MPF-IP
FORTH
Manual

Acer ◆®

COPYRIGHT

DISCLAIMER

Preface

MPF-IP FORTH Manual is written for those who wish to learn FORTH with MPF-IP, a production of Multitech Industrial Corporation. Readers may better understand this fourth generation computer language by following instructions stated in the manual. For those who do not have MPF-IP, the manual offers an opportunity to know FORTH.

You can start to practice by inserting a FORTH EPROM on socket U3. Options such as printer and I/O M may be connected to enhance its capability. In addition, an independent and complete system may be set up by adding EPROM WRITER to FORTH.

FORTH combines merits of both high level language and low level language. It is a highly structured language. You may define your own WORD (instructions are called words in FORTH) if necessary. The system provides basic words for arithmetic and logic operations and stack manipulation. However, users themselves may define stronger and more adequate words for specific situation without any restriction.

FORTH uses postfix notation to write programs. Therefore, expressions 5 + 3, 5 * 3 in BASIC are changed to 5 3 + , 5 3 * in FORTH. It is possible for you to encounter a few difficulties in using this notation at the very beginning. But, you can make the best use of its function once you get used to it.

This manual is a helpful guide for FORTH beginners. We hope you enjoy reading it.

TABLE of CONTENTS

1

Introduction

1.1 The Source of FORTH-MPF-IP

The MPF-IP is a convenient instrument to learn the FORTH language. It contains an 8K-bytes EPROM. The EPROM records the FORTH language and can be inserted into the socket U3. The FORTH begins to work by both turning on the machine and pressing CTRL-B. Press the RESET key and turn on the machine again also help initialize.

The programs for FORTH-MPF-IP are based on 8080 programs of FIG-FORTH. The FORTH has functions as an interpreter, a compiler and an editor. It also contains the international FORTH-79 standard commands (Oct. 1980). In addition, we provide other words especially for MPF-IP which will be discussed later.

1.2 Essentials and Options

The essentials for FORTH-MPF-IP are as follows:
      (1) a system unit (4K RAM)
      (2) a FORTH-MPF-IP EPROM
In addition, some other options may be used.
      (1) printer: It prints data output for permanent record.
      (2) I/O M: The expansion memory strengthens the edit function. Its I/O port makes FORTH show a strong control capability.
      (3) EPROM WRITER: A full set of independent applied system is used in accordance with FORTH system.
      (4) SGB & SSB: They are used to produce sounds.

1.3 ASCII Codes in FORTH

The commands in FORTH are composed of a series of characters, separated by spaces. The characters include a full set of ASCII codes, excluding backspace, carriage return, null, and space; and control codes, excluding CTRL-P (used to control printer) and CTRL-G (used to control speaker). The following strings are some examples:

    -FIND   ."   BEGIN   `?   (EMIT)

On MPF-IP's keyboard, ⬅ stands for carriage return, ⬅ stands for backspace, ▭ stands for space. Null is a self-produced code in the FORTH, which can not be seen at the keyboard. Refer to Appendix A for the codes generated by the other keys.
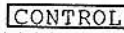
Three basic commands in FORTH [,], and [COMPILE] are replaced by (*, *) and <COMPILE> respectively, as [and] can not be generated by the MPF-IP keyboard.

1.4 Entering FORTH and Exiting FORTH-MPF-IP

(A) Entering FORTH-MPF-IP
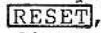
There are two ways to enter the system, if the screen displays *****MPF-I-PLUS***** or ⅄ after you turn on the machine.

1. Press CONTROL and B simultaneously. The FIP (Fluorescent Indicator Panel) will black out for a few seconds, and the screen displays ****FORTH-MPF-IP****. It enters the system and waits for commands. It is the cold start which clears the commands outside the system, and makes an initialization. Memory in the pseudo-disk track becomes 0.

2. Press CONTROL and C simultaneously. The screen displays ****FORTH-MPF-IP****, and waits for you to input commands. It is the warm start that is generally used in reentering the system and keeping the established dictionary commands before exiting FORTH. Initialization is only made for few variables. No clearing is implemented on the dictionary and the pseudo disk memory.

(B) Exiting FORTH-MPF-IP

There are also two ways to exit the system.

1. Press the RESET key.
Whenever you press RESET, MPF-IP returns to the initial status. The screen displays *****MPF-I-PLUS*****.

2. Input MON ⬅ .
The MPF-IP is now under the control of the monitor. The screen displays the monitor's prompt > .

1.5 An Overview of the FORTH Language

You may follow the steps listed below to enter into the FORTH-MPF-IP system.

a. Be sure to turn off the power, and then insert FORTH-MPF-IP EPROM into the socket U3.

b. Connect all options.

c. Turn on the power, and the screen displays *****MPF-I-PLUS*****.

d. Press CONTROL and B simultaneously, and the screen displays ****FORTH-MPF-IP****.

e. Press ⬅ , and the screen displays OK∧, indicating it is in the FORTH system.

1.5.1 "Word" and "Dictionary" in FORTH

Every kind of computer language has its own notation to indicate what will be executed, such as instructions LD A,B and ADD A,C in the Assembly Language; and statements For I=0 TO 255 and PRINT A+B in BASIC. In FORTH, we use "word" to execute a command.

A "Word" is composed of one or more than one characters. It is the code for an event or a procedure. In FORTH-MPF-IP, each "word" is related to an event. For example, the word * multiplies two numbers in the memory and saves the result back to the memory. The word EMIT takes the number in the memory as an ASCII code, and prints or displays its corresponding ASCII character. Primitive "WORD"s supplied in FORTH-MPF-IP can be illustrated by pressing VLIST ⬅ . Consult the following printout:

VLIST	
F009	TASK
3AF4	MON
3AE8	EI
3ADD	DI
3AD1	IM0
3AC4	IM1
3AB7	IM2
3AA1	NEXT
3A88	END-CODE
3A73	CODE
3A60	TREAD
3A2E	TWRITE
39D4	.S
39C7	0>
39A6	D<
398F	DEPTH
394D	ROLL
393A	J
3934	EXIT
391D	2OVER
3909	2SWAP
38DA	.CPU
38CF	INTVECT
38C1	INTFLAG
38A9	;INT
386D	INDEX
3818	LIST
37D9	VLIST
37C9	U.
37BC	?
37B0	.
37A2	D.
3791	.R
376E	D.R
3756	#S
372B	#
3715	SIGN
36FC	#>
36ED	<#
36D0	SPACES
36BF	WHILE
369D	ELSE
3686	IF
366D	REPEAT
3656	AGAIN
3648	END
3634	UNTIL

361E	+LOOP
3608	LOOP
35F5	DO
35EA	THEN
35CF	ENDIF
35BD	BEGIN
35AB	BACK
356E	FORGET
3555	'
3535	➤
3503	LOAD
34C2	DUMP
34B7	FLUSH
349B	R/W
347D	BLOCK
346F	BUFFER
3458	EMPTY-BUFFERS
3446	UPDATE
343B	+BUF
3430	PREV
3425	USE
3410	P!
33FB	P@
33C3	MESSAGE
33AF	.LINE
338D	<LINE>
336E	M/MOD
335C	*/
334B	*/MOD
333B	MOD
332B	/
331B	/MOD
330C	*
32E6	M/
32CB	M*
32B6	MAX
32A0	MIN
3292	DABS
3283	ABS
3271	D+-
325F	+-
324B	S->D
3220	COLD
31DD	WARM
319C	ABORT
316F	QUIT
315C	<
314C	DEFINITIONS

3134	EDITOR		2A62	CFA
3121	FORTH		2A52	LFA
30E7	VOCABULARY		2A42	LATEST
30CE	IMMEDIATE		2A1D	TRAVERSE
3080	INTERPRET		2A06	?DUP
3050	?STACK		29F7	SPACE
3035	DLITERAL		29E1	PICK
3018	LITERAL		29D2	ROT
2FFC	<COMPILE>		29C4	>
2F76	ID.		29A4	U<
2F41	ERROR		2982	<
2F33	<ABORT>		2976	=
2F09	-FIND		2968	-
2EB1	NUMBER		2951	C,
2E66	CONVERT		2940	,
2E14	WORD		2934	ALLOT
2E01	PAD		2924	HERE
2DE9	HOLD		2915	2-
2DDA	BLANKS		2908	1-
2DC9	ERASE		28FB	2+
2DAB	FILL		28EE	1+
2D6C			28A4	CALL
2D54	QUERY		2899	RL
2CDA	EXPECT		2890	RH
2CAB	."		2887	RE
2C92	<.">		287E	RD
2C5F	-TRAILING		2875	RC
2C31	TYPE		286C	RB
2C1E	COUNT		2863	RF
2BF7	DOES>		285A	RA
2BE7	CREATE		2851	RHL'
2BD0	;CODE		2846	RDE'
2BBA	<;CODE>		283B	RBC'
2BA4	DECIMAL		2830	RAF'
2B8E	HEX		2825	RIY
2B7C	SMUDGE		281B	RIX
2B67	*>		2811	RHL
2B58	<*		2807	RDE
2B41	COMPILE		27FD	RBC
2B27	?LOADING		27F3	RAF
2B0A	?CSP		27E9	UABORT
2AF7	?PAIRS		27DC	UR/W
2AE0	?EXEC		27D1	UCR
2AC8	?COMP		27C7	UEMIT
2AAE	?ERROR		27BB	UKEY
2A9B	!CSP		27B0	U?TERMINAL
2A86	PFA		279F	UB/SCR
2A70	NFA		2792	UB/BUF

2785	ULIMIT		24DB	BOUNDS
2778	UFIRST		24C9	2DUP
276B	UC/L		24BB	DUP
2760	HLD		24AE	SWAP
2756	R#		249F	2DROP
274D	CSP		2491	DROP
2743	FLD		2482	OVER
2739	DPL		2464	DNEGATE
272F	BASE		244D	NEGATE
2724	STATE		2429	D+
2718	CURRENT		241C	+
270A	CONTEXT		240B	0<
26FC	OFFSET		2404	NOT
26EF	SCR		23F0	0=
26E5	OUT		23E9	R@
26DB	>IN		23D4	R>
26D1	BLK		23BE	>R
26C7	VOC-LINK		23AA	LEAVE
26B8	DP		2393	;S
26AF	FENCE		237B	RP!
26A3	WARNING		236D	RP@
2695	WIDTH		2356	SP!
2689	TIB		2347	SP@
267F	R0		2334	XOR
2676	S0		2321	OR
2669	B/SCR		230F	AND
2659	B/BUF		22C2	U/MOD
2649	LIMIT		228C	U*
2639	FIRST		2276	CMOVE
2629	C/L		2264	CR
261F	BL		2255	?TERMINAL
2616	3		223F	KEY
260E	2		2229	EMIT
2606	1		21DE	ENCLOSE
25FE	0		2194	<FIND>
25E8	USER		2166	DIGIT
25D4	VARIABLE		2152	I
25B6	CONSTANT		2136	<DO>
259D	;		212A	<+LOOP>
2575	:		20F1	<LOOP>
2562	2!		20DC	0BRANCH
2555	C!		20C4	BRANCH
2546	!		20B5	EXECUTE
252C	2@		20A0	LIT
251E	C@		OK	
250F	@			
2501	TOGGLE			
24EA	+!			

Sometimes, one word alone is not enough to complete an execution. Several words are then composed to form a program. The program is regarded as a new word, which may be used as a unit to form a more complex execution. This is exactly the process to write a program in FORTH: to put several words together to complete the purpose.

Examples:

The unit price for a fountain pen is US$5,00. The total price for n fountain pens is :

TOTAL PRICE = n * UNIT PRICE

We can define a word 5* which combines 5 and *.

Input   : 5* 5 * ; [⏎]
Display   : 5* 5 * ; OK∧

The word . (DOT) is used to print the result.

The following table lists words related to number output:

Words	Stack Manipulation and Action
D.R	(d n ---) Print double number d in an n-character field, right justified.
D.	(d ---) Print double number d and leave a space to its right.
U.	(un ---) Print unsigned integer number un and leave a space to its right.
.R	(n1 n2 ---) Print signed integer number n1 in an n-character field, right justified.
.	(n ---) Print signed integer number n and leave a space to its right.

?	(addr ---) Print signed integer number in address addr and leave a space to its right.

We can define a new word FOUNTAIN-PEN both to print the result and to count the total price.

Input   : FOUNTAIN-PEN 5* . ; [⏎]
Display   : FOUNTAIN-PEN 5* . ; OK∧

Example: the total price for 7 fountain pens

Input   7 FOUNTAIN-PEN [⏎]
Display   7 FOUNTAIN-PEN 35 OK∧

Example: the total price for 9 fountain pens

Input   9 FOUNTAIN-PEN [⏎]
Display   9 FOUNTAIN-PEN 45 OK∧

All words in the system are stored in the dictionary. It is a one-directional serial table. Every word is different in length. However, they are defined completely or contain all necessary data for execution. The dictionary may be expanded toward the higher end of the memory. The dictionary may also be divided into several vocabularies. Each vocabulary contains related words.

