the three bytes. SET.TIME sets the clock, and READ.TIME reads the clock, using READ.CLOCK to copy the clock into MYCLOCK until CHECK.CLOCK shows that the clock hadn't changed while being read.

56 There is no case statement (multiple choice branching) in FORTH. This version of CASE: is taken from BYTE, August 1980. The case numbering is from zero, sequentially upward. In the example, a branching word ANIMAL is defined. If we type 0 ANIMAL, then the first routine named after ANIMAL in line 11 should be executed. The routines' names do not have to begin with numbers, as 0PET, 1PET, 2PET: what is important is the order in which they follow the branching word (ANIMAL).

57 - 59 FORTH DIMENSIONS devoted an entire issure (II/3) to the CASE structure, and various suggestions for implementing it in FORTH. Screen 57 presents Major Robert Selzer's suggestion. It's use is seen on screen 59, in the example of a video editor, called VEDIT. The proposed CASE construct has the advantage of not requiring sequential case numbering, and the disadvantage of requiring a THEN for each use of CASE. However, his example, VEDIT (with supporting definitions on screen 58) is excellent. In a later issue of FORTH DIMENSIONS (II/5) Edgar H. Fey presented a greatly expanded video editor constructed on the principles outlined by Selzer.

60 - 62 An implementation of graphics on the ABC80. It controls a matrix of 78 (horizontal=X) by 72 (vertical=Y).

45 52 SET.DOT will set a dot at X=45 and Y=52.

45 52 CLEAR.DOT will clear that dot.

45 52 ?DOT will be TRUE (<>0) if the dot at X=45, Y=52 is set, otherwise, FALSE(0).

The above definitions are found on screen 60. A few examples are found on screen 61, and some commentary text, on screen 62.

63 - 64 Y/N is a word that is used to ask a question requiring a yes/no answer in a program, and wait for the reply. A very simple example of its use is found on screen 64.

Temporary screens, included on the first cassette, but over-written by the second cassette.

34 Non-destructive stack print, adapted from BRODIE's book.

35 F, an example from BRODIE

36 This screen shows the FORTH-79 definition of some FORTH words. Before trying to use Brodie's examples which require CREATE, you should enter this definition, since fig-FORTH's CREATE works differently than poly-FORTH's CREATE. But do not enter this version of CREATE before loading the assembler, since the assembler is built around fig-FORTH's CREATE!

*** REFERENSLISTA ***

FORTH

1. Bernstein, Mark 1982. The Computer Toolbox. - Byte, March 1982. KOD (SÄRKOD): instrumentation, control (art)

2. Borden, D.W. 1979. GLOSSARY. - FD I/4 p 44. KOD (SÄRKOD): improved HELP (art)

3. Brodie, Leo 1981. Starting FORTH. - Prentice Hall. KOD (SÄRKOD): general (book)

4. Burton, M. 1982. The Game of Reverse. - FD III/5. KOD (SÄRKOD): Reverse, game (art)

5. Burton, Michael 1981. Increasing fig-FORTH Disk Access Speed. - FD III/2. KOD (SÄRKOD): skew factor for sectran, CP/M (art)

6. Butler, David 1982. A Video Version of Master Mind. - FD III/5. KOD (SÄRKOD): Master Mind, game (art)

7. Cassady, John J. 1979. fig-FORTH for 8080 Assembly Source Listing Release 1.1. - Forth Interest Group. KOD (SÄRKOD): source code, 8080, CP/M (man)

8. Cassady, John J. 1980. 8080 Assembler. - private. KOD (SÄRKOD): assembler, 8080 (priv)

9. Cole, Barry A. 1981. A Stack Diagram Utility. - FD III/1 p 23. KOD (SÄRKOD): stack diagram (art)

10. Daniel, S.H. 1981. The Forth, Inc. Line Editor. - FD III/3 p 80. KOD (SÄRKOD): editor, poly-FORTH (art)

11. Dessy, Raymond E. & Starling, Michael K. 1980. Fourth generation languages. - International Laboratory, March 1980. KOD (SÄRKOD): general, multi-tasking, 16 bit (art)

12. Fey, Edgar H. 1980. FEDIT. - FD II/5 p141. KOD (SÄRKOD): FEDIT, editor, errata in FD III/4 and FD III/5 (art), see VEDIT (art)

13. Fritzson, Richard 1981. Write Your Own FORTH Interpreter. - Microcomputing, Feb. 1981. KOD (SÄRKOD): interpreter, mini-FORTH (art)

14. Fritzson, Richard 1981. Write Your Own Pseudo-FORTH Compiler. - Microcomputing, March 1981. KOD (SÄRKOD): compiler, mini-FORTH (art)

15. Grotke, Guy T. 1982. Transfer of Forth Screens by Modem. - FD III/5 p 162. KOD (SÄRKOD): modem, transfer, communication (art)

16. Harris, Kim 1980. FORTH Extensibility, Or How to Write a Compiler in 25 Words or Less. - Byte, Aug. 1980. KOD (SÄRKOD): excellent introduction to creating new defining words (art)

17. Harris, Kim 1981. Some New Editor Extensions. - FD II/6 p 156. KOD (SÄRKOD): NEW, UNDER (art)

18. Haydon, Glen B. 1981. Elements of a FORTH Data Base Design. - FD III/2. KOD (SÄRKOD): data base (art)

19. Hogan, Thom 1982. Discover FORTH. - Osborne/McGraw-Hill. KOD (SÄRKOD): general (book)

20. James, John S. 1980. What is FORTH? A Tutorial Introduction. - Byte, Aug. 1980. KOD (SÄRKOD): general (art)

21. Laxen, Henry 1982. A Technical Tutorial: Table Lookup Examples. - FD III/5. KOD (SÄRKOD): table lookup (art)

22. Lewis, Tony 1982. The 31 Game. - FD III/5. KOD (SÄRKOD): 31, game (art)

23. Loeliger, R.G. 1981. Threaded Interpretive Languages. - Byte Books/McGraw-Hill. KOD (SÄRKOD): general, Z80 (book)

24. Miles, William D. 1981. Using ENCLOSE on 8080. - FD III/2. KOD (SÄRKOD): ENCLOSE for blocks >=256 (art)

25. Miller, A. Richard & Miller, Jill 1980. BREAKFORTH Into FORTH. - Byte, Aug. 1980. KOD (SÄRKOD): BREAKFORTH, game, MMSFORTH (art)

26. Petersen, Joel V. 1981. Recursion and the Ackermann Function. - FD III/3 p 89. KOD (SÄRKOD): recursion, MYSELF, Ackermann (art)

27. Ragsdale, William F. 1978. HELP . - FD I/2 p 19. KOD (SÄRKOD): HELP (art)

28. Ragsdale, William F. 1980. Installation Manual. - Forth Interest Group. KOD (SÄRKOD): glossary, model, editor (man)

29. Reece, Peter 1982. A Disk Operating System for FORTH. - Byte, April, 1982 p 322. KOD (SÄRKOD): DOS, operating system (art)

30. Selzer, Major Robert A. 1980. VEDIT. - FD II/3. KOD (SÄRKOD): VEDIT, video editor; CASE (art)

31. Ting 1980. Systems guide to fig-FORTH. - Mountain View Press. KOD (SÄRKOD): general, inner workings (book)

32. van der Eijk, Paul 1981. Optimizing Dictionary Searches. - FD III/2 p 57. KOD (SÄRKOD): link-field first (art)

33. van der Eijk, Paul 1981. Tracing Colon Definitions. - FD III/2 p 58. KOD (SÄRKOD): TRACE (art)

34. White, Art 1982. Forth for the Novice. - Microcomputing, Feb. 1982. KOD (SÄRKOD): How it works (art)

35. Williams, Gregg 1980. Forth Glossary. - Byte, Aug. 1980. KOD (SÄRKOD): glossary, general (art)

fig-FORTH INSTALLATION MANUAL

1.0 INTRODUCTION

The fig-FORTH implementation project occurred because a key group of Forth fanciers wished to make this valuable tool available on a personal computing level. In June of 1978, we gathered a team of nine systems level programmers, each with a particular target computer. The charter of the group was to translate a common model of Forth into assembly language listings for each computer. It was agreed that the group's work would be distributed in the public domain by FIG. This publication series is the conclusion of the work.

2.0 DISTRIBUTION

All publications of the Forth Interest Group are public domain. They may be further reproduced and distributed by inclusion of this credit notice:

> This publication has been made available by the Forth Interest Group, P. O. Box 1105, San Carlos, Ca 94070

We intend that our primary recipients of the Implementation Project be computer users groups, libraries, and commercial vendors. We expect that each will further customize for particular computers and redistribute. No restrictions are placed on cost, but we expect faithfulness to the model. FIG does not intend to distribute machine readable versions, as that entails customization, revision, and customer support better reserved for commerical vendors.

Of course, another broad group of recipients of the work is the community of personal computer users. We hope that our publications will aid in the use of Forth and increase the user expectation of the performance of high level computer languages.