1.5.2 Stacks in FORTH

The system uses two stacks to save temporarily data and addresses. One is the Data Stack, the other is the Return Stack. The stack generally refers to the Data Stack unless otherwise specified.

The stack is a certain area in memory used to save and retrieve the data. We may call it a last-in first-out memory. If you input 1 3 5 7 [⏎], the display of 1 3 5 7 OK∧ means all words have been executed. Four numbers have been stored in the stack. The first-in number 1 is placed at the bottom of the stack. And the last-in number 7 is at the top of the stack.

The following is a conceptual diagram of stack in memory. The first-in number 1 is at the highest end in

memory and numbers input later are lined up through the
lower end.

```
 |top in memory|      |top in memory|      |top in memory|
→|      1       |      |      1       |      |      1       |
 |              |     →|      3       |      |      3       |
 |              |      |              |      |      5       |
 |              |      |              |     →|      7       |
 |  bottom in   |      |  bottom in   |      |  bottom in   |
 |    memory    |      |    memory    |      |    memory    |

 |pointer|            |pointer|            |pointer|
 |_____|            |_____|            |_____|
```
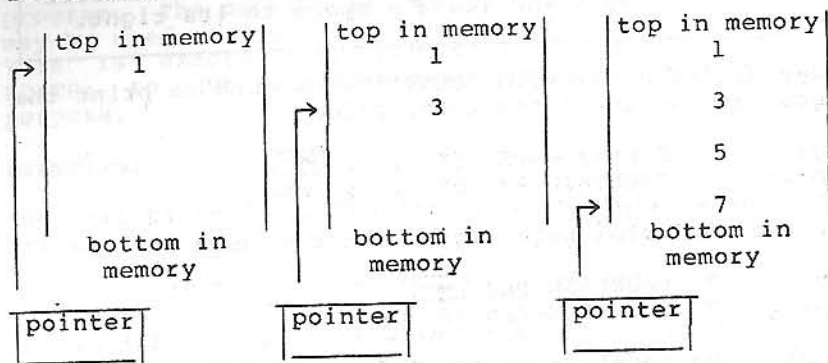
The following is a conceptual illustration of the
stack. That the stack extends upward is the same as
that plates are piled in a restaurant. The first-set
plate is at the bottom, so that the last-set ones are
taken first. When you add a new number on the stack,
it is pushed on the top of the stack. When you take one
off the stack, you pop the number away.

```
                                         →|      7       |
                                          |      5       |
                     →|      3       |     |      3       |
  →|      1       |    |      1       |     |      1       |
   | bottom of    |    | bottom of    |     | bottom of    |
   | the stack    |    | the stack    |     | the stack    |

   |pointer|          |pointer|           |pointer|
   |_____|          |_____|           |_____|
```
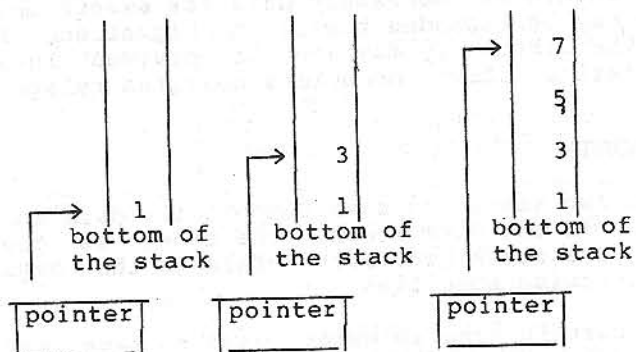
There are four numbers on the stack. Use the word
(Dot) to print the numbers.

Input   . . . . [←┘] (Do not forget to leave a
                          space between the dots.)
Display   . . . . 7 5 3 1 OK

Data used in FORTH words are mostly taken from the data
stack. Data are placed onto the stack in any of the

three ways listed below:
   1) words keyed in from the keyboard;
   2) words in the source program;
   3) values resulted from execution of words.

The return stack stores the address for the word to be
executed next. Its function is like that of the stack
in a general computer system, that is, to save the
address of the next instruction in the main program
when it calls subroutine. The return stack is mainly
used to control calls among words and return action.
However, under specific condition, the return stack
does additional work, such as:

a. the index and the limit used in the DO...LOOP;

b. some numbers which are not easy to manipulate on the
   temporary data stack.

The return stack is closely connected to the system's
operation. Be sure to use it carefully . Any misuse
may cause an irrevocable result to the entire system.

1.5.3 Postfix Notation

Arithmetic operations for most computer languages are
as follows: 5+3 which is familiar to most people. FORTH
uses postfix notation, and the expression in the above
will be: 5 3 +. The reason for the adoption of this
peculiar notation in FORTH is that all words take
necessary data from stack and put result onto stack.
Interactions among words are greatly reduced in this
way. Words of different levels may exchange data
provided by the stack in a rather complex operation. In
the operation 5 3 + , 5 is placed on the stack
first, followed by 3. Addition operator + takes out and
adds 5 and 3, and saves the result 8 on the stack.
After the operation, 5 and 3 are removed from the
stack.
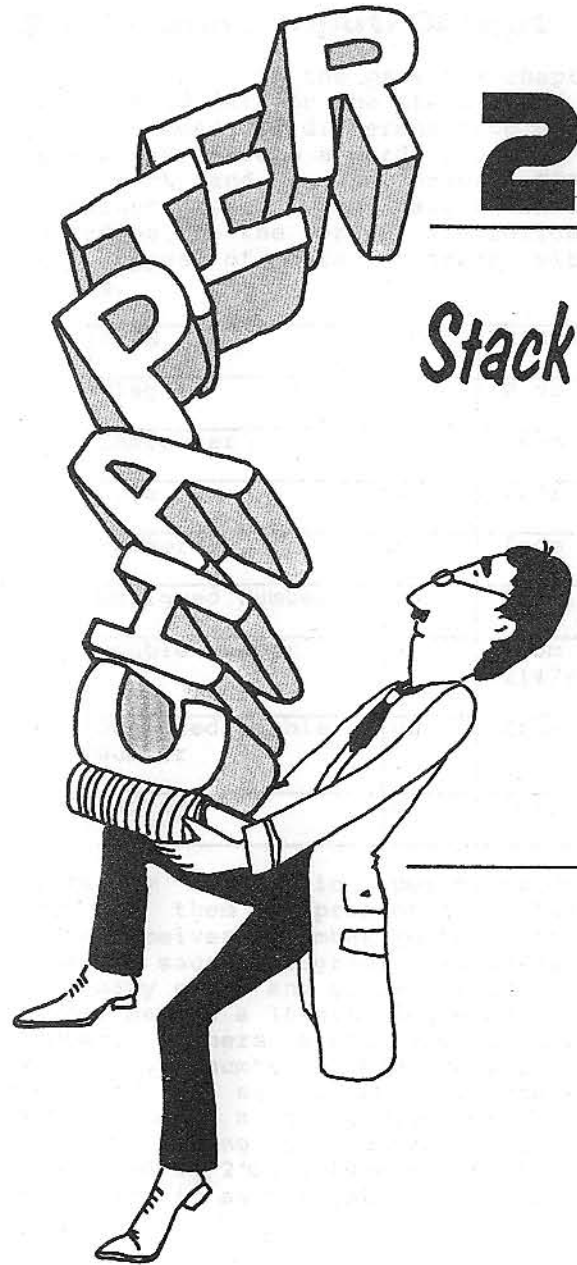
CHAPTER 2

Stack Manipulation and Arithmetic Operations

2.1 Number Input/Output

We mentioned in the previous chapter that many FORTH words need data on the stack, and the number of data items needed is different from one word to another. Before you execute a word, you have to know the data on the stack and in what order. FORTH data may have different types, you have to choose the right one according to the words. The following table lists the main types of data, together with their codes and ranges.

TYPE	CODE	RANGE
Flag	f	0 or non-0
Character	c	from 0 to 127
Byte	b	from 0 to 255
Number	n	from -32768 to 32767
Unsigned Number	un	from 0 to 65535
Double Number	d	from -2147483648 to 2147483647
Unsigned Double Number	ud	from 0 to 4294967295
Address	a	from 0 to 65535

Primarily arithmetic operations deal with integers. Most of them are presented as 16-bit numbers. When FORTH receives a number (either from the keyboard, or from the source program), it transfers the number into a binary one, and pushes it on the stack. The input number may be a 16-bit single number or a 32-bit double number. Numbers with a decimal point will be regarded as a double number by the decoder, otherwise they are regarded as single numbers. The word.(dot)removes the single number at the topmost of the stack and change it into a string and display it. Single number is presented by 2's complement. If it is larger than 32767. We regard it as a negative number. For example:

```
Input      5  .  [←]
Display    5  .  5  OKʌ

Input      -300  .  [←]     (Press the shift key and "I"
                            letter  simultaneously  to
                            get a minus sign.)
Display    -300  .  -300  OKʌ

Input      32769  .  [←]
Display    32769  .  -32767  OKʌ
```

In the last example, 32769 exceeds 32767, thus the system regards it as a negative one. You may avoid this as shown below:

```
Input      32769  0  D.  [←]
Display    32769  0  D.  32769  OKʌ
```

That is, you may push a 0 above 32769 on the stack and make it a double number. 'D. is used to print the double number stored on the top of the stack.

If there is a decimal point, FORTH regards it as a double number. For example, 32769 is regarded as a 32-bit double number. FORTH only recognizes the decimal point and its place, but the decimal point does not affect the conversion. Try the following example:

```
Input      32769.  D.  [←]
Display    32769.  D.  32769  OKʌ

Input      327.69  D.  [←]
Display    327.69  D.  32769  OKʌ

Input      3.2769  D.  [←]
Display    3.2769  D.  32769  OKʌ

Input      3.27.69  D.  [←]
Display    3.27.69  D.  32769  OKʌ
```

The number of digits following the decimal point are recorded in the system variable DPL. If you want to identify it for related numeric operations, you may use DPL.

As described earlier, you may place a 0 above a single number to get a double number. Similarly, a double number may be divided into two single numbers. If a double number is divided into a higher 16-bit and a lower 16-bit, the higher one will be on the top of the stack.
Such as:

```
Input      6553.6  .  .  [←]
Display    6553.6  .  .  1  0  OKʌ

Input      300.  .  .  [←]
Display    300  .  .  0  300  OKʌ
```

2.2 Words for Arithmetic Operations

The following table lists the arithmetic words used in FORTH, including single number words, double number words, and mixed operation words.

Words	Stack Manipulation and Action
+	(n1 n2 - n3) n1 + n2. Leave the sum n3 on the stack.
-	(n1 n2 - n3) n1 - n2. Leave the difference n3 on the stack.
1 +	(n - n + 1)
1 -	(n - n - 1)
2 +	(n - n + 2)
2 -	(n - n - 2)
*	(n1 n2 - n3)
/	(n1 n2 - n3) n1 is divided by n2. Leave the quotient n3 on the stack.
/MOD	(n1 n2 - n3 n4) n1 is divided by n2. Leave the remainder n3 and the quotient n4 on the stack.

*/	(n1 n2 n3 - n4) n1 multiplies n2 and then the product is divided by n3. Leave the quotient n4 on the stack.
*/MOD	(n1 n2 n3 - n4 n5) It is the same as */. Leave the remainder n4 and the quotient n5 on the stack.
U*	(un1 un2 - ud) Multiply two unsigned numbers un1 and un2. Leave the product (double number) ud on the stack.
U/MOD	(ud un1 - un2 un3) The double number ud is divided by un1. Leave the remainder un2 and the quotient un3 on the stack.
MAX	(n1 n2 - n3) Leave the larger one of n1 and n2 on the stack.
MIN	(n1 n2 - n3) Leave the smaller one of n1 and n2 on the stack.
ABS	(n1 - n2) Leave n1's absolute value on the stack.
NEGATE	(n1 - n2) Change the sign of the topmost value on the stack.
AND	(n1 n2 - n3) Leave the resultant value from logical AND.
OR	(n1 n2 - n3) Leave the resultant value from logical OR.
XOR	(n1 n2 - n3) Leave the resultant value from logical Exclusive-OR.

D +	(d1 d2 - d3) Add double numbers d1 and d2. Leave the sum d3 on the stack.
DNEGATE	(d1 - d2) Change the sign of topmost double number on the stack.
DABS	(d1 - d2) Leave the absolute value of the topmost double number on the stack.
M*	(n1 n2 - d) Multiply two single numbers n1 and n2. Leave the product (double number) d on the stack.
M/	(d n1 - n2 n3) The double number d n1 is divided by a single number n1. Leave the remainder n2 and the quotient n3 on the stack. The sign of the quotient is the same as that of the dividend d.
M/MOD	(ud1 un2 - un3 ud4) The unsigned double number ud1 is divided by the unsigned number u2. Leave the remainder u3 and the unsigned double number quotient ud4.
MOD	(n1 n2 - n3) n1 is divided by n2. Leave the remainder n3 on the stack. The sign of n3 is the same as that of n1.

Example: Find the product of 35*7

Input   35 7 * . ⏎
Display 35 7 * . 245 OKʌ.

Example: Find the quotient of 31/4

Input   31 4 / . ⏎
Display 31 4 / . 7 OKʌ

You may use the word MOD to display the remainder.

```
Input     31  4  MOD  .  [←]
Display   31  4  MOD  .  3  OK^
```

The word */ is provided in FORTH for calculation of ratio. The following example can be used to calculate percentage.

```
Input     :  %  100  */  ;  [←]
Display   :  %  100  */  ;  OK^
```

```
Input     675  15  %  .  [←]
Display   675  15  %  .  101  OK^
```

So far, we have used the number base of 10 (decimal) in the examples. However, the system variable BASE may convert the base number.

We have defined the following words in the FORTH-MPF-IP.

```
:  HEX  16  BASE  !  ;
:  DECIMAL  10  BASE  !  ;
```

```
Input     16  .  [←]
Display   16  .  16  OK^
```

In the following example the base number is changed.

```
Input     16  HEX  .  [←]
Display   16  HEX  .  10  OK^
```

Once you have converted a base number, the system will keep it as changed until you set a new base number or turn on the power again.

```
Input     30  DECIMAL  .  [←]
Display   30  DECIMAL  .  48  OK^
```

```
Input     255  HEX  .  [←]
Display   255  HEX  .  FF  OK^
```

If you wish to use the base 8, you may define a word as follows:

```
:  OCTAL  8  BASE  !  ;
```

2.3 Stack Manipulation

FORTH is a well-designed, versatile and effective language. It always input/output numbers in last-in first-out order. FORTH also provides a set of useful words for stack manipulation, so that you may search a specific number from a certain place in stack. See the following table.

Words	Stack Manipulation and Action
DUP (n - n n)	Copy the topmost value on the stack.
DROP (n -)	Remove the topmost value on the stack.
SWAP (n1 n2 - n2 n1)	Change top two values on the stack.
OVER (n1 n2 - n1 n2 n1)	Copy the second value of the stack.
ROT (n1 n2 n3 - n2 n3 n1)	Rotate top three values on the stack. Bring the third one to top.
?DUP (n - n(n))	Copy the topmost value if it is non-zero.
PICK (n1 - n2)	Copy the n1th value of the stack.
ROLL (n -)	Bring the nth value of the stack to the top.
DEPTH (- n)	Place the numbers of current value on the stack.
2SWAP (d1 d2 - d2 d1)	Change two double numbers on the stack.
2DUP (d - d d)	Copy the topmost double number on the stack.
2DROP (d -)	Remove the topmost double number on the stack.

2OVER (d1 d2 — d1 d2 d1)	Copy the second double number on the stack.
.S (—)	Print the contents of the stack without altering or removing the numbers from the stack.

Before you set out to learn how to use the words concerning the stack manipulation, you must understand how the word .S works. In short, the word .S will let you observe changes in the stack. Try the following example:

```
Input      1   2   3   ⌫
Display    1   2   3   OKʌ
```

```
Input      .S  ⌫
Display    1   2   3   OKʌ
```

Execute .S does not change the data in the stack at all. In contrast, try the following:

```
Input      .   .   .   ⌫
Display    .   .   .   3   2   1   OKʌ
```

The first . takes the topmost number 3. The second and the third take 2 and 1 respectively. No data remained in the stack after execution of the word . (dot).

You may practice the stack manipulation with related words and use .S to examine its current status.

```
Input      1   2   3   DUP  ⌫
Display    1   2   3   DUP  OKʌ
```

```
Input      .S  ⌫
Display    1   2   3   3   OKʌ
```

```
Input      DROP  .S  ⌫
Display    1   2   3   OKʌ
```

```
Input      SWAP  .S  ⌫
Display    1   3   2   OKʌ
```

```
Input      OVER  .S  ⌫
Display    1   3   2   3   OKʌ
```

```
Input      ROT  .S  ⌫
Display    1   2   3   3   OKʌ
```

n PICK allows you to copy nth number and place it on the top of the stack. Continue the last example again:

```
Input      3   PICK  .S  ⌫
Display    1   2   3   3   2   OKʌ
```

1 PICK has the same result as DUP. 2 PICK has the same result as OVER.

n ROLL allows you to move the nth number on top of the stack. 3 ROLL has the same result as ROT. 2 ROLL has the same result as SWAP.

```
Input      4   ROLL  .S  ⌫
Display    1   3   3   2   2   OKʌ
```

CHAPTER 3
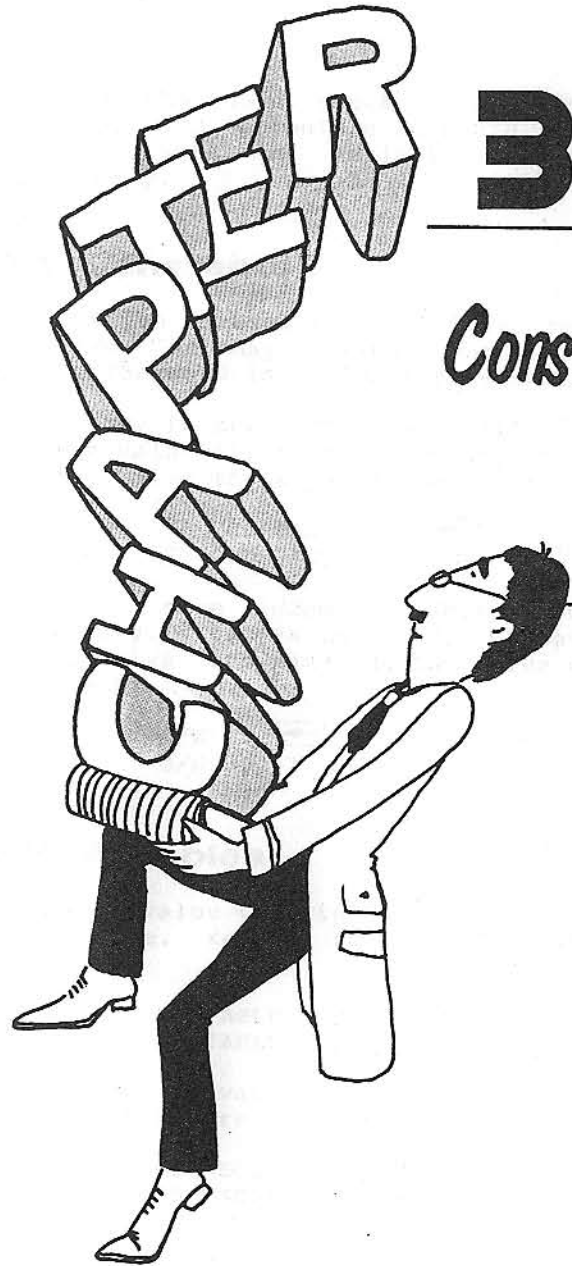
Constants, Varibles and Arrays

We use the data stack to save the information to transfer and manipulate in FORTH. It is necessary to set constants and variables if some data is used frequently.

3.1 Constants

If a value is used frequently and related to a special function, we may define a word with a name for the value. The word is called a constant.

It is easy to set a constant. All you have to do is to input a value first, and then key in the word CONSTANT. Finally, give the word a name.

```
Input      7   CONSTANT  D/W  ⬅
Display    7   CONSTANT  D/W  OK▴
```

Now, we have added a constant word D/W to the dictionary. Its value is 7. Whenever you need the value, give its name and the value will be placed on the stack.

```
Input      D/W  .  ⬅
Display    D/W  .  7  OK▴
```

3.2 Variables

We call a value that is changed frequently in a program a variable. Key in the word VARIABLE, and then its name.

```
Input      VARIABLE  SCORE  ⬅
Display    VARIABLE  SCORE  OK▴
```

The value of a variable is undefinite. You should give it a value before using it.

```
Input      60  SCORE  !  ⬅
Display    60  SCORE  !  OK▴
```

3.3 The Usage of Constants and Variables

The following table lists words for memory read/write.

Words	Stack Manipulation and Action
@	(a - n) Push the value n at address a on the stack.
!	(n a -) Save the value n in address a.
C@	(a - b) Push the byte b at address a on the stack.
C!	(b a -) Save the byte b in address a.
2@	(a - d) Push the double number d at address a on the stack.
2!	(d a -) Save the double number d in address a.
+!	(n a -) Add n to the number at address a.
?	(a -) Fetch the value at address a and print it.

When you point out the name of a variable, its address is pushed on the stack. You should use @ to take its value out.

Input   SCORE @ . ⏎
Display   SCORE @ . 60 OK⌄

The word ? is composed of two words @ and . (dot).

Input   SCORE ? ⏎
Display   SCORE ? 60 OK⌄

The word +! is derived from the word !

Input   20 SCORE +! ⏎
Display   20 SCORE +! OK

We have added 20 to the original number in the SCORE and save the result back. The number is changed from 60 to 80.

Input   SCORE ? ⏎
Display   SCORE ? 80 OK

The word ! makes things easy for you to change the value of a variable. It also helps you change the value of a constant once you know its address. The word ' (tick) may find out the address of a word.

Input   ' D/W . ⏎
Display   ' D/W . -4077 OK⌄

To change the value of a constant :

Input   5 ' D/W ! ⏎
Display   5 ' D/W ! OK⌄

Input   D/W . ⏎
Display   D/W . 5 OK⌄

3.4 Arrays

The parameter field address is saved on the stack upon the execution of a variable defined with VARIABLE. We may enlarge the parameter field for more numbers and bytes, which become arrays. The main purpose is to save the memory space.

Suppose we would like to build an array with 20 bytes:

      VARIABLE DATA   18 ALLOT
      DATA   20 ERASE

18 ALLOT means that we add 18 bytes, storage to the reserved 2 bytes in the parameter field. As illustrated below:

VARIABLE DATA	2 BYTE	18 (ALLOT) THE REST BYTES ...	

→

   direction for dictionary extension   ← next word

DATA 20 ERASE is to clear the 20 bytes in the parameter field into zeroes.

The following words help you to retrieve the data in the array.

DATA @	Fetch the value of the first number.
DATA 2+ @	Fetch the value of the second number.
....	and so on ....

The following words help you to save the value in this array.

10 DATA !	Save the value of the first number.
20 DATA 2+ !	Save the value of the second number.
....	....

CHAPTER 4

Dictionary, Vocabulary and Memory Map

4.1 Memory Map

The following chart is the MPF-IP's 64 K memory map.

```
FFFF ┌──────────────────────┐                    ┌──────────────────┐ EFFF
     │     SYSTEM USE       │                    │    EPB RAM       │
FEB? ├──────────────────────┤                    │                  │ D800
     │ FORTH USER VARIABLE  │──UP                ├──────────────────┤ D7FF
FE00 ├──────────────────────┤──R0                │   I/O M RAM      │
     │                      │                    │                  │ C000
     │ FORTH RETURN STACK   │                    ├──────────────────┤ BFFF
     ├──────────────────────┤                    │   I/O M ROM      │
     │ TERMINAL INPUT BUFFER│                    │                  │ B000
FD80 ├──────────────────────┤──TIB               └──────────────────┘
     │                      │──S0
     │  FORTH DATA STACK    │
FD00 ├──────────────────────┤
     │ FORTH USER DICTIONARY│
F00B ├──────────────────────┤──DP
     │                      │
     │       TASK           │
F000 ├──────────────────────┤                    ┌──────────────────┐ 7FFF
     │                      │                    │                  │
     │  OPTION              │                    │      PRT         │
     │  EXTENSION           │                    │                  │ 6000
     │  MEMORY              │                    ├──────────────────┤ 5FFF
     │                      │                    │      SSB         │
3FFF ├──────────────────────┤                    │                  │ 5000
     │                      │                    │                  │
     │   FORTH-MPF-IP       │                    │                  │
2000 ├──────────────────────┤                    └──────────────────┘ 4000
1FFF │                      │
     │  MPF-IP              │
     │  MONITOR             │
0    └──────────────────────┘
```
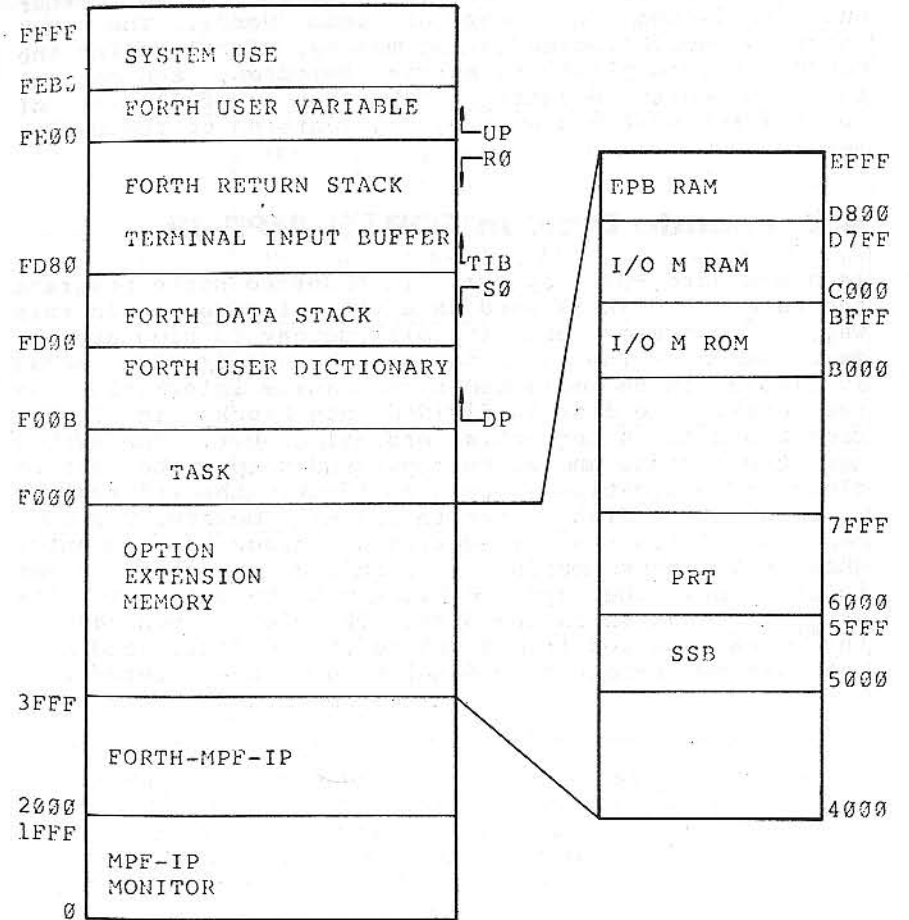
FORTH-MPF-IP uses a memory space of 8k bytes, from $2000 to $3FFF. The system will insert the word TASK at F000 after the machine is turned on or the execution of the cold start (using the word COLD). The user's dictionary grows upward from F00B. For more details about TASK, please see Chapter 9. $FE00 through $FEB0 is allocated for user variables, consult appendix D.