3.0 MODEL ORGINIZATION

The fig-FORTH model deviates a bit from the usual loading method of Forth. Existing systems load about 2k bytes in object form and then self-compile the resident system (6 to 8 k bytes). This technique allows customization within the high level portion, but is impractical for new implementors.

Our model has 4 to 5 k bytes written as assembler listings. The remainder may be compiled typing in the Forth high-level source, by more assembly source, or by disc compilation. This method enhances transportability, although the larger portion in assembly code entails more effort. About 8k bytes of memory is used plus 2 to 8k for workspace.

3.1 MODEL OVER-VIEW

The model consists of 7 distinct areas. They occur sequentially from low memory to high.

- Boot-up parameters
- Machine code definitions
- High level utility definitions
- Installation dependent code
- High level definitions
- System tools (optional)
- RAM memory workspace

3.2 MODEL DETAILS

Boot-up Parameters

This area consists of 34 bytes containing a jump to the cold start, jump to the warm re-start and initial values for user variables and registers. These values are altered as you make permanent extensions to your installation.

Machine Code Definitions

This area consists of about 600 to 800 bytes of machine executable code in the form of Forth word defintions. Its purpose is to convert your computer into a standard Forth stack computer. Above this code, the balance of Forth contains a pseudo-code compiled of "execution-addresses" which are sequences of the machine address of the "code-fields" of other Forth definitions. All execution ultimately refers to the machine code definitions.

High-level Utility Definitions

These are colon-definitions, user variables, constants, and variables that allow you to control the "Forth stack computer". They comprise the bulk of the system, enabling you to execute and compile from the terminal. If disc storage (or a RAM simulation of disc) is available, you may also execute and compile from this facility. Changes in the high-level area are infrequent. They may be made thru the assembler source listings.

Installation Dependent Code

This area is the only portion that need change between different installations of the same computer cpu. There are four code fragments:

(KEY) Push the next ascii value (7 bits) from the terminal keystroke to the computation stack and execute NEXT. High 9 bits are zero. Do not echo this character, especially a control character.

(EMIT) Pop the computation stack (16 bit value). Display the low 7 bits on the terminal device, then execute NEXT. Control characters have their natural functions.

(?TERMINAL) For terminals with a break key, wait till released and push to the computation stack 0001 if it was found depressed; otherwise 0000. Execute NEXT. If no break key is available, sense any key depression as a break (sense but don't wait for a key). If both the above are unavailable, simply push 0000 and execute NEXT.

(CR) Execute a terminal carriage return and line feed. Execute NEXT.

When each of these words is executed, the intepreter vectors from the definition header to these code sequences. On specific implementations it may be necessary to preseve certain registers and observe operating system protocols. Understand the implementors methods in the listing before proceeding!

R/W This colon-definition is the standard linkage to your disc. It requests the read or write of a disc sector. It usually requires supporting code definitions. It may consist of self-contained code or call ROM monitor code. When R/W is assembled, its code field address is inserted once in BLOCK and once in BUFFER.

An alternate version of R/W is included that simulates disc storage in RAM. If you have over 16 k bytes this is practical for startup and limited operation with cassette.

High-level Definitions

The next section contains about 30 definitions involving user interaction: compiling aids, finding, forgetting, listing, and number formating. These definitions are placed above the installation dependent code to facilitate modification. That is, once your full system is up, you may FORGET part of the high-level and re-compile altered definitions from disc.

Sytsem Tools

A text editor and machine code assembler are normally resident. We are including a sample editor, and hope to provide Forth assemblers. The editor is compiled from the terminal the first time, and then used to place the editor and assembler source code on disc.

It is essential that you regard the assembly listing as just a way to get Forth installed on your system. Additions and changes must be planned and tested at the usual Forth high level and then the assembly routines updated. Forth work planned and executed only at an assembly level tends to be non-portable, and confusing.

RAM Workspace

For a single user system, at least 2k bytes must be available above the compiled system (the dictionary). A 16k byte total system is most typical.

The RAM workspace contains the computation and return stacks, user area, terminal input buffer, disc buffer and compilation space for the dictionary.

4.0 INSTALLATION

We see the following methods of getting a functioning fig-FORTH system:

1. Buy loadable object code from a vendor who has customized.
2. Obtain an assembly listing with the installation dependent code supplied by the vendor. Assemble and execute.
3. Edit the FIG assembly listing on your system, re-write the I-O routines, and assemble.
4. Load someone else's object code up to the installation dependent code. Hand assemble equivalents for your system and poke in with your monitor. Begin execution and type in (self-compile) the rest of the system. This takes about two hours once you understand the structure of Forth (but that will take much more time!).

Let us examine Step 3, above, in fuller detail. If you wish to bring up Forth only from this model, here are the sequential steps:

4.1 Familiarize yourself with the model written in Forth, the glossary, and specific assembly listings.

4.2 Edit the assembly listings into your system. Set the boot-up parameters at origin offset 0A, 0B (bytes) to 0000 (warning=00).

4.3 Alter the terminal support code (KEY, EMIT, etc,) to match your system. Observe register protocol specific to your implementation!

4.4 Place a break to your monitor at the end of NEXT, just before indirectly jumping via register W to execution. W is the Forth name for the register holding a code field address, and may be differently referenced in your listings.

4.5 Enter the cold start at the origin. Upon the break, check that the interpretive pointer IP points within ABORT and W points to SP!. If COLD is a colon-definition, then the IP has been initialized on the way to NEXT and your testing will begin in COLD. The purpose of COLD is to initialize IP, SP, RP, UP, and some user variables from the start-up parameters at the origin.

4.6 Continue execution one word at a time. Clever individuals could write a simple trace routine to print IP, W, SP, RP and the top of the stacks. Run in this single step mode until the greeting message is printed. Note that the interpretation is several hundred cycles to this stage!

4.7 Execution errors may be localized by observing the above pointers when a crash occurs.

4.8 After the word QUIT is executed (incrementally), and you can input a "return" key and get OK printed, remove the break. You may have some remaining errors, but a reset and examination of the above registers will again localize problems.

4.9 When the system is interpreting from the keyboard, execute EMPTY-BUFFERS to clear the disc buffer area. You may test the disc access by typing: 0 BLOCK 64 TYPE
This should bring sector zero from the disc to a buffer and type the first 64 characters. This sector usually contains ascii text of the disc directory. If BLOCK (and R/W) doesn't function--happy hunting!

5.0 If your disc driver differs from the assembly version, you must create your own R/W. This word does a range check (with error message), modulo math to derive sector, track, and drive and passes values to a sector-read and sector-write routine.

RAM DISC SIMULATION

If disc is not available, a simulation of BLOCK and BUFFER may be made in RAM. The following definitions setup high memory as mass storage. Referenced 'screens' are then brought to the 'disc buffer' area. This is a good method to test the start-up program even if disc may be available.

```
HEX
4000  CONSTANT  LO   ( START OF BUFFER AREA )
6800  CONSTANT  HI   ( 10 SCREEN EQUIVALENT )
: R/W  >R  ( save boolean )
   B/BUF  *  LO  +  DUP
   HI  >  6  ?ERROR ( range check )
   R>  IF ( read )  SWAP  ENDIF
   B/BUF  CMOVE  ;
```

Insert the code field address of R/W into BLOCK and BUFFER and proceed as if testing disc. R/W simulates screens 0 thru 9 when B/BUF is 128, in the memory area $4000 thru $6BFF.

fig-FORTH VARIABLE NAME FIELD

A major FIG innovation in this model, is the introduction of variable length definition names in compiled dictionary entries. Previous methods only saved three letters and the character count.

The user may select the letter count saved, up to the full natural length. See the glossary definition for WIDTH.

In this model, the following conventions have been established.

1. The first byte of the name field has the natural character count in the low 5 bits.
2. The sixth bit = 1 when smudged, and will prevent a match by (FIND).
3. The seventh bit = 1 for IMMEDIATE definitions; it is called the precedence bit.
4. The eighth or sign bit is always = 1.
5. The following bytes contain the names' letters, up to the value in WIDTH.
6. In the byte containing the last letter saved, the sign bit = 1.
7. In word addressing computer, a name may be padded with a blank to a word boundary.

The above methods are implemented in CREATE. Remember that -FIND uses BL WORD to bring the next text to HERE with the count preceeding. All that is necessary, is to limit by WIDTH and toggle the proper delimiting bits.

5.0 MEMORY MAP

The following memory map is broadly used. Specific installations may require alterations but you may forfeit functions in future FIG offerings.

The disc buffer area is at the upper bound of RAM memory. It is comprised of an integral number of buffers, each B/BUF+4 bytes. B/BUF is the number of bytes read from the disc, usually one sector. B/BUF must be a power of two (64, 128, 256, 512 or 1024). The constant FIRST has the value of the address of the start of the first buffer. LIMIT has the value of the first address beyond the top buffer. The distance between FIRST and LIMIT must be N*(B/BUF+4) bytes. This N must be two or more.