The extension memory ranges from $4000 to $EFFF. You can insert different kinds of additional option boards when necessary. The printer enables you to better understand the functions of some words. The I/O M extension board increases the memory, facilitating the FORTH to accomplish its editing features, and control the I/O ports directly. EPB makes a full set of application system possible, in addition to increasing memory.

4.2 Pseudo Disk in FORTH-MPF-IP

In a standard FORTH system, in order to store programs and data the disk is used as a virtual memory. In this way, the system uses the disk memory to simulate the main memory. The user may use the read/write words available in the main memory to manage information on the disk. The disk is divided into blocks in FORTH. Each block has a sequential ordinal number. The system use the ordinal number to input and output the entire block of information. When you input, the information is read into a disk buffer in the main memory. The user can then fetch the information or change the contents. When this disk buffer is required to store other information, the updated block will be output to its original location in the disk. Therefore, you can get the data required from anywhere on the disk, and need not worry about the details of the read/write operations. The following table lists the words for disk memory.

Words	Stack Manipulation and Action
BLOCK	(n - a) Load the nth block data to a disk buffer, and place the start address on the stack.
BUFFER	(n - a) Allocate a buffer to store the new data of the nth block. Place the buffer address on the stack.
UPDATE	(-) Mark the updated data in the last used buffer.
SAVE BUFFERS	(-) Save the updated buffer data back to the disk.
EMPTY BUFFERS	(-) Clear all the data in buffer, thus avoid being saved back in the disk.
LIST	(n -) Load the nth block character to a buffer and print it.
LOAD	(n -) Load the nth block character and compile or execute.
SCR	(- a) The system variable containing current block number.

FORTH-MPF-IP does not have a real disk. Therefore, a part of memory (28k bytes) is used as a pseudo disk. The length for each pseudo disk is limited to 512 bytes. In practice, many disk commands can not be used without proper modification.

Memory $8000 to $EFFF is divided into 56 blocks. The ordinal number is from 0 to 55. Usually, an extension

memory is required (EPB or I/O M) for the use of pseudo disk memory.

The word BLOCK is defined as follows:

```
: BLOCK 56 MOD OFFSET @ + 512 * FIRST + ;
```

56 MOD limits the block numbers to the range 0 through 55. OFFSET is a user variable. Its initial value is 0. FIRST is a constant used to save the starting address of the pseudo disk. Its initial value is $8000. FIRST + enables you to obtain the address of the first byte of the block.

The user may change the values of OFFSET and FIRST to adjust the location of pseudo disk buffer in coordination with the real memory address of the system.

If you have only the system unit of MPF-IP available, then the RAM covers $F000 to $FFFF, which does not fall within the range of the pseudo disk. In this case, you may change the value of FIRST, so that the pseudo disk start with address you need.

```
Input     HEX  [⬅]
Display   HEX  OKʌ
```

```
Input     0  BLOCK  U.  [⬅]
Display   0  BLOCK  U. 8000 OKʌ
```

```
Input     F200  UFIRST  !  [⬅]
Display   F200  UFIRST  !  OKʌ
```

UFIRST is the user variable for FIRST. Save the value into UFIRST and you can fetch the value from UFIRST upon execution of FIRST.

```
Input     0  BLOCK  U.  [⬅]
Display   0  BLOCK  U.  F200  OK
```

The dictionary grows upward from $F000. The data stack goes down from $FD80. The pseudo disk is between the two. Avoid any overlap, otherwise the system's operation may be affected.

IF I/O M board is implemented and its RAM address is from $C000 to $D7FF, then the ordinal number of pseudo disk blocks will range from $20 (hex) to $2C , you may use these ordinal numbers directly, or change OFFSET so that $C000 becomes a pseudo disk block with an ordinal number of 0. You may do it with FIRST as well.

```
Input     COLD  [⬅]
Display   ****FORTH-MPF-IP****
```

```
Input     HEX  [⬅]
Display   HEX  OKʌ
```

```
Input     20  BLOCK  U.  [⬅]
Display   20  BLOCK  U.  C000  OKʌ
```

```
Input     20  OFFSET  !  [⬅]
Display   20  OFFSET  !  OKʌ
```

```
Input     0  BLOCK  U.  [⬅]
Display   0  BLOCK  U.  C000  OKʌ
```

Among the disk commands of FORTH-MPF-IP, the word BLOCK place the start address of the corresponding block onto the stack. BUFFER works the same way BLOCK does, while EMPTY-BUFFERS clears the pseudo disk memory (ranging from $8000 through $EFFF). The remaining words, such as UPDATE, +BUF, FLUSH and R/W has no effect.

4.3 Print the Message

The word . (dot) is used to print a number on the stack. It is necessary to use another word to print a message.

```
Input     CR ." I AM MPF-IP " [⬅]
Display   I AM MPF-IP OKʌ
```

Separated from the following messages by a space, ." (dot-quote) is used to print the message, until " (delimiter) is encountered.

4.4 Define a New Word

The FORTH system allows you to define your own words. These words work the same way as those primitive words supplied by the language. The names of the user-defined word can contain up to 31 characters. All ASCII characters can be used, except space, back-space, Null and CR.

A new word is defined as follows: start with a colon (:), which is followed by a space, and then the name of a new word, followed again by another space, after that is the events to be executed, finally a semicolon (;), which indicates the end of the new word.

```
Input     :  TEST  3  *  .  ;  ⏎
Display   :  TEST  3  *  .  ;  OKʌ

Input     8  TEST  ⏎
Display   8  TEST  24  OKʌ

Input     5  TEST  ⏎
Display   5  TEST  15  OKʌ
```

4.5 Structure of FORTH Words

All FORTH words has the same structure, whether it is a high-level word, a low-level word, a constant, or a variable. The following table illustrates the structure of a high-level word.

84	1	Ø	Ø	Ø	Ø	1	Ø	Ø	
54				T					NAME FIELD
45				E					
53				S					
D4				T					
Ø									LINK FIELD
FØ									
87									CODE FIELD
25									
14									
26	3 ADDRESS								
A									PARAMETER FIELD
33	* ADDRESS								
AE									
37	. ADDRESS								
91									
23	; ADDRESS								

Each word has four fields: name field, link field, code field, and parameter field.

The first one is the name field. Its length varies according to the length of words name. The first byte specifies the number of characters of the word's name. Bit 7 (MSB) of the first as well as the last bytes of the name field are set to 1 to mark the range of the name field. We call bit 6 of the first byte the precedence bit, which is used to control compiling. The precedence bit is set to 1 if compiler directives should be executed immediately to carry out a specific compiling. However, precedence bit is usually set to Ø. In this case, its address is compiled into dictionary and becomes a part of high level words during compiling.

Bit 5 of the first byte is the smudge bit. Before the word is well defined, the smudge bit can protect the compiler from compiling the unfinished word. The smudge bit is cleared to be Ø when a high level word is defined, so that it can be compiled or interpreted for execution.

The link field saves an address, which is the name field address of the previous word in the dictionary. Name field and link field combines all words in the dictionary. When you wish to find out a specific word, FORTH follows the sequential stream, and compares the input name with the name field of each word. If they are different, jump to the name field of the previous word from the link field and make comparison with the next word.

The code field saves an address, which pointed to a machine code routine. The machine-code are executed before executing this word. Different code fields correspond to different machine-code routines. These machine-code routines are called interpreter or inner interpreter for the FORTH words.

The last one is the parameter field. Its length varies with different words. When executing inner interpreter, the inner interpreter makes use of the data in the parameter field to accomplish the task defined by the word. The values of constants and variables are saved in this field. The high level parameter field saves a series of code field addresses of other words. The high

level word interpreter finds out the addresses in order and executes the words. That is why we call high level word interpreter the address interpreter.

The parameter field of low level words contains a series of machine codes. The code field address contains the parameter field address. Therefore, when executing a low level word, you execute the machine code program in the parameter field directly. The program is the code interpreter of the low level word itself.

4.6 The Dictionary

As described in the previous section, all the words in FORTH are connected one after another by name field and link field. Its structure is illustrated as follows:

the previous word

| name field |
| link field |
| . |
| . |

| name field |
| link field |
| . |
| . |

| name field |
| link field |
| . |
| . |

the last defined word

HERE
(the next usable address for words)
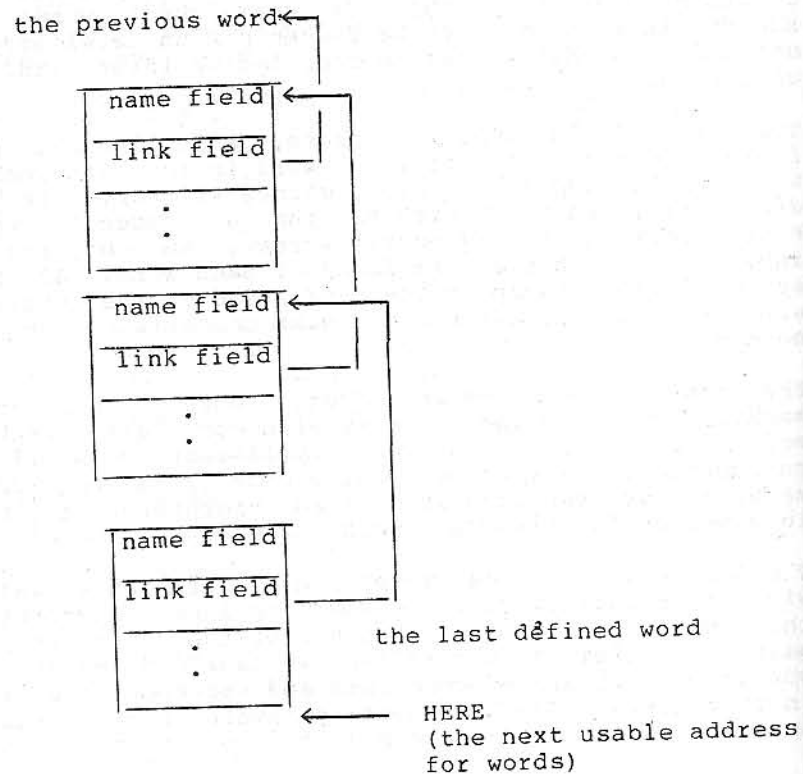
HERE is a FORTH word. It places the next usable address on the stack. Its value will change with the increasing number of the words.

After you have defined new words in FORTH, there are times that you would like to erase them. In this case, use the word FORGET. FORGET erases the word and the words defined later than that.

You may define a dummy word before the words for the test.

Input   : DUMMY ; ⏎
Display : DUMMY ; OK⌃

Then, executing FORGET DUMMY will erase everything defined later than the word DUMMY.