Constant B/SCR has the value of the number of buffers per screen; i.e. 1024 / B/BUF.

The user area must be at least 34 bytes; 48 is more appropriate. In a multi-user system, each user has his own user area, for his copy of system variables. This method allows reentrant use of the Forth vocabulary.

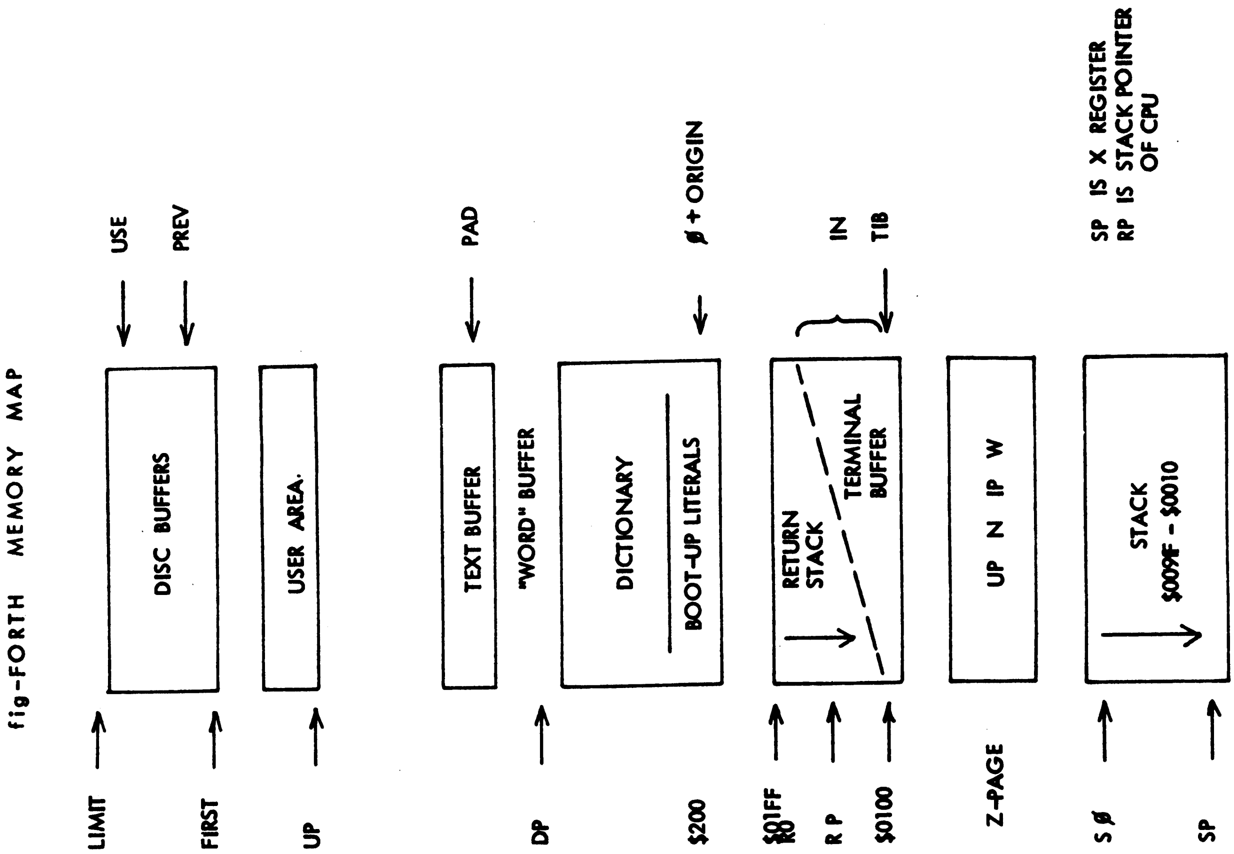
The terminal input buffer is decimal 80 bytes (the hex 50 in QUERY) plus 2 at the end. If a different value is desired, change the limit in QUERY. A parameter in the boot-up literals locates the address of this area for TIB. The backspace character is also in the boot-up origin parameters. It is universally expected that "rubout" is the backspace.

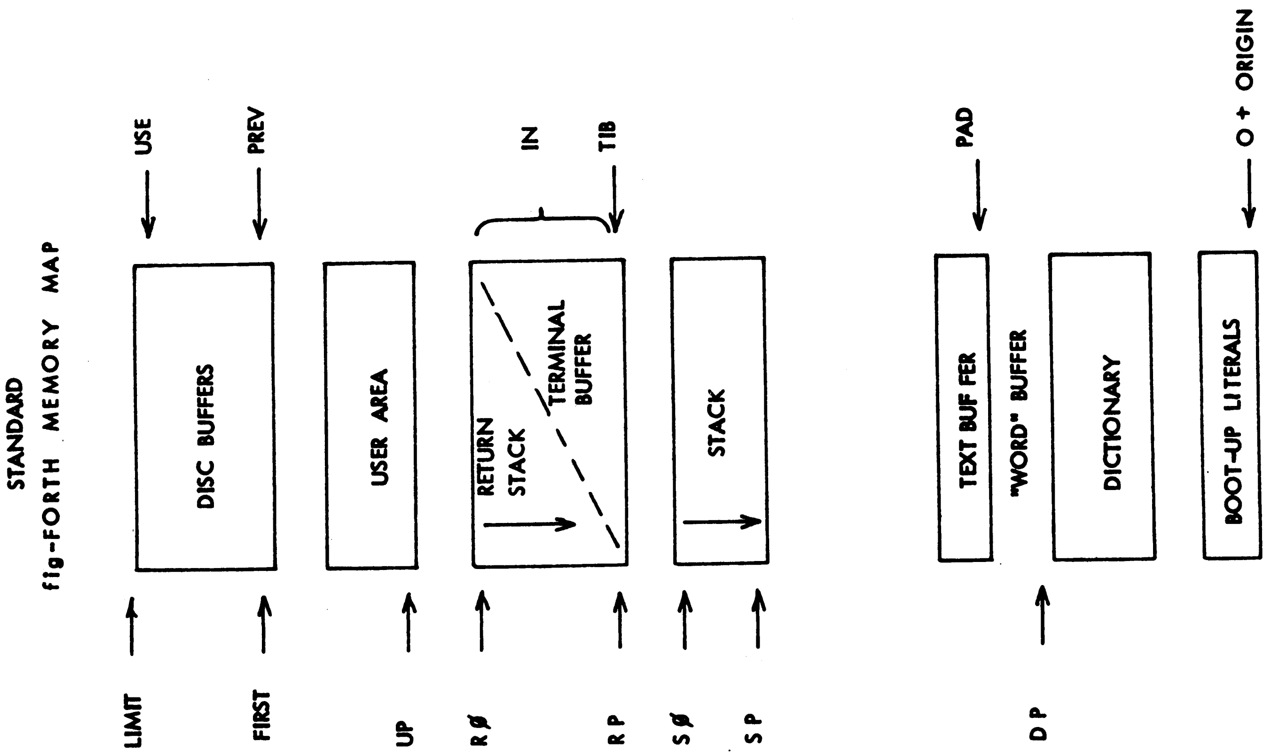
The return stack grows downward from the user area toward the terminal buffer. Forty-eight bytes are sufficient. The origin is in R0 (R-zero) and is loaded from a boot-up literal.

The computation stack grows downward from the terminal buffer toward the dictionary, which grows upward. The origin of the stack is is in variable S0 (S-zero) and is loaded from a boot-up literal.

After a cold start, the user variables contain the addresses of the above memory assignments. An advanced user may relocate while the system is running. A newcomer should alter the startup literals and execute COLD. The word +ORIGIN is provided for this purpose. +ORIGIN gives the address byte or word relative to the origin depending on the computer addressing method. To change the backspace to contol H type:

```
HEX  08  0E  +ORIGIN  !  ( byte addresses)
```


6502
fig-FORTH MEMORY MAP
LIMIT
FIRST
UP
DP
$200
$01FF
R0
R P
$0100
Z-PAGE
S ∅
SP
USE
PREV
PAD
∅ + ORIGIN
IN
TIB
DISC BUFFERS
USER AREA.
TEXT BUFFER
"WORD" BUFFER
DICTIONARY
BOOT-UP LITERALS
RETURN STACK
TERMINAL BUFFER
UP N IP W
STACK
$009F - $0010
SP IS X REGISTER
RP IS STACK POINTER OF CPU


STANDARD
fig-FORTH MEMORY MAP
LIMIT
FIRST
UP
R∅
R P
S∅
S P
D P
USE
PREV
IN
TIB
PAD
O + ORIGIN
DISC BUFFERS
USER AREA
RETURN STACK
TERMINAL BUFFER
STACK
TEXT BUF FER
"WORD" BUFFER
DICTIONARY
BOOT-UP LITERALS

This glossary contains all of the word definitions in Release 1 of fig-FORTH. The definitions are presented in the order of their ascii sort.