Input   : TEST1 5 + . ; ⏎
Display : TEST1 5 + . ; OK⌃

Input   : TEST2 5 * . ; ⏎
Display : TEST2 5 * . ; OK⌃

Input   FORGET DUMMY ⏎
Display FORGET DUMMY OK⌃

TEST1 and TEST2 are also erased after executing FORGET DUMMY.

CHAPTER 5

Structural Conditional Control

The structural program means that in the program the logical flow should follow one of the three ways listed below:

1) Consecutive Process: operating step by step. This is regularly used in high level words.

2) Conditional Branch: If the condition is true, do event A, otherwise do event B; event C follows A or B, as illustrated below:

```
                    Condition
        false                        true
     +--------+                  +--------+
     |   B    |                  |   A    |
     +--------+                  +--------+
              +--------+
              |   C    |
              +--------+
```
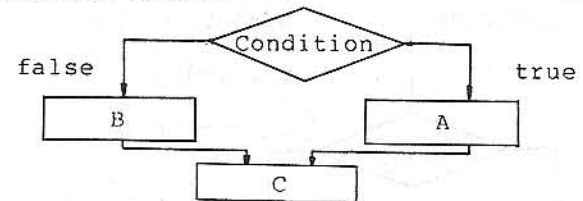
3) Loop: Repeat event A until a condition is true, and then do event B, as illustrated below:

```
              +--------+
              |   A    |
              +--------+
      false
              Condition
                        true
              +--------+
              |   B    |
              +--------+
```
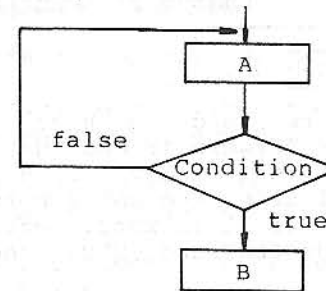
FORTH provides the use with words of all these three types, which enable you to write structural programs.

5.1 Conditional Branch

The conditional branch gives the computer the capability to make decisions. In FORTH, it is used to test the value on the top of the stack and decide if it is necessary to change the order of execution.

Below we will show you how it works.

: DEFINITION	Define a new word.
CONDITION	Produce a logical flag (zero or non-zero) and place it on the stack.
IF THIS	Fetch the flag and test it, if it is non-zero, execute THIS.
ELSE THAT	Execute THAT, if the flag is 0.
THEN CONTINUE	Continue with the following words.
;	The end.
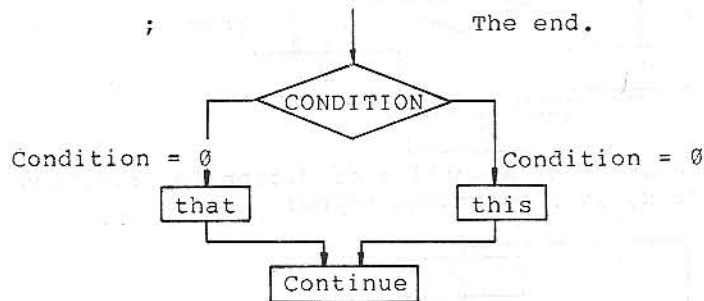
IF, ELSE and THEN are used in high level word definitions. All words between IF and THEN combine to make a "structure". IF tests the value on the top of the stack. If it is not zero, the words between IF and THEN will be executed. If it is zero, execution will jump to the words between ELSE and THEN, and continue with words that follow.

IF includes a test value 0=, which will use up the topmost value (logical flag). If this flag is to be used again between IF and THEN, you have to duplicate and save it before executing IF.

The following is another conceptual diagram for conditional branch. ELSE may be omitted in the structure IF...ELSE...THEN, if the test result is false, program flow skips the words between IF...THEN to execute the words after THEN.
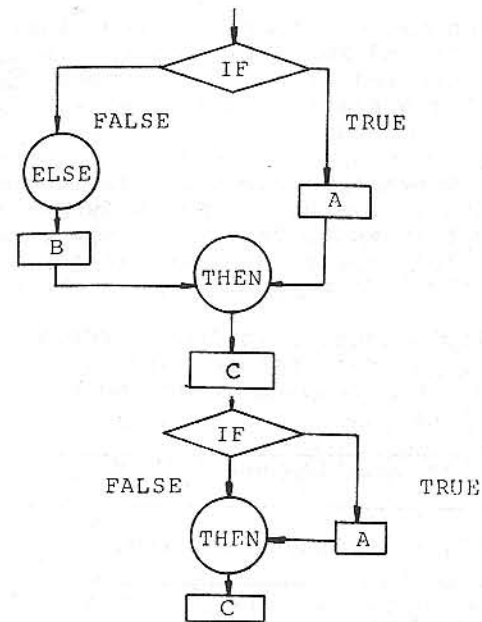
5.2 Compare Words

Compare words are usually divided into three kinds.

1) Words used to test the topmost value on the stack, such as 0=, 0>, and 0<.

2) Words used to test the two topmost values on the stack, such as =, >, and <.

3) Words used to test 32 bits double number on the stack, such as D<.

All compare words remove the value they require from the stack and return a flag. If the result is true, 1 (stands for true) is returned to the stack. If the result is false, 0 (stands for false) is returned to the stack. The word NOT reverses the flag, that is, change 0 to 1, and 1 to 0.

Suppose you wish to test a condition which is not smaller than 0 (larger than 0 or equals to 0), you may define it as follows:

```
: >=  0<  NOT ;
```

Results from comparison may be processed with logical operators such as AND, OR and XOR. Flags as the results of comparison can be treated as regular numbers and processed with arithmetic operators such as +, -, * and /.

The - (subtraction) operator may be used as compare word as well. The result of subtraction between two equal numbers is definitely zero. Otherwise, the result will be non-zero (which implies a true flag). The result is not necessarily "1", though.

The following table lists compare words in FORTH-MPF-IP. These words are usually used before IF and UNTIL and give them a flag, which is used to select the execution sequence thereafter.

Words	Stack Manipulation and Action
<	(n1 n2 - f) If n1<n2, f=1. Otherwise, f=0.
=	(n1 n2 - f) If n1=n2, f=1.
>	(n1 n2 - f) If n1>n2, f=1.
0<	(n - f) If n<0, f=1.
0=	(n - f) If n=0, f=1.
0>	(n - f) If n>0, f=1.
D<	(d1 d2 - f) If d1<d2, f=1.
U<	(un1 un2 -) If the double number un1<un2, f=1.
NOT	(f1 - f2) Reverse the value of the flag on the stack.

5.3 Loop

Loop has two basic types: finite and indefinite. The finite loop is set to repeat a certain number of times. The indefinite loop continue to circulate until a condition is met or a specific event develops. Among the indefinite loops, you will find one that will repeat endlessly until an external force is applied. This is generally called an infinite loop.

Some FORTH words can contain different kinds of loops in the word definitions, in order to handle a sequence of commands to be executed repeatedly. These structures can only be defined in the new words. They must not be input from the keyboard, and executed immediately, otherwise an error will develop.

5.3.1 Finite Loop

The finite loop can be classified into two kinds according to the way the loop index increments:

1)   limit   index   DO   words   LOOP

Each time the words between DO and LOOP are executed, the index increments by one, and then execution continues until the index equals to the limit.

The loop index and the limit are saved in the return stack temporarily to avoid problems arising from using the data stack when executing words between DO and LOOP.

2)   limit   index   DO   words   incr   +LOOP

The index increments by incr for each loop until the index equals to is equal than the limit.

If the INCR is negative, the limit should be smaller than the index. The index decrements by INCR for each loop until the index is smaller than or equal to the limit.

DO, LOOP, and +LOOP should be used in the definition. They must not be executed immediately. Otherwise, the system will send back an error message.

Define the following word in your system:

```
: TEST1  5  0  DO  I  .  LOOP  ;
```

The two numbers before DO are used to control the loop. 0 is the initial value of the index. 5 is the limit. 0 and 5 will be saved in the return stack upon execution of DO. The word I will copy the index on the data stack, and the word prints it. When execution comes to LOOP, the current value of the index increments and compares it with the limit. If it exceeds or equals to the limit, the loop stops, the limit and the index on the return stack will be removed, and execution continues with the words after LOOP. If the index does not exceed the limit, execution will jump to the word DO and executes the words between DO and LOOP again.

```
Input     TEST1  ⏎
Display   TEST1  0  1  2  3  4  OKʌ
```

The loop stops immediately when the limit equals 5.

Try the following word:

```
: TEST2  10  0  DO  I  .  3  +LOOP  ;
```

When using the word +LOOP, if the INCR is not a negative number (in this definition, the INCR is 3), the index should be smaller than the limit.

```
Input     TEST2  ⏎
Display   TEST2  0  3  6  9  OKʌ
```

When the index equals 9, another increment at the +LOOP will make the amount 12, which exceeds the limit and ends the loop.

Define the following word:

```
: TEST3  -4  0  DO  I  .  -1  +LOOP  ;
```

If the INCR is negative, the index should be larger than the limit.

```
Input     TEST3  ⏎
Display   TEST3  0  -1  -2  -3  OKʌ
```

The loop stops when the index is smaller than or equals

the limit.

The return stack saves the limit and the index' between the words DO-LOOP. They will be removed automatically upon completion of the loop. The system will lose control if there are operations affecting the return stack during execution of the DO-LOOP. The words R@, >R, and R> may access the return stack for data required. Be careful when you use these words. R> should follow >R, so that the contents of the return stack will not change.

The following are two important rules to remember when you use DO-LOOP.

1) DO should be followed by LOOP or +LOOP in a definition.

2) The words between DO and LOOP can not change the contents of the stack, that is , the stacks should remain intact against the execution. There might be exceptions in specific occasions, but they should be avoided if there are other ways.

5.3.2 Indefinite Loop

The indefinite loop has also two types: one is BEGIN...UNTIL, another is BEGIN...WHILE...REPEAT.

1) BEGIN words condition UNTIL

Execute the words continuously until condition produces a true flag on the stack.

2) BEGIN words1 condition WHILE words2 REPEAT

Execute words1 at least once, then if the condition is true, execute words2 and jump back to execute words1 at REPEAT. If the condition is false, the loop ends and jumps to the words after REPEAT.

Try to define the following words:

```
: TEST4  BEGIN  KEY  DUP  EMIT  65  =  UNTIL  ;
```

KEY reads the ASCII code of a character from the keyboard, and EMIT prints the character. The loop ends when the character is A (ASCII code of A is 65).

```
Input      TEST4 ◄━┛ BKCFA
Display    TEST4 BKCFA  OK
```

5.3.3 Infinite Loop

BEGIN...UNTIL may be used to set up an infinite loop.
Consider the following structure:

```
:  ......... BEGIN ........ Ø  UNTIL  ;
```

The flag (Ø) that UNTIL examines is always false,
therefore, the loop will never come to an end. We have
an infinite loop structure unique to FORTH-MPF-IP.

```
:  ..... BEGIN ..... AGAIN  ;
```

The words between BEGIN and AGAIN will be executed over
and over again.

The infinite loop is usually used in a complete set of
operating system as a main program. The input device
reads the data first. The system then processes it, and
outputs the data. Finally, execution starts from the
beginning anew.

As described in the section of the indefinite loop, the
stack must not be changed, or the system will run out
of order.

In the following list, you will find the words used to
set up the loop and control the return stack.

Words	Stack Manipulation and Action
IF XXX ELSE YYY THEN ZZZ	IF : (f -) If f does not equal Ø, execute XXX, otherwise execute YYY and then ZZZ. ELSE YYY may not be used.
DO XXX LOOP	DO : (nl n2 -) LOOP : (-) Set up a loop structure. The index is incremented from n2 to nl-1.
DO XXX +LOOP	DO : (nl n2 -) +LOOP : (n3 -) As DO...LOOP, n3 is the INCR of the index.
LEAVE	(-) Set the limit equal to index. The loop ends at the next LOOP or +LOOP encountered.
BEGIN XXX UNTIL	UNTIL : (f -) Set up an indefinite loop. If the flag is Ø, start the loop all over again at UNTIL.
BEGIN XXX WHILE YYY REPEAT	WHILE : (f -) Set up an indefinite loop. If the flag is Ø when executing WHILE, jump to the words after REPEAT and end the loop, otherwise execute YYY.
BEGIN XXX AGAIN	Set up an infinite loop.
END	Same as UNTIL.
ENDIF	Same as THEN.
>R	(n -) Remove the topmost value on the stack, and save it on the return stack.
R>	(- n) Remove the value from the return stack, and save it to the data stack.
R@	(- n) Copy the topmost value on the return stack to the data stack.
I	(- n) As R@, used in DO-LOOP and put the index on the data stack.
J	(- n) Used in DO-LOOP, and copy the index of the outside loop to the data stack.

CHAPTER 6

Printing Strings and Numbers

A string is a set of characters and symbols, saved in memory as ASCII codes. The string is the only way that the computer input/output the message to communicate with the operators. Words and data are input as strings. The computer interprets them as instruction codes. It also transcribes the data into strings when outputting the results.