The first line of each entry shows a symbolic description of the action of the proceedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes "---" indicate the execution point; any parameters left on the stack are listed. In this notation, the top of the stack is to the right.

The symbols include:

addr	memory address
b	8 bit byte (i.e. hi 8 bits zero)
c	7 bit ascii character (hi 9 bits zero)
d	32 bit signed double integer, most significant portion with sign on top of stack.
f	boolean flag. 0=false, non-zero=true
ff	boolean false flag=0
n	16 bit signed integer number
u	16 bit unsigned integer
tf	boolean true flag=non-zero

The capital letters on the right show definition characteristics:

C	May only be used within a colon definition. A digit indicates number of memory addresses used, if other than one.
E	Intended for execution only.
L0	Level Zero definition of FORTH-78
L1	Level One definition of FORTH-78
P	Has precedence bit set. Will execute even when compiling.
U	A user variable.

Unless otherwise noted, all references to numbers are for 16 bit signed integers. On 8 bit data bus computers, the high byte of a number is on top of the stack, with the sign in the leftmost bit. For 32 bit signed double numbers, the most significant part (with the sign) is on top.

All arithemetic is implicitly 16 bit signed integer math, with error and under-flow indication unspecified.

! n addr --- L0

Store 16 bits of n at address. Pronounced "store".

!CSP

Save the stack position in CSP. Used as part of the compiler security.

d1 --- d2 L0

Generate 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. Used between <# and #>. See #S.

#> d --- addr count L0

Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.

#S d1 --- d2 L0

Generates ascii text in the text output buffer, by the use of #, until a zero double number n2 results. Used between <# and #>.

' --- addr P,L0

Used in the form:

' nnnn

Leaves the parameter field address of dictionary word nnnn. As a compiler directive, executes in a colon-definition to compile the address as a literal. If the word is not found after a search of CONTEXT and CURRENT, an appropriate error message is given. Pronounced "tick".

(P,L0

Used in the form:

(cccc)

Ignore a comment that will be delimited by a right parenthesis on the same line. May occur during execution or in a colon-definition. A blank after the leading parenthesis is required.

(.") C+

The run-time proceedure, compiled by ." which transmits the following in-line text to the selected output device. See ."

(;CODE) C

The run-time proceedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.

(+LOOP) n --- C2

The run-time proceedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.

(ABORT)

Executes after an error when WARNING is -1. This word normally executes ABORT, but may be altered (with care) to a user's alternative proceedure.

(DO) C

The run-time proceedure compiled by DO which moves the loop control parameters to the return stack. See DO.

(FIND) addr1 addr2 --- pfa b tf (ok)
addr1 addr2 --- ff (bad)

Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field and boolean true for a good match. If no match is found, only a boolean false is left.

(LINE) n1 n2 --- addr count

Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 64 indicates the full line text length.

(LOOP) C2

The run-time proceedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.

(NUMBER) d1 addr1 --- d2 addr2

Convert the ascii text beginning at addr1+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.

***** n1 n2 --- prod L0

Leave the signed product of two signed numbers.

***/** n1 n2 n3 --- n4 L0

Leave the ratio n4 = n1*n2/n3 where all are signed numbers. Retention of an intermediate 31 bit product permits greater accuracy than would be available with the sequence:

n1 n2 * n3 /

***/MOD** n1 n2 n3 --- n4 n5 L0

Leave the quotient n5 and remainder n4 of the operation n1*n2/n3
A 31 bit intermediate product is used as for */.

+ n1 n2 --- sum LO
Leave the sum of n1+n2.

+! n addr --- LO
Add n to the value at the address. Pronounced "plus-store".

+- n1 n2 --- n3
Apply the sign of n2 to n1, which is left as n3.

+BUF addl --- addr2 f
Advance the disc buffer address addr1 to the address of the next buffer addr2. Boolean f is false when addr2 is the buffer presently pointed to by variable PREV.

+LOOP n1 --- (run)
addr n2 --- (compile) P,C2,LO
Used in a colon-definition in the form:

DO ... n1 +LOOP

At run-time, +LOOP selectively controls branching back to the corresponding DO based on n1, the loop index and the loop limit. The signed increment n1 is added to the index and the total compared to the limit. The branch back to DO occurs until the new index is equal to or greater than the limit (n1>0), or until the new index is equal to or less than the limit (n1<0). Upon exiting the loop, the parameters are discarded and execution continues ahead.

At compile time, +LOOP compiles the run-time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. n2 is used for compile time error checking.

+ORIGIN n --- addr
Leave the memory address relative by n to the origin parameter area. n is the minimum address unit, either byte or word. This definition is used to access or modify the boot-up parameters at the origin area.

, n --- LO
Store n into the next available dictionary memory cell, advancing the dictionary pointer. (comma)

- n1 n2 --- diff LO
Leave the difference of n1-n2.

--> P,LO
Continue interpretation with the next disc screen. (pronounced next-screen).

-DUP n1 -- n1 (if zero)
n1 -- n1 n1 (non-zero) LO
Reproduce n1 only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

-FIND --- pfa b tf (found)
--- ff (not found)
Accepts the next text word (delimited by blanks) in the input strean to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true is left. Otherwise, only a boolean false is left.

-TRAILING addr n1 --- addr n2
Adjusts the character count n1 of a text string beginning address to suppress the output of trailing blanks. i.e. the characters at addr+n1 to addr+n2 are blanks.

. n --- LO
Print a number from a signed 16 bit two's complement value, converted according to the numeric BASE. A trailing blanks follows. Pronounced "dot".

." P,LO
Used in the form:

." cccc"

Compiles an in-line string cccc (delimited by the trailing ") with an execution proceedure to transmit the text to the selected output device. If executed outside a definition, ." will immediately print the text until the final ". The maximum number of characters may be an installation dependent value. See (.").

.LINE line scr ---
Print on the terminal device, a line of text from the disc by its line and screen number. Trailing blanks are suppressed.

.R n1 n2 ---
Print the number n1 right aligned in a field whose width is n2. No following blank is printed.

/ n1 n2 --- quot LO
Leave the signed quotient of n1/n2.

/MOD n1 n2 --- rem quot LO
Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

0 1 2 3 --- n

These small numbers are used so often that is is attractive to define them by name in the dictionary as constants.

0< n --- f L0

Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.

0= n --- f L0

Leave a true flag is the number is equal to zero, otherwise leave a false flag.

0BRANCH f --- C2

The run-time proceedure to conditionally branch. If f is false (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL, and WHILE.

1+ n1 --- n2 L1

Increment n1 by 1.

2+ n1 --- n2

Leave n1 incremented by 2.

: P,E,L0

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'. The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled.

; P,C,L0

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

;CODE P,C,L0

Used in the form:

    : cccc .... ;CODE
        assembly mnemonics

Stop compilation and terminate a new defining word cccc by compiling (;CODE). Set the CONTEXT vocabulary to ASSEMBLER, assembling to machine code the following mnemonics.

When cccc later executes in the form:

    cccc nnnn

the word nnnn will be created with its execution proceedure given by by the machine code following cccc. That is, when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

;S P,L0

Stop interpretation of a screen. ;S is also the run-time word compiled at the end of a colon-definition which returns execution to the calling proceedure.

< n1 n2 --- f L0

Leave a true flag if n1 is less than n2; otherwise leave a false flag.

<# L0

Setup for pictured numeric output formatting using the words:

    <# # #S SIGN #>

The conversion is done on a double number producing text at PAD.

<BUILDS C,L0

Used within a colon-definition:

    : cccc <BUILDS ...
        DOES> ... ;

Each time cccc is executed, <BUILDS defines a new word with a high-level execution proceedure. Executing cccc in the form:

    cccc nnnn

uses <BUILDS to create a dictionary entry for nnnn with a call to the DOES> part for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time proceedures to written in high-level rather than in assembler code (as required by ;CODE).

= n1 n2 --- f L0

Leave a true flag if n1=n2; otherwise leave a false flag.

> n1 n2 --- f L0

Leave a true flag if n1 is greater than n2; otherwise a false flag.

>R n --- C,L0

Remove a number from the computation stack and place as the most accessable on the return stack. Use should be balanced with R> in the same definition.

? addr -- L0

Print the value contained at the address in free format according to the current base.

?COMP

Issue error message if not compiling.

?CSP

Issue error message if stack position differs from value saved in CSP.

?ERROR f n ---
Issue an error message number n, if the boolean flag is true.

?EXEC
Issue an error message if not executing.

?LOADING
Issue an error message if not loading

?PAIRS n1 n2 ---
Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match.

?STACK
Issue an error message is the stack is out of bounds. This definition may be installation dependent.

?TERMINAL --- f
Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. This definition is installation dependent.

@ addr --- n LO
Leave the 16 bit contents of address.

ABORT LO
Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation.

ABS n --- u LO
Leave the absolute value of n as u.

AGAIN addr n --- (compiling) P,C2,LO
Used in a colon-definion in the form:
BEGIN ... AGAIN
At run-time, AGAIN forces execution to return to corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R> DROP is executed one level below).

At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile-time error checking.

ALLOT n --- LO
Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or re-origin memory. n is with regard to computer address type (byte or word).

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

B/BUF --- n
This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK.

B/SCR --- n
This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each.

BACK addr ---
Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address.

BASE --- addr U,LO
A user variable contaning the current number base used for input and output conversion.

BEGIN --- addr n (compiling) P,LO
Occurs in a colon-definition in form:
BEGIN ... UNTIL
BEGIN ... AGAIN
BEGIN ... WHILE ... REPEAT
At run-time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the correspoinding UNTIL, AGAIN or REPEAT. When executing UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs.

At compile time BEGIN leaves its return address and n for compiler error checking.

BL --- c
A constant that leaves the ascii value for "blank".

BLANKS addr count ---
Fill an area of memory begining at addr with blanks.

BLK --- addr U,LO
A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

BLOCK n --- addr LO
Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from disc to which ever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is rewritten to disc before block n is read into the buffer. See also BUFFER, R/W UPDATE FLUSH

BLOCK-READ
BLOCK-WRITE These are the preferred names for the installation dependent code to read and write one block to the disc.

BRANCH C2,L0
The run-time proceedure to unconditionally branch. An in-line offset is added to the interpretive pointer IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT.

BUFFER n --- addr
Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to the disc The block is not read from the disc. The address left is the first cell within the buffer for data storage.

C! b addr ---
Store 8 bits at address. On word addressing computers, further specification is necessary regarding byte addressing.

C, b ---
Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer. This is only available on byte addressing computers, and should be used with caution on byte addressing minicomputers.

C@ addr --- b
Leave the 8 bit contents of memory address. On word addressing computers, further specification is needed regarding byte addressing.

CFA pfa --- cfa
Convert the parameter field address of a definition to its code field address.

CMOVE from to count ---
Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceeding toward high memory. Further specification is necessary on word addressing computers.

COLD
The cold start proceedure to adjust the dictionary pointer to the minimum standard and restart via ABORT. May be called from the terminal to remove application programs and restart.

COMPILE C2
When the word containing COMPILE executes, the execution address of the word following COMPILE is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in additon to simply compling an execution address (which the interpreter already does).

CONSTANT n --- L0
A defining word used in the form:
n CONSTANT cccc
to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.

CONTEXT --- addr U,L0
A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

COUNT addr1 --- addr2 n L0
Leave the byte address addr2 and byte count n of a message text beginning at address addr1. It is presumed that the first byte at addr1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE.

CR L0
Transmit a carriage return and line feed to the selected output device.

CREATE
A defining word 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 words parameter field. The new word is created in the CURRENT vocablary.

CSP ---- addr U
A user variable temporarily storing the stack pointer position, for compilation error checking.

D+ d1 d2 --- dsum
Leave the double number sum of two double numbers.

D+- d1 n --- d2
Apply the sign of n to the double number d1, leaving it as d2.

D. d --- L1
Print a signed double number from a 32 bit two's complement value. The high-order 16 bits are most accessable on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced D-dot.

D.R d n ---

Print a signed double number d right aligned in a field n characters wide.

DABS d --- ud

Leave the absolute value ud of a double number.

DECIMAL L0

Set the numeric conversion BASE for decimal input-output.

DEFINITIONS L1

Used in the form:

cccc DEFINITIONS

Set 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.

DIGIT c n1 --- n2 tf (ok)
c n1 --- ff (bad)

Converts the ascii character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag.

DLIST

List the names of the dictionary entries in the CONTEXT vocabulary.

DLITERAL d --- d (executing)
d --- (compiling) P

If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it to the stack. If executing, the number will remain on the stack.

DMINUS d1 --- d2

Convert d1 to its double number two's complement.

DO n1 n2 --- (execute)
addr n --- (compile) P,C2,L0

Occurs in a colon-definition in form:

DO ... LOOP
DO ... +LOOP

At run time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop 'I' will copy the

urrent value of the index to the stack. See I, LOOP, +LOOP, LEAVE.

When compiling within the colon-definition, DO compiles (DO), leaves the following address addr and n for later error checking.

DOES> L0

A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multidiminsional arrays, and compiler generation.

DP ---- addr U,L

A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT.

DPL ---- addr U,L0

A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used hold output column location of a decimal point, in user generated formating. The default value on single number input is -1.

DR0
DR1

Installation dependent commands to select disc drives, by preseting OFFSET. The contents of OFFSET is added to the block number in BLOCK to allow for this selection. Offset is supressed for error text so that is may always originate from drive 0.

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

DUMP addr n --- LO
Print the contents of n memory locations beginning at addr. Both addresses and contents are shown in the current numeric base.

DUP n --- n n LO
Duplicate the value on the stack.

ELSE addr1 n1 --- addr2 n2 (compiling) P,C2,L0
Occurs within a colon-definition in the form:
IF ... ELSE ... ENDIF
At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect.

At compile-time ELSE emplaces BRANCH reserving a branch offset, leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.

EMIT c --- LO
Transmit ascii character c to the selected output device. OUT is incremented for each character output.

EMPTY-BUFFERS LO
Mark all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not written to the disc. This is also an initialization proceedure before first use of the disc.

ENCLOSE addr1 c --- ddr1 n1 n2 n3
The text scanning primitive used by WORD. From the text address addr1 and an ascii delimiting character c, is determined 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. This proceedure will not process past an ascii 'null', treating it as an unconditional delimiter.

END P,C2,LO
This is an 'alias' or duplicate definition for UNTIL.

ENDIF addr n --- (compile) P,C0,L0
Occurs in a colon-definition in form:
IF ... ENDIF
IF ... ELSE ... ENDIF
At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDIF. Both names are supported in fig-FORTH. See also IF and ELSE.

At compile-time, ENDIF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error tests.

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

ERROR line --- in blk
Execute error notification and re-start of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive 0 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING=0, n is just printed as a message number (non disc installation). If WARNING is -1, the definition (ABORT) is executed, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). fig-FORTH saves the contents of IN and BLK to assist in determining the location of the error. Final action is execution of QUIT.

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

EXPECT addr count --- LO
Transfer characters from the terminal 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.

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

FILL addr quan b ---
Fill memory at the address with the specified quantity of bytes b.

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

FLD --- addr U

A user variable for control of number output field width. Presently unused in fig-FORTH.

FORGET E,L0

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

FORTH P,L1

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

HERE --- addr L0

Leave the address of the next available dictionary location.

HEX L0

Set the numeric conversion base to sixteen (hexadecimal).

HLD --- addr L0

A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD c --- L0

Used between <# and #> to insert an ascii character into a pictured numeric output string.
e.g. 2E HOLD will place a decimal point.

I --- n C,L0

Used within a DO-LOOP to copy the loop index to the stack. Other use is implementation dependent. See R.

ID. addr ---

Print a definition's name from its name field address.

IF f --- (run-time)
--- addr n (compile) P,C2,L0

Occurs is a colon-definition in form:
IF (tp) ... ENDIF
IF (tp) ... ELSE (fp) ... ENDIF
At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead thru the true part. If f is false (zero), execution skips till just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional.; if missing, false execution skips to just after ENDIF.

At compile-time IF compiles 0BRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.

IMMEDIATE

Mark the most resently made definition so that when encountered at compile time, it will be executed rather than being compiled. i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceeding it with [COMPILE].

IN --- addr L0

A user variable containing the byte offset within the current input text buffer (terminal or disc) from which the next text will be accepted. WORD uses and moves the value of IN.

INDEX from to ---

Print 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 disc screens.

INTERPRET

The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disc) depending on STATE. If the word name cannot be found after a search of CONTEXT and then CURRENT it is converted to a number according to the current base. That also failing, an error message echoing the name with a " ?" will be given.
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. The decimal point has no other purpose than to force this action. See NUMBER.

KEY --- c LO

Leave the ascii value of the next terminal key struck.

LATEST --- addr

Leave the name field address of the topmost word in the CURRENT vocabulary.

LEAVE C,LO

Force termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution prodeeds normally until LOOP or +LOOP is encountered.

LFA pfa --- lfa

Convert the parameter field address of a dictionary definition to its link field address.

LIMIT ---- n

A constant leaving the address just above the highest memory available for a disc buffer. Usually this is the highest system memory.

LIST n --- LO

Display the ascii text of screen n on the selected output device. SCR contains the screen number during and after this process.

LIT --- n C2,LO

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

LITERAL n --- (compiling) P,C2,LO

If compiling, then compile the stack value n as a 16 bit literal. This definition is immediate so that it will execute during a colon definition. The intended use is:

```
  : xxx    [ calculate ]  LITERAL  ;
```

Compilation is suspended for the compile time calculation of a value. Compilation is reusumed and LITERAL compiles this value.

LOAD n --- LO

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

LOOP addr n --- (compiling) P,C2,LO

Occurs in a colon-definition in form:

```
   DO  ...  LOOP
```

At run-time, LOOP selectively controls branching 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.