Users are requested to control the printing formats and locations for the numbers. In FORTH, we may use the string combination to control the conversion of numbers and printing format.

6.1 Strings Manipulating Words

The following table lists some basic string commands. They are used to set or move the string data.

Words	Stack Manipulation and Action
CMOVE	(a1 a2 n -) Move n bytes from address a1 to address a2.
FILL	(a n b -) Fill memory beginning at address a with a sequence n copies of b.
ERASE	(a n -) Erase n bytes starting from address a.
BLANKS	(a n -) Fill an area of memory beginning at address a with n blanks (ASCII code = 32).
DUMP	(a n -) Print n bytes starting from address a.

The following example shows the result obtained by using DUMP.

```
HEX   OK
3000  20  DUMP
 3000  FC  25  AC  2A
 3004  8F  24  60  2A
 3008  3E  29  91  23
 300C  C7  4C  49  54
 3010  45  52  41  CC
 3014  EE  2F  87  25
 3018  22  27   D  25
 301C  DA  20   8   0
OK
```

We usually use a string buffer to handle strings. The word PAD can get the address of the string buffer.

    : PAD HERE 68 + ;

PAD is a memory range in the dictionary. It moves as the dictionary changes. The data in PAD should be used before defining a new word, otherwise we can not be sure if the original data still exists.

6.2 Single Character Input/Output

KEY is a basic input command in FORTH. When KEY is executed, the system will wait for you to input a character, and then push its ASCII code on the stack. You may use the ASCII code when necessary later.

Input   KEY ⏎
Display   KEY

The cursor is displayed on the screen while you can not find OK. This is because the word KEY is not yet finished. The system waits for you to input a character.

Input   A
Display   KEY OK⌄

The character A is not displayed, but the ASCII code for the character A is placed on the stack.

Input   . ⏎
Display   . 65 OK⌄

The word EMIT removes the ASCII code from the stack and prints its corresponding character.

Input   65 EMIT ⏎
Display   65 EMIT A OK⌄

6.3 String Input/Output

The word TYPE may output a whole string. It needs two parameters: one is the address of the string in memory, the other is the string length (number of characters).

Example: PAD 16 TYPE

prints 16 characters stored in the PAD buffer.

The following table contains words for string output.

Words	Stack Manipulation and Action
."XXX"	(-) Print the string XXX, the last " is used as a delimiter.
TYPE	(a n -) Print the n bytes starting from address a.
-TRAILING	(a n1 - a n2) Remove trailing blanks in the string of n1 character starting from address a. Reduce n1 to n2 for printing by using TYPE.
MESSAGE	(n - .) Print the characters on the nth line in the 4th block. n may be negative or larger than 15, so as to print characters out of the 4th block. If WARNING contains 0, this command only prints n. If WARNING contains 1, prints characters stored in the disk.
PAD	(- a) Push the starting address of string buffer a on the stack. The string buffer moves with top of the dictionary. Input and output strings are saved in the string buffer for future use.

COUNT	(a – a + 1 n) Place the string length n stored in the address a on the stack, and add one to a. The results may be used by word TYPE for printing.
EMIT	(c -) Send a character to terminal whose ASCII code is on the stack to terminal.
CR	(-) Position the cursor to the beginning of the next line.

The basic word for inputting a string is EXPECT. It is used in the form below:

    addr n EXPECT

As this word is executed, FORTH will wait for the user to input n characters and save the string in memory starting from addr. We may use the word to store the input string anywhere we want in the memory.

```
Input      HEX  ⬅
Display    HEX  OK⌃

Input      F400  2  EXPECT ⬅
Display    F400  2  EXPECT
```

Same as KEY, OK does not display on the screen. This indicates the execution of EXPECT is not finished yet.

```
Input      A
Display    F400  2  EXPECT  A

Input      B
Display    F400  2  EXPECT  ABOK⌃
```

We can use DUMP to examine the content in the memory.

```
    F400  4  DUMP
     F400  41  42  0  0
    OK
```

The FORTH has a special memory range for saving input characters for text interpreter. It is called a terminal input buffer (TIB). The starting address is saved in the system variable TIB. The word that inputs string by using the buffer is QUERY.

    : QUERY TIB @ 80 EXPECT 0 >IN ! ;

QUERY receives 80 characters or all the characters coming before CR, and input them to TIB. It sets the character pointer >IN to 0 for interpreting. The following table contains some basic words for input in the system.

Words	Stack Manipulation and Action
KEY	(– c) Read the data and push its ASCII code to the stack.
?TERMINAL	(– f) 1 is put on the stack if a key is pressed; 0 is put on the stack if no key is pressed.
EXPECT	(a n -) Input n characters from keyboard and save it in the memory starting from address a.
QUERY	(-) Read a line of characters (80 at most), and save it in the TIB.

6.4 Printing Format for Numbers

The fundamental word for printing numbers is D.R. Earlier in this book, we have introduced some words such as D.R, D., U., .R, ., and ?. However, these words can print numbers in the form of integer, they can not insert special symbols such as decimal point or comma.

Sometimes we have to insert a specific symbol, such as, the dash (-),the dollar sign (S), the slash (/), and the colon (:).

FORTH provides words for printing numbers as illustrated in the following table.

Words	Stack Manipulation and Action
<#	<# Begin conversion of a value to a numeric string.
#	(udl - ud2) Evaluate the number following udl, the result ud2 is placed on the stack. The number is added to the output numeric string.
#S	(ud - 0 0) Convert all the ud until the remainder is a zero. The number evaluated is added to the output string.
HOLD	(c -) Add the character c to the output string.
SIGN	(n -) If n<0, add a minus sign to the output string.
#>	(d --- a n) Drop the double number d. Place the address of output number string a and number of characters n on the stack.

FORTH converts the saved values to the number string according to the following procedure.

1) The numbers are converted in the order from the right to the left.

2) The value for conversion on the stack must be a double number.

Consult the following table which describes a number of ways to arrange data in printable format.

Value to print	Steps to take before <#
16 bit number	Add 0 to make a 32-bit double number.
15 bit single number	DUP ABS 0 Save the signs (plus or minus) on the 3rd position of the stack, to be used later by SIGN.
32 bit double number	None.
31 bit double number	SWAP OVER DABS Save the signs.

Define the following word:

```
: $D.  SWAP  OVER  DABS  <#  #  #  46  HOLD
       #S  36  HOLD  SIGN  #>
       TYPE  SPACE  ;
```

Save a 31 bit double number on the stack before using $D.. SWAP OVER DABS convert the value on the stack to a double number, and reserve the sign. <# sets a buffer to save the bytes converted from the number you want to print.

uses the current base to convert a digit to a character, and saves it in the buffer. The digit will be removed from the original number. For example, suppose 789 is in the stack. After executing #, the character 9 will be put in the buffer, and 78 is still on the stack.

46 HOLD inserts a decimal point in the buffer (46 is the ASCII code for . (dot)).

#S converts the numbers remained on the stack to the bytes in the buffer and remains a double number 0 on the stack.

36 HOLD adds a "$" (dollar sign) in the buffer (36 is

the ASCII code for "$").

If the 3rd value on the stack is negative, the word
SIGN puts the character "-" (minus) in the buffer and
removes the sign of the value.

#> ends the conversion. The double number 0 is removed,
but the start address in the buffer and the length
after conversion remain on stack.

TYPE uses the address and length left by #> to output
the result of conversion in the buffer.

Try the following examples:

```
Input      3456.  $D.  ⏎
Display    3456.  $D.  $34.56   OK⌃

Input      -123.  $D.  ⏎
Display    -123.  $D.  -$1.23   OK⌃
```

7
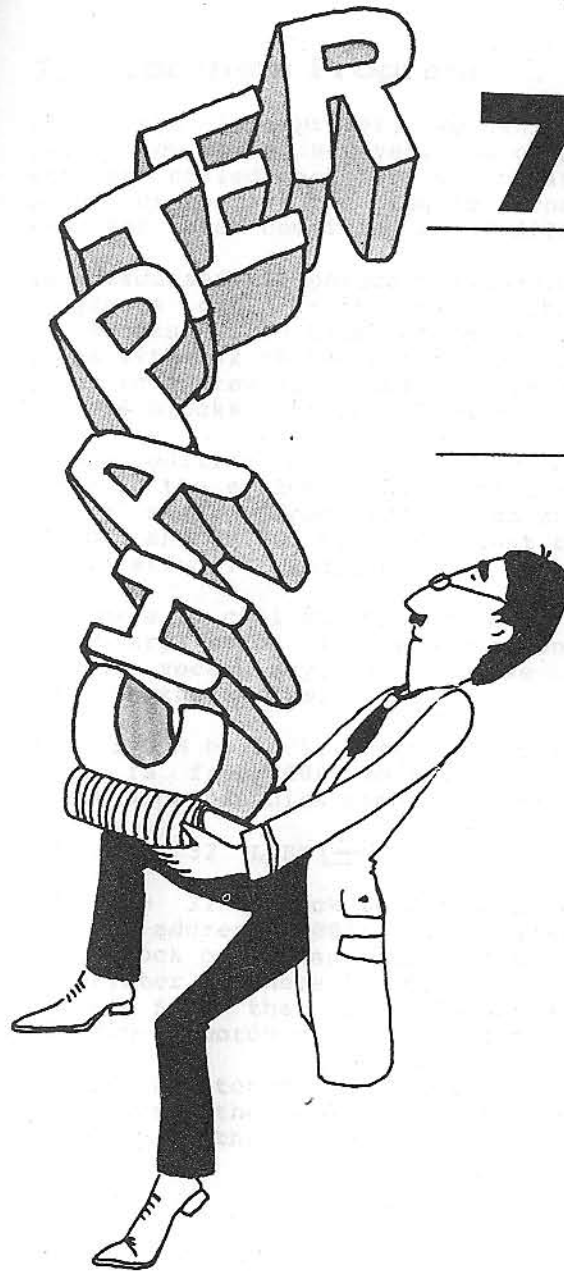
Editor

7.1 Editing a Program

Under the interpreter, we can key in a program to define new words. However, the completed definition can not be called back for modification. Editing words allow us to save the program's contents in a magnetic tape for later compiling and modification.

We discussed the pseudo disk memory in Chapter 4. The program's contents is saved in the pseudo disk memory as blocks. Each block contains 512 characters in 16 lines with 32 characters in each line. We allocate 28K bytes in system as pseudo disk memory, which is divided into 56 blocks. Its serial number is from 0 to 55.

Before editing a program, you have to know the RAM range in the system. The initial value for pseudo disk memory starts from $8000. You can set the value of OFFSET and UFIRST to assure that the program is edited in the effective RAM range.

You have to call EDITOR before editing. EDITOR is a vocabulary word. It sets the context vocabulary as editing vocabulary, so that we may use the editing words in the system.

If an I/O M board is installed to the system. Its RAM range is from $C000 to $D7FF. Use the word LIST to select a pseudo disk memory for editing.

Input   32 LIST ⏎

Now, the 32nd block is selected for editing (starting from the address $C000), and prints the characters on 32nd block on the screen. (It will print the data on the printer,if there is any). 32 is saved in the system variable SCR, that is, it is set as the current block. All editing words change the data only in this block.

The word L fetches the serial number of the block from SCR and uses the word LIST to print it. Key in the word L to display the characters in the current block.

7.2 Line Editing Words

The editing words input strings to the current block or modify its characters. Most words are used to handle strings. Editing words usually save strings in a special string buffer. You obtain the starting address of the buffer from the word PAD.

The characters saved in PAD can be used repeatedly so that you do not have to key in each time you use them. PAD saves temporarily the strings for input, insertion, deletion, and search.

The editing cursor is used to point out the current editing byte symbolized with ∧ on the screen, and ↓ on the printer. Its value is from 0 to 511, saved in system variable R#, which records the line number and character number under editing. Many editing words use the cursor for subsequent editing.

We call words T, P, U, X line editing words, which are used to manage an entire line of data (32 bytes).

The word to set the nth line as the current line is :

        n T [⏎]

n is from 0 to 15, which indicates the line number currently under editing and prints the line. At the same time, the entire line of characters are saved in PAD. Editing cursor (value in R#) is also placed before the first character of the nth line.

The word T is usually used to move the cursor to a specific location for subsequent editing.

The word to input a line of characters on a specific line is:

        P XXXX [⏎]

(XXXX represents a string, with a length of up to 32 bytes). The string XXXX is input in the line that the cursor is located and replace (overwrites) the original characters. XXXX is also saved in PAD buffer. If you input the carriage return ([⏎]) immediately after P, the characters in PAD are moved without changes to the line currently under editing. If you

insert two spaces between P and [⏎], characters in PAD and the current line will all be cleared to spaces.

As the word P is an independent command, it has to be delimited from the strings with a space, while the second space will be regarded as part of the string. The word P has the following three usages:

1) P XXXX [⏎]   Put XXXX in the current line.

2) P [⏎]   Move characters in PAD to the
   (No space in   current line.
   between)

3) P   [⏎]   Clear PAD and current line.
   (Two or more
   spaces in between)

The word U is used to input a line of characters immediately under the current line, and push the subsequent lines down one line.

        U XXXX [⏎]

Characters on the 15th line will be erased.

The word U has also three usages:

1) U XXXX [⏎]   Input XXXX immediately under the
                     current line. Lines move the
                     subsequent down one line and
                     clear the 15th line.

2) U [⏎]   Move string in PAD immediately
                     under the current line. Move the
                     subsequent lines down one line.

3) U   [⏎]   Clear PAD and the current line.
                     Move the subsequent lines down
                     one line.

To delete the current line, type

        X [⏎]

The word X deletes the current line. The subsequent lines "scroll" up one line. The last (15th) line is filled with spaces. The characters on the deleted line are saved in PAD buffer.

-72-

-73-

7.3 Editing a String

String editing words include F, D, TILL, I. To modify a small section in a line, they can effectively search, add, or delete a section of characters or strings.

F XXXX ⬅

The word F searches for the string XXXX starting from the cursor's current position. If it finds the target, it prints the entire line containing the string, and moves the cursor positioned after the string. If it does not, it prints an error message, and moves the cursor to the beginning of the block.

D XXXX ⬅

The word D searches for the string XXXX from the characters after the cursor and deletes it. The cursor is placed after the deleted string. If the target string is not in the block, it prints an error message, and moves the cursor to the beginning of the block.

TILL XXXX ⬅

The word TILL deletes data in the range from the cursor, to the XXXX (inclusive).

I XXXX ⬅

The word I inserts the string XXXX after the current position of the cursor, and moves the cursor positioned after the string.

The following table lists editing words in the FORTH-MPF-IP.

Words	Stack Manipulation and Action
T	(n −) Print the nth line and move the cursor to the beginning of the line.
P XXX	(−) Place the string XXX on the current line.
U XXX	(−) Insert XXXX under the current line. Move the subsequent lines down one line.
X	(−) Delete the current line. Move the subsequent lines up one line. The deleted characters are saved in PAD.
F XXXX	(−) Search for the string XXX from the cursor position. The cursor is placed after the target string. If the string is not found, the cursor moves to the beginning of the line 0.
D XXXX	(−) Delete the string XXX found somewhere after the cursor position.
I XXXX	(−) Place the string XXXX after the cursor.
TILL XXX	(−) Delete characters in the range from the cursor and the string (inclusive).
COPY	(n1 n2 −) Copy data in block n1 to n2.
CLEAR	(n −) Clear the nth block.
TOP	(−) Move the cursor to the beginning of line 0.

L	(-) Reprint the current block.
LIST	(n -) Print the nth block and set it as current block.
INDEX	(n1 n2 -) Print characters on line 0 of each block starting from block n1 through block n2.
CONTROL I	(n -) Move the cursor n bytes. CONTROL I is the TAB key.

Execution of a cold start on the FORTH-MPF-IP will clear the memory from $8000 through $EFFF to zeros (ASCII NULL). You have to edit data line by line starting from line 0. FORTH stops compiling when it encounters an ASCII NULL, and no compiling will be executed after a null line is encountered.

If the block you are editing is not cleared (e.g. move the editing block outside of $8000 - $EFFF by the use of OFFSET or UFIRST), the last line should include the word ;S or EXIT to stop editing. ;S and EXIT have the same effect that ASCII NULL does.

After editing, you may use the word TWRITE to save the data in pseudo disk to the magnetic tape. The procedure is as follows:

Suppose you want to save the data in block 1 through block 5 to the tape with a filename of TEST.

```
Input      1 5 TWRITE  [←]
Display    < NAME >=ʌ

Input      TEST
Display    < NAME >=TESTʌ
```

Set the recorder ready and press the RECORD key, and finally press [←], the MPF-IP sends out a sound and begins to transmit the data to the tape until the screen displays:

< NAME >=TEST OK

which indicates the end of transmission.

The word TREAD reads the data on the tape to the pseudo disk. Remember that the value of OFFSET and UFIRST must be the same as before to avoid loading the data to incorrect locations.

```
Input      TREAD  [←]
Display    < NAME >=ʌ

Input      TEST
Display    < NAME >=TESTʌ

Input      [←]
Display    ....
```

which means the system is waiting for input of data. Please refer to MPF-IP operation manual on saving to and reading from the tape, and the format for the stored data.

7.4 Compiling FORTH Words

If the program is written in the memory block of the pseudo disk, be sure to compile the words in the block to the dictionary before you perform the test.

Suppose you want to compile the words in the first block.

```
Input      1 LOAD  [←]
```

Words in the first block will be executed in sequence. Any newly defined words will be added to the dictionary after compilation.

Very few application programs can be written and fit in one memory block. FORTH-MPF-IP has a word ➤ , which carries the compilation ahead into the next consecutive block until it meets ;S, EXIT, or ASCII NULL.

Printing the original program on the printer helps the user examine its contexts to facilitate modification and test during compilation. The word LIST prints a block's data on the printer in an area of 16 lines with 32 characters on each line.

For example: 3 LIST [←]

The word nl n2 INDEX prints the characters in line 0 of
each block from blocks nl through n2.   Therefore, line
0 is usually used as a remark to explain the content of
the block.

CHAPTER 8

Interrupt Signal
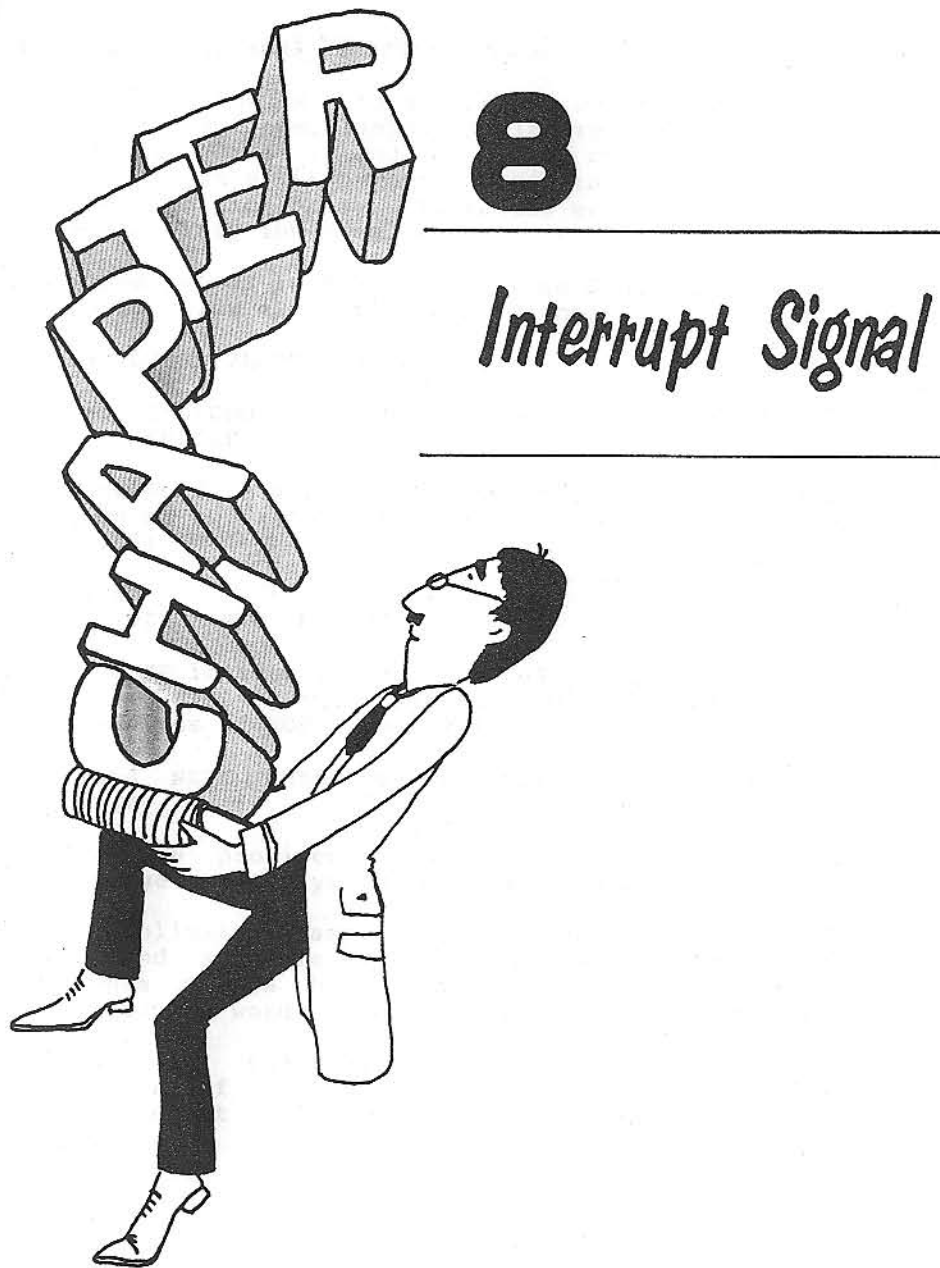
8.1 Low Level Words in FORTH

FORTH allows the user to define new words in high level as well as low level languages. It provides a primitive Assembler: words , and C,. They can move a 16-bit number or an 8-bit number on the stack to the upper part of the dictionary. These two words enable us to establish every low-level word.

Low level words in FORTH start with CODE and end with END-CODE. Below are their definitions:

```
:  CODE  ?EXEC  CREATE  !CSP  ;

:  END-CODE  CURRENT  @  CONTEXT  !  ?EXEC  ?CSP
   SMUDGE  ;
```

The last word in a low-level word must jump to the word NEXT so as to execute the next word. Take a look on this word:

```
HEX
:  NEXT  0C3  C,  2078  ,  ;
```

0C3 is a JP instruction code of Z-80 CPU. (refer to Z-80 Assembly Language Programming Manual) 2078 is an entry address of FORTH-MPF-IP inner interpreter (NEXT).

The word NEXT puts the instruction JP 2078 on the dictionary.

MPF-IP-FORTH provides the preceding three words, and you may use them as you start the system.

In the following example, we will define a variable COUNTS and a low-level word COUNT-DOWN. COUNT-DOWN decrements COUNTS by one consecutively until COUNTS becomes 0. The word can be used as a delay subroutine.

Enclosed in the parentheses are the Assembly equivalents of the FORTH definition. For details, please consult Z-80 Assembly Language Programming Manual.

HEX

VARIABLE   COUNTS

CODE   COUNT-DOWN

2A C, COUNTS   (LD HL,(COUNTS))

2B C,   (DEC HL)

7C C,   (LD A,H)

B5 C,   (OR L)

20 C, FB C,   (JR NZ,FB)

NEXT END-CODE

Set the value of COUNTS first, and then execute COUNT-DOWN.

Input   7FFF COUNTS ! ⏎
Display   7FFF COUNTS ! OK˄

Input   COUNT-DOWN ⏎
           The FIP will black out for a few seconds
           and then

Display   COUNT-DOWN OK˄

The user should find out all machine codes before using
, and C,, and compile them one by one into the
dictionary. The procedure to find all machine codes by
the Assembler is as follows:

1) Execute the word MON to enter into the MPF-IP
    monitor program.
2) Execute the Assembler under the monitor program and
    write down the machine codes (refer to MPF-IP
    Operation Manual).
3) Input CTRL-C to execute a warm start, and use , and
    C, to compile the machine codes into the
    dictionary.

FORTH-MPF-IP supports a word CALL, which allows the
user to call machine language subroutines in high level
words, and system variables to save registers, such as
RA, RB, RC, RD, RE, RF, RH, RL, RAF, RBC, RDE, RHL,

RIX, RIY, RAF', RBC', RDE', and RHL'. The word CALL can
use these variables to transmit parameters and results
of execution. C@ and C! are used to fetch and store 8-
bit registers. @ and ! are used to fetch and store 16-
bit register pairs.

The system first fetchs numbers from RAF, RBC, RDE,
RHL, RIX, RIY and stores numbers in registers AF, BC,
DE, HL, IX, IY, before the word CALL is executed to
enter machine language subroutine. In other words, if
the called sub-routine needs some parameters saved in
registers, the user can save the parameters in the
register variable first, and then execute CALL. The
system saves the values in registers AF, BC, DE, HL,
IX, IY to the variables RAF, RBC, RDE, RHL, RIX, RIY
before the sub-routine returns to the FORTH, so as to
transmit the results of execution.

The monitor program has a sound generation subroutine.
Its address is $874. Two parameters are related to this
subroutine.

1) Register C   period = $2*(44+13*C)$ clock states

2) Register HL   number of periods (times of
                     execution)

The larger the value in C is , the lower the frequency
it has; the smaller value, the higher frequency. The
larger the value in Register HL is, the longer the
sound continues.