At compile-time, LOOP compiles (LOOP) and uses addr to calculate an offset to DO. n is used for error testing.

M* n1 n2 --- d

A mixed magnitude math operation which leaves the double number signed product of two signed number.

M/ d n1 --- n2 n3

A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3, from a double number dividend and divisor n1. The remainder takes its sign from the dividend.

M/MOD ud1 u2 --- u3 ud4

An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend ud1 and single divisor u2.

MAX n1 n2 --- max LO

Leave the greater of two numbers.

MESSAGE n ---

Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero, the message will simply be printed as a number (disc un-available).

MIN n1 n2 --- min LO

Leave the smaller of two numbers.

MINUS n1 --- n2 LO

Leave the two's complement of a number.

MOD n1 n2 --- mod LO

Leave the remainder of n1/n2, with the same sign as n1.

MON

Exit to the system monitor, leaving a re-entry to Forth, if possible.

MOVE addr1 addr2 n ---
Move the contents of n memory cells (16 bit contents) beginning at addr1 into n cells beginning at addr2. The contents of addr1 is moved first. This definition is appropriate on on word addressing computers.

NEXT
This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executed but is the return point for all code proceedures. It acts by fetching the address pointed by IP, storing this value in register W. It then jumps to the address pointed to by the address pointed to by W. W points to the code field of a definition which contains the address of the code which executes for that definition. This usage of indirect threaded code is a major contributor to the power, portability, and extensibility of Forth. Locations of IP and W are computer specific.

NFA pfa --- nfa
Convert the parameter field address of a definition to its name field.

NUMBER addr --- d
Convert a character string left at addr with a preceeding 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.

OFFSET --- addr U
A user variable which may contain a block offset to disc drives. The contents of OFFSET is added to the stack number by BLOCK. Messages by MESSAGE are independent of OFFSET. See BLOCK, DR0, DR1, MESSAGE.

OR n1 n2 -- or LO
Leave the bit-wise logical or of two 16 bit values.

OUT --- addr U
A user variable that contains a value incremented by EMIT. The user may alter and examine OUT to control display formating.

OVER n1 n2 --- n1 n2 n1 LO
Copy the second stack value, placing it as the new top.

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

PFA nfa --- pfa
Convert the name field address of a compiled definition to its parameter field address.

POP
The code sequence to remove a stack value and return to NEXT. POP is not directly executable, but is a Forth re-entry point after machine code.

PREV ---- addr
A variable containing the address of the disc buffer most recently referenced. The UPDATE command marks this buffer to be later written to disc.

PUSH
This code sequence pushes machine registers to the computation stack and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code.

PUT
This code sequence stores machine register contents over the topmost computation stack value and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code.

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

QUIT L1
Clear the return stack, stop compilation, and return control to the operators terminal. No message is given.

R --- n
Copy the top of the return stack to the computation stack.

R# --- addr U
A user variable which may contain the location of an editing cursor, or other file related function.

R/W addr blk f ---

The fig-FORTH standard disc read-write linkage. addr specifies the source or destination block buffer, blk is the sequential number of the referenced block; and f is a flag for f=0 write and f=1 read. R/W determines the location on mass storage, performs the read-write and performs any error checking.

R> --- n LO

Remove the top value from the return stack and leave it on the computation stack. See >R and R.

R0 --- addr U

A user variable containing the initial location of the return stack. Pronounced R-zero. See RP!

REPEAT addr n --- (compiling) P,C2

Used within a colon-definition in the form:

BEGIN ... WHILE ... REPEAT

At run-time, REPEAT forces an unconditional branch back to just after the correspoinding BEGIN.

At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.

ROT n1 n2 n3 --- n2 n3 n1 LO

Rotate the top three values on the stack, bringing the third to the top.

RP!

A computer dependent proceedure to initialize the return stack pointer from user variable R0.

S->D n --- d

Sign extend a single number to form a double number.

S0 --- addr U

A user variable that contains the initial value for the stack pointer. Pronounced S-zero. See SP!

SCR --- addr U

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

SIGN n d --- d LO

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

SMUDGE

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

SP!

A computer dependent proceedure to initialize the stack pointer from S0.

SP@ --- addr

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

SPACE LO

Transmit an ascii blank to the output device.

SPACES n --- LO

Transmit n ascii blanks to the output device.

STATE --- addr LO,U

A user variable containg the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.

SWAP n1 n2 --- n2 n1 LO

Exchange the top two values on the stack.

TASK

A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety.

THEN P,C0,LO

An alias for ENDIF.

TIB --- addr U

A user variable containing the address of the terminal input buffer.

TOGGLE addr b ---

Complement the contents of addr by the bit pattern b.

TRAVERSE addr1 n --- addr2

Move across the name field of a fig-FORTH variable length name field. addr1 is the address of either the length byte or the last letter. If n=1, the motion is toward hi memory; if n=-1, the motion is toward low memory. The addr2 resulting is address of the other end of the name.

TRIAD scr ---

Display on the selected output device the three screens which include that numbered scr, begining with a screen evenly divisible by three. Output is suitable for source text records, and includes a reference line at the bottom taken from line 15 of screen4.

TYPE addr count --- LO

Transmit count characters from addr to the selected output device.

U* u1 u2 --- ud

Leave the unsigned double number product of two unsigned numbers.

U/ ud u1 --- u2 u3

Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

UNTIL f --- (run-time)
addr n --- (compile) P,C2,LO

Occurs within a colon-definition in the form:

BEGIN ... UNTIL

At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues ahead.

At compile-time, UNTIL compiles (OBRANCH) and an offset from HERE to addr. n is used for error tests.

UPDATE LO

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

USE --- addr

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

USER n --- LO

A defining word used in the form:

n USER cccc

which 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. When cccc is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable.

VARIABLE E,LO

A defining word used in the form:

n VARIABLE cccc

When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch or store may access this location.

VOC-LINK --- addr U

A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETting thru multiple vocabularys.

VOCABULARY E,L

A defining word used in the form:

VOCABULARY cccc

to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary into which new definitions are placed.

In fig-FORTH, cccc will be so chained as to include all definitions of the vocabulary in which cccc is itself defined. All vocabularys ulitmately chain to Forth. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.

VLIST

List the names of the definitions in the context vocabulary. "Break" will terminate the listing.

WARNING --- addr U

A user variable containing a value controlling messages. If = 1 disc is present, and screen 4 of drive 0 is the base location for messages. If = 0, no disc is present and messages will be presented by number. If = -1, execute (ABORT) for a user specified proceedure. See MESSAGE, ERROR.

WHILE f --- (run-time)
ad1 n1 --- ad1 n1 ad2 n2 P,C2

Occurs in a colon-definition in the form:

BEGIN ... WHILE (tp) ... REPEAT

At run-time, WHILE selects conditional execution based on boolean flag f. If f is true (non-zero), WHILE continues execution of the true part thru to REPEAT, which then branches back to BEGIN. If f is false (zero), execution skips to just after REPEAT, exiting the structure.

At compile time, WHILE emplaces (OBRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.

WIDTH --- addr U

In fig-FORTH, a user variable containing the maximum number of letters saved in the compilation of a definitions' name. It must be 1 thru 31, with a default value of 31. The name character count and its natural characters are saved, up to the value in WIDTH. The value may be changed at any time within the above limits.

WORD c --- LO

Read the next text characters from the input stream being interpreted, until a delimiter c is found, storing the packed character string begining at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and ends with two or more blanks. Leading occurances of c are ignored. If BLK is zero, text is taken from the terminal input buffer, otherwise from the disc block stored in BLK. See BLK, IN.

X

This is pseudonym for the "null" or dictionary entry for a name of one character of ascii null. It is the execution proceedure to terminate interpretation of a line of text from the terminal or within a disc buffer, as both buffers always have a null at the end.

XOR n1 n2 --- xor L1

Leave the bitwise logical exclusive-or of two values.

[P,L1

Used in a colon-definition in form:

```
: xxx   [  words  ]   more  ;
```

Suspend compilation. The words after [are executed, not compiled. This allows calculation or compilation exceptions before resuming compilation with]. See LITERAL,].

[COMPILE] P,C

Used in a colon-definition in form:

```
: xxx   [COMPILE]  FORTH  ;
```

[COMPILE] will force the compilation of an immediate defininition, that would otherwise execute during compilation. The above example will select the FORTH vocabulary when xxx executes, rather than at compile time.

] L1

Resume compilation, to the completion of a colon-definition. See [.

EDITOR USER MANUAL

by Bill Stoddart
of FIG, United Kingdom

FORTH organizes its mass storage into "screens" of 1024 characters. If, for example, a diskette of 250k byte capacity is used entirely for storing text, it will appear to the user as 250 screens numbered 0 to 249.

Each screen is organized as 16 lines with 64 characters per line. The FORTH screens are merely an arrangement of virtual memory and need not correspond exactly with the screen format of a particular terminal.

Selecting a Screen and Input of Text

To start an editing session the user types EDITOR to invoke the appropriate vocabulary.

The screen to be edited is then selected, using either:

```
n LIST  ( list screen n and select it for editing ) OR
n CLEAR ( clear screen n and select for editing )
```

To input new test to screen n after LIST or CLEAR the P (put) command is used.

Example:

```
0 P THIS IS HOW
1 P TO INPUT TEXT
2 P TO LINES 0, 1, AND 2 OF THE SELECTED SCREEN.
```

Line Editing

During this descirption of the editor, reference is made to PAD. This is a text buffer which may hold a line of text used by or saved with a line editing command, or a text string to be found or deleted by a string editing command.

PAD can be used to transfer a line from one screen to another, as well as to perform edit operations within a single screen.

Line Editor Commands

n H Hold line n at PAD (used by system more often than by user).

n D Delete line n but hold it in PAD. Line 15 becomes blank as lines n+1 to 15 move up 1 line.

n T Type line n and save it in PAD.

n R Replace line n with the text in PAD.

n I Insert the text from PAD at line n, moving the old line n and following lines down. Line 15 is lost.

n E Erase line n with blanks.

n S Spread at line n. n and subsequent lines move down 1 line. Line n becomes blank. Line 15 is lost.

Cursor Control and String Editing

The screen of text being edited resides in a buffer area of storage. The editing cursor is a variable holding an offset into this buffer area. Commands are provided for the user to position the cursor, either directly or by searching for a string of buffer text, and to insert or delete text at the cursor position.

Commands to Position the Cursor

TOP	Position the cursor at the start of the screen.
N M	Move the cursor by a signed amount n and print the cursor line. The position of the cursor on its line is shown by a __ (underline).

String Editing Commands

F text	Search forward from the current cursor position until string "text" is found. The cursor is left at the end of the text string, and the cursor line is printed. If the string is not found an error message is given and the cursor is repositioned at the top of screen.
B	Used after F to back up the cursor by the length of the most recent text.
N	Find the next occurrence of the string found by an F command.
X text	Find and delete the string "text."
C text	Copy in text to the cursor line at the cursor position.
TILL text	Delete on the cursor line from the cursor till the end of the text string "text."
NOTE:	Typing C with <u>no text</u> will copy a null into the text at the cursor position. This will abruptly stop later compiling! To delete this error type TOP X 'return'.

Screen Editing Commands

n LIST	List screen n and select it for editing
n CLEAR	Clear screen n with blanks and select it for editing
n1 n2 COPY	Copy screen n1 to screen n2.
L	List the current screen. The cursor line is relisted after the screen listing, to show the cursor position.
FLUSH	Used at the end of an editing session to ensure that all entries and updates of text have been transferred to disc.

Editor Glossary

TEXT c ---
Accept following text to pad. c is text delimiter.

LINE n --- addr
Leave address of line n of current screen. This address will be in the disc buffer area.

WHERE n1 n2 ---
n2 is the block no., n1 is offset into block. If an error is found in the source when loading from disc, the recovery routine ERROR leaves these values on the stack to help the user locate the error. WHERE uses these to print the screen and line nos. and a picture of where the error occurred.

R# --- addr
A user variable which contains the offset of th editing cursor from the start of the screen.

#LOCATE --- n1 n2
From the cursor position determine the line-no n2 and the offset into the line n1.

#LEAD --- line-address offset-to-cursor

#LAG --- cursor-address count-after-cursor-till-EOL

-MOVE addr line-no ---
Move a line of text from addr to line of current screen.

H n ---
Hold numbered line at PAD.

E n ---
Erase line n with blanks.

S n ---
Spread. Lines n and following move down. n becomes blank.

D n ---
Delete line n, but hold in pad.

M n ---
Move cursor by a signed amount and print its line.

T n ---
Type line n and save in PAD.

L ---
List the current screen.

R n ---
Replace line n with the text in PAD.

n ---
Put the followng text on line n.

I n ---
Spread at line n and insert text from PAD.

TOP ---
Position editing cursor at top of screen.

CLEAR n ---
Clear screen n, can be used to select screen n for editing.

FLUSH ---
Write all updated buffers to disc. This has been modified wo cope with an error in the Micropolis CPM disc drivers.

COPY n1 n2 ---
Copy screen n1 to screen n2.

-TEXT Addr 1 count Addr 2 -- boolean
True if strings exactly match.

MATCH cursor-addr bytes-left-till-EOL str-addr str-count
--- tf cursor-advance-till-end-of-matching-text
--- ff bytes-left-till-EOL
Match the string at str-addr with all strings on the cursor line forward from the cursor. The arguments left allow the cursor R# to be updated either to the end of the matching text or to the start of the next line.

1LINE --- f
Scan the cursor line for a match to PAD text. Return flag and update the cursor R# to the end of matching text, or to the start of the next line if no match is found.

FIND ---
Search for a match to the string at PAD, from the cursor position till the end of screen. If no match found issue an error message and reposition the cursor at the top of screen.

DELETE n ---
Delete n characters prior to the cursor.

N ---
Find next occurrence of PAD text.

F ---
Input following text to PAD and search for match from cursor position till end of screen.

B ---

Backup cursor by text in PAD.

X ---

Delete next occurrence of following text.

TILL ---

Delete on cursor line from cursor to end of the following text.

C ---

Spread at cursor and copy the following text into the cursor line.

THE FORTH ASSEMBLER - COMMENTS, CRITICISM and EXTENSIONS

ABSTRACT

The original version of the FORTH 8080 assembler is described and a proposed "improved" version is discussed. Extensions for the implementation of Z80 instructions (those not found in the 8080) are suggested.

INTRODUCTION

The fig-FORTH implementation on the ABC80 is derived from the 8080 version originally coded by John J. Cassady. About two years ago Cassady wrote, in FORTH, an assembler which interprets Intel 8080 mnemonics and lays out the appropriate machine code directly. It is therefore a one pass assembler which loads code directly. Program flow is controlled by the FORTH structured control words BEGIN WHILE UNTIL REPEAT IF ELSE and THEN. Cassady has recently released a newer version of the assembler.

ORIGINAL VERSION

The assembler uses pure 8080 mnemonics but the order of use is reversed, as is typical of the FORTH method of operation. With a conventional assembler the instruction order would be:

instruction destination, source (MOV A,L)

while the FORTH assembler requires

source destination instruction (L A MOV).

For ABC80 users, the question immediately arises, "Isn't there a Z80 assembler for FORTH that would allow us to use Z80 mnemonics and those powerful additional instructions which the 8080 doesn't have?". The answer is:

1 - yes, there are Z80 assemblers for FORTH,
2 - no, you can't use pure Z80 mnemonics with them, and
3 - yes, you can use the additional Z80 instructions with them.

Implementors of Z80 and 6502 assemblers (in FORTH) usually put a comma at the end of the instruction mnemonics to set off the source-destination-instruction sequence from the following source-destination-instruction sequence. That is a minor detail that can easily be implemented in the 8080 assembler, if you prefer it. A more important change from the conventional assembler to the equivalent FORTH assembler, is the description of source and destination. The simple instruction LD can direct information transfer between registers, to register pairs, indirectly to the byte addressed by a register pair and it can use immediate data or the contents of registers or indirectly addressed bytes. This variety of addressing modes requires the use of flags to indicate the addressing mode, while the 8080 mnemonics uses different mnemonic instructions for the different addressing modes (MVI moves immediate data to a register, LXI loads immediate data into a register pair, etc.) Therefore, the use of 8080 mnemonics simplifies the assembler construction, while the use of Z80 mnemonics forces the introduction of additional mnemonics to indicate addressing mode.

A FORTH word is defined, in FORTH, by starting with a colon, followed by the name of the new definition, and then a list of FORTH words which define the function, and concluded with a semicolon. If a word is to be defined with assembler code, the definition is started with the word CODE (instead of a colon), then followed by the name of the new definition, and then a list of assembler mnemonics. The definition must end in a jump to the inner interpreter. The address of the inner interpreter is found in the constant NEXT. The final instructions of a definition would be

```
NEXT JMP C;
```

where the C; replaces the semicolon in a colon definition.

Following the common practice in FORTH, information is passed to and from routines on the stack. A CODE routine could then begin

```
CODE TEST H POP D POP ...
```

The word TEST may be expected to return a value. If this value is found in the HL register pair at the conclusion of the routine, the end of TEST would be

```
... H PUSH NEXT JMP C;
```

If we take a closer look at the source code for the inner interpreter, we will find:

```
DPUSH:    PUSH DE
HPUSH:    PUSH HL
NEXT:     LD   A,(BC) , etc.
```

If we wish to push the HL register pair and then continue through the inner interpreter, we can jump directly to the instruction preceeding NEXT. We can therefore add to Cassady's assembler the following constants:

```
NEXT 1 - CONSTANT HPUSH
NEXT 2 - CONSTANT DPUSH
```

Now we can conclude a CODE definition with

```
HPUSH JMP C;
```

or two values may be put on the stack (from the DE and HL register pairs) with

```
DPUSH JMP C;
```

NEW VERSION

Cassady released a new version of the 8080 assembler in FORTH DIMENSIONS III/6. This version does not include conditional call, conditional jump nor conditional return instructions since these operations are handled automatically by the FORTH structured program flow operators (BEGIN, WHILE, UNTIL, REPEAT, IF, ELSE, THEN), and he has removed AGAIN, since it is not included in FORTH-79, and is generally quite useless. However, he has also removed compiler security (error checking) because it interferred with more advanced assembler techniques. If you write, for example

```
(condition) IF 1 H LXI
            ELSE 0 H LXI
            HPUSH C;
```

then the original assembler version will complain "CONDITIONALS NOT PAIRED" (you forgot to write THEN) but the newer version will not detect the error. Another change introduced in the newer version is a new definition of NEXT. In this version, NEXT is not the address of the inner interpreter, but a jump to the inner interpreter! A routine would conclude with

```
... NEXT C;
```

and PSH1 is a jump to HPUSH and PSH2, a jump to DPUSH.

My personal opinion is

1 - The conditional calls, jumps and returns should be removed from the assembler

vocabulary

2 - I would like to retain error checking until I become more expert in FORTH assembler programming

3 - The new definition of NEXT is not consistant with other FORTH assemblers I have seen, nor is it consistant with the source code, so I find it an unnecessary confusion.

EXTENSIONS

Several assemblers I have seen have replaced the old concluding word C; with the FORTH-79 word END-CODE.

```
: END-CODE  /COMPILE/ C; ;
```

The above definition will allow you to use END-CODE or C; as you prefer. Notice that, since C; is an immediate word, it must be preceeded by /COMPILE/ in this definition.

In order to use the more powerful Z80 instructions not found in the 8080 mnemonics, I would like to define the following words. I will use Z80 mnemonics and my assembler will be a hybrid.

```
ASSEMBLER DEFINITIONS HEX
D9 1MI EXX

: BITAD <BUILDS C, DOES> CB C, C@ SWAP 8* + + C, ;
40 BITAD BIT
80 BITAD RES
C0 BITAD SET
```

These instructions are used in the form:

```
A 7 BIT ( test bit 7 in the accumulator )
M 0 SET ( set bit 0 of the byte pointed to by the )
        ( contents of the HL register pair )
```

```
: 2BYTE <BUILDS C, DOES> ED C, C@ C, ;
67 2BYTE RRD    6F 2BYTE RLD    A0 2BYTE LDI
A1 2BYTE CPII   A2 2BYTE INI    A3 2BYTE OUTI
A8 2BYTE LDD    A9 2BYTE CPD    AA 2BYTE IND
AB 2BYTE OUTD   B0 2BYTE LDIR   B1 2BYTE CPIR
B2 2BYTE INIR   B3 2BYTE OTIR   B8 2BYTE LDDR
B9 2BYTE CPDR   BA 2BYTE INDR   BB 2BYTE OTDR
FORTH DEFINITIONS DECIMAL
```

Put the above definitions onto a free screen on the system diskette, and add the number of the screen plus the instruction LOAD to the loading screen for the assembler (probably screen 42). Note that Z80's CPI has been called CPII in order to avoid conflict with 8080's CPI.

EXAMPLES

```
CODE ODD.PARITY.SEND ( ascii-char --   )
( assuming a 7 bit ASCII character on the stack )
( set the 8th bit, if necessary, so that the 8 bit )
( character will have odd parity, then send it      )
     D POP ( get char. in E register   )
     E A MOV A ANA ( set flags )
     PE IF E 7 SET
     THEN TXD CALL NEXT JMP
END-CODE

CODE ODD.PARITY.RECEIVE ( -- char-2, flag-1 )
( receive an ASCII character and check its parity )
( return the character plus a flag. )
( flag=2 if  reception was interrupted by pressing )
( a key )
( flag=1 if parity is odd )
( flag=0 if parity is even )
     RXD CALL ( char. in E )
     2 H LXI
     65013 LDA A 7 BIT 0= ( key pressed ? )
     IF ( no key pressed ) H DCX
          E A MOV A ANA ( set flags )
          PE IF H DCX THEN
     THEN DPUSH JMP
END-CODE
```

Uppsala, 1982.07.28
Bob

SCR # 66

```
( ASSEMBLER 5                                    )
ASSEMBLER DEFINITIONS HEX
NEXT 1 - CONSTANT HPUSH
NEXT 2 - CONSTANT DPUSH
: END-CODE  /COMPILE/ C; ;
D9 1MI EXX
: BITAD <BUILDS C, DOES> CB C, Cé
  SWAP 8* + + C, ;
40 BITAD BIT
80 BITAD RES
C0 BITAD SET
: 2BYTE <BUILDS C, DOES> ED C, Cé
  C, ;
67 2BYTE RRD     6F 2BYTE RLD
A0 2BYTE LDI     A1 2BYTE CPII
A2 2BYTE INI     A3 2BYTE OUTI
A8 2BYTE LDD     A9 2BYTE CPD
AA 2BYTE IND     AB 2BYTE OUTD
B0 2BYTE LDIR    B1 2BYTE CPIR
FORTH DEFINITIONS DECIMAL
;S
```

ABC80 SYSTEM-DISK   (1982.08.02)

SCR # 67

```
( ASSEMBLER 6                                    )
ASSEMBLER DEFINITIONS HEX
B2 2BYTE INIR     B3 2BYTE OTIR
B8 2BYTE LDDR     B9 2BYTE CPDR
BA 2BYTE INDR     BB 2BYTE OTDR
FORTH DEFINITIONS DECIMAL
;S
CODE ODD.PARITY.SEND ( ascii-1 -- )
     D POP ( char. in E reg )
     E A MOV A ANA ( set flags )
     PE IF E 7 SET
        THEN
     TXD CALL NEXT JMP
END-CODE
CODE ODD.PARITY.RECEIVE
( -- ascii-2, flag-1 )
     RXD CALL ( char. in E )
     2 H LXI 65013 LDA A 7 BIT 0=
     IF H DCX E A MOV A ANA PE
        IF H DCX THEN
     THEN DPUSH JMP END-CODE      ;S
```

ABC80 SYSTEM-DISK   (1982.08.02)

Programvaran i ABC80
Listning med kommentarer

ABC-klubbens Rapport 1 som är en listning av programvaran i ABC 80 har slutsålts och styrelsen har beslutat att göra ett nytryck och hålla samma facila pris som tidigare, 80 kr inkl moms men exkl frakt.

Klubben måste ha skriftliga beställningar och dessa behöver inte vara någon märklig blankett, huvudsaken att det går att tyda namn och adress. Skicka gärna vykort eller märk kuvertet Rapport 1 eller "Disassembler".

ABC-klubben fick tillgång till ett examensarbete som utförts vid institutionen för tillämpad elektronik vid KTH. I detta arbete redovisas:

1. En kommenterad disassemblerlista av programvaran i ABC 80.
2. Adresstabeller där det står var olika satser kompileras och exekveras, samt var funktioner och operatorer exekveras.
3. Flödesscheman av några viktiga rutiner.
4. Ett tillägg som kort beskriver hur DOS-filer ligger på skivor.

Disassemblerlistan innehåller kommentarer rörande kommandorutinerna, satskompilering och uttryckskompilering, fixningsrutinen (som skaffar variabelutrymme och adresser till variabel och radreferenser), den del av filhanteringen som hör till BASIC-tolken, kassettrutinerna, tillbakalistningen från internt format av BASIC-rad, m m. Det som fattas är främst exekvering av funktioner, satser och operatorer (samt DOS-rutinerna).

Arbetet omfattar c:a 320 sidor, listningen är skriven med maskin men kommentarerna är handskrivna. Trycket är kontorsoffset och den tekniska kvaliteten är p g a orginalets utseende inte den högsta men är fullt läsbart.

Med tanke på första tryckningens strykande åtgång är det bäst att Du snarast försäkrar Dig om ett exemplar för framtida studier i det inre av ABC 80. Rapporten kan även vara till hjälp för den som vill försöka komma underfund med programvaran i ABC-800.

RAPPORT 1

DISASSEMLERN för ABC80

Ännu finns exemplar kvar av nytrycket av klubbens Disassembler.
Examensarbetet av Arne Stockman
Beställ genom att skicka vykort eller brev, märkta "RAPPORT 1" eller "DISASSEMBLER".

Levereras mot postförskott 80 Skr + frakt.

SAMLINGSNUMMER 1980 + 1981

inklusive ABC-kassetter under sammanklippning.
Beräknas färdigt för leverans omkring midsommar.
OBS ! Allt eller intet. Vi delar ej på paketet kassetter - samlingsnummer.
Vill ni beställa ??!!
Gör det medelst vykort (vackra !).
Levereras som postförskott = **100 Skr**

Holms Gårds Tryckeri
590 98 EDSBRUK