   HEX

   : TONE1 100 RHL ! 7F RC C! 874 CALL ;

   : TONE2 800 RHL ! 10 RC C! 874 CALL ;

You will get two different kinds of sound when
executing TONE1 and TONE2.

8.2 Low Level Interrupt Handler

The following words are provided in FORTH-MPF-IP to handle interrupt signals.

Words	Stack Manipulation and Action
EI	(-) Enable interrupt
DI	(-) Disable interrupt
IM0	(-) Set interrupt mode 0
IM1	(-) Set interrupt mode 1
IM2	(-) Set interrupt mode 2
INTVECT	(- addr) System variable, which saves interpretive interrupt vector.
INTFLAG	(- addr) System variable, which saves interpretive interrupt flag.
;INT	Ends an interpretive interrupt word.

For interrupt handling in low-level words, we can use EI, DI to control IFF (internal interrupt flip-flop in Z-80), and use IM0, IM1, IM2 to select interrupt mode. The other steps are the same as the Assembly. Please refer to Z-80 CPU manual.

MPF-IP sets a vector address, which can save the entry address of the interrupt handling subroutine to handle interrupt mode 1.

Examples:

DI   (Disable interrupt)

IM1   (Set interrupt mode 1)

HEX HERE   (Reserve entry address of the program)

E5 C,   (Push HL)

21 C, INTFLAG ,   (LD HL, INTFLAG)

FECB ,   (Set 7, (HL))

EI C,   (Pop HL)

4DED ,   (RETI)

FF01 !   (Save entry address of the program into vector address)

EI   (Enable interrupt)

8.3 Interpretive Interrupt Handling Process

The so-called interpretive interrupt handling is the definition of the interrupt handling process in high level words. FORTH-MPF-IP has set two system variables INTVECT and INTFLAG. Every word must return to the inner interpreter after execution and proceed to the next word. The inner interpreter examines the INTFLAG to handle interrupt signal properly. The INTFLAG uses 2 bits in one byte. Its format and significance are as follows:
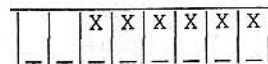
bit 6 : 1 -- inhibited interpretive interrupt
        0 -- not inhibited interpretive interrupt

bit 7 : 1 -- interpretive interrupt request
        0 -- interpretive interrupt not request

When the inner interpreter examines INTFLAG and handles interrupt signal, it fetches CFA (code field address) in INTVECT and begins to execute the interrupt handling program.

The interpretive interrupt handling takes the following steps:

1) Set interrupt mode 1;
2) Save CFA of the interrupt handling word in INTVECT;
3) Set INTFLAG bit 7 to 1 when producing interrupt signal develops.

Be sure the interrupt handling word in (2) should end with ;INT. Step in (3) should be executed in low-level words.

Suppose we have saved the previous examples in the dictionary. The following example explains the usage of interpretive interrupt handling.

        DI   (Disable interrupt)

        : INHANDLER ." INTERRUPT HANDLER" ;INT
                                (The word ;INT ends the
                                definition of interrupt
                                handling words).

        ' INHANDLER CFA INTVECT !
                                (Save CFA of the interruption
                                handling word in the
                                INTVECT).

        EI   (Enable interrupt).

        : TEST BEGIN ." X" ?TERMINAL UNTIL ;
                                (Define a test word).

When executing TEST, you will see X's displayed on the screen continuously. When interrupt signal develops, the machine outputs INTERRUPT HANDLER and then goes on to output X continuously until you press any key.

CHAPTER 9

Application Programs

9.1 Using P@ and P!

The words P@ and P! in FORTH-MPF-IP are similar to IN and OUT in Assembly language. Connect the I/O M board to the machine if you want to use them, you will find these two words make it easy to control the I/O ports.

Please refer to IOM-MPF-IP Operation Manual on how to connect the IOM-MPF-IP to the MPF-IP. There is a PIO on IOM-MPF-IP, the addresses are from 68H to 6BH. Connect sockets TR1, TR2, TR3 of J3 to the sockets PA0, PA1, PA2 of J6 respectively. Type

    HEX

    ØF   6A   P!   (set PIO port A as output)

and you will see the red, yellow, and green lights on IOM-MPF-IP are on. PA2 (red), PA1 (yellow), PA0 (green) of port A controls the three LEDs. If the output is 1, the LED turns off. If it is 0, the LED turns on.

Input   FE   68   P! [⏎]   the green light on

Input   FD   68   P! [⏎]   the yellow light on

Input   FF   68   P! [⏎]   all lights off

Input   FA   68· P! [⏎]   the red and the green
                                   lights on

The following table shows the stack manipulations for P@ and P!.

Words	Stack Manipulation and Action
P@	(addr--n) Input data n from I/O port addr.
P!	(n addr--) Output data n to I/O port addr.

9.2 Developing Application Programs

This section discusses the process for developing application programs. Basically, we need EPB-MPF-IP. If it is used together with IOM-MPF-IP, we can write programs on EPB-MPF-IP and produce EPROM and then move the EPROM to the socket with the same address on IOM-MPF-IP. Refer to the EPB-MPF-IP operation manual to connect the EPB-MPF-IP to the MPF-IP. The addresses where the application programs is to be located must have a RAM available. Suppose the starting address for application program is $D800, the general process is as follows:

1) Be sure the addresses for application program have a RAM available and does not intermix with other units.

2) Turn on the machine, and enter into FORTH-MPF-IP (CTRL-B).

3) Delete the word TASK.

```
    FORGET   TASK
```

4) Move the system variable DP (dictionary pointer) to the location seven bytes above the starting address of the application program. The added 7 bytes will be used to store machine codes later.

```
    HEX
    D800   7   +   DP   !
```

If the starting address is different from the example above, you need only change D800.

5) Compile the application program to the dictionary, and use the word VLIST to verify.

6) Move the DP to its original address and restore the word TASK.

```
    F000   DP   !
    :   TASK   ;
```

7) Input the machine codes (boot program for the application program) to the 7 bytes above the starting address of the application program.

```
21  D800  C!              (LD   HL, LAST)
F005  @  D801  !
22  D803  C!              (LD   (F005), HL)
F005  D804  !
C9  D806  C!              (RET)
```

8) Save the application program onto the recorder.

9) Use EPB-MPF-IP to input the application program to the EPROM.

10) Turn off the machine. Replace the RAM of the same address with the EPROM.

11) Turn on the machine, and enter into the FORTH-MPF-IP (CTRL-B).

12) Execute the boot program of the application program.

```
        HEX   D800   CALL   DECIMAL
```

13) Use the word VLIST to examine if the application program is in the dictionary.

In step 5), be sure to compile the application program within the range of the RAM. All variables in the application program must be user variables.

Example:

```
*****MPF-I-PLUS*****        Turn on the machine.
<                           (CTRL-B to
****FORTH-MPF-IP****        enter into FORTH-MPF-IP)
FORGET TASK OK
HEX OK
D800 7 + DP ! OK            Compile    the    application
                            program.

: TEST1 5 * . ; OK
: TEST2 5 + . ; OK
VLIST                       Make sure the  application
D825 TEST 2                 program has been compiled into
D811 TEST 1                 the dictionary
3AF4 MON
3AE8 EI
OK
F000 DP ! OK
: TASK ; OK
VLIST
F009 TASK
D825 TEST2
D811 TEST1
3AF4 MON
3AE8 EI
OK
21 D800 C! OK               Load the boot program
F005 @ D801 ! OK            of the application program
22 D803 C! OK
F005 D804 ! CK
C9 D806 C! OK
                            Enter into the monitor program.
MON                         Use the disassembler in the
<D>=D800                    monitor program to  examine
                            if  the  boot program  is
                            right (be sure to connect  the
                            printer).
<
D800 21 LD HL,D81B
D803 22 LD  (F005),HL
D806 C9 RET
```

Save the application program onto the recorder (refer
to MPF-IP operation manual).

Input the application program to EPROM (refer to EPB-
MPF-IP users' manual).

Turn off the machine and replace the RAM with the
EPROM.

```
****MPF-I-PLUS****          Turn on the machine.
<                           (CTRL-B to
****FORTH-MPF-IP****        enter into FORTH-MPF-IP)
VLIST                       Inspect the condition  after a
F009 TASK                   cold start.
3AF4 MON
3AE8 EI
OK
HEX D800 CALL OK            Execute the boot program of
                            the application program.
VLIST                       Verify the application
F009 TASK                   program  is  linked  to  the
D825 TEST2                  dictionary.
D811 TEST1
3AF4 MON
3AE8 EI
3ADD DI
OK
DECIMAL OK
3 TEST1 15 OK               Test the application program.
5 TEST2 10 OK
```
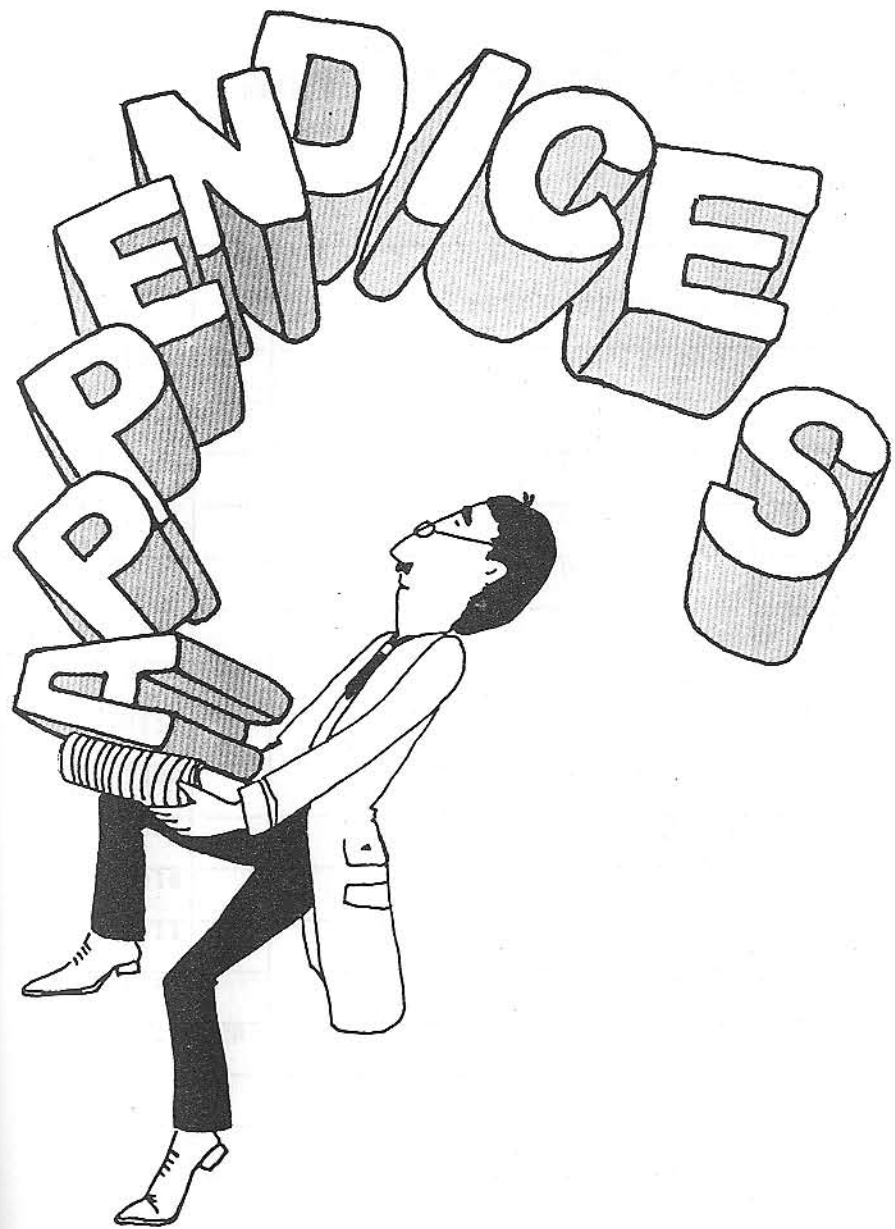
A MPF-IP ASCII Codes

MSD	0	1	2	3	4	5	6	7
LSD	000	001	010	011	100	101	110	111
0 0000			space	0	@	P		
1 0001			!	1	A	Q		
2 0010			"	2	B	R		
3 0011			#	3	C	S		
4 0100			$	4	D	T		
5 0101			%	5	E	U		
6 0110			&	6	F	V		
7 0111			l	7	G	W		
8 1000			(	8	H	X	→	
9 1001			)	9	I	Y	↓	
A 1010			*	:	J	Z		
B 1011			+	;	K			
C 1100			,	<	L			
D 1101	CR		−	=	M			
E 1110			.	>	N	↑		
F 1111			/	?	O	←		

-97-

B MPF-IP FORTH Glossary

B.1 Stack Notation

The first line for each entry describes the execution
of the definition.
(Stack parameters before execution --- Stack parameters
after execution)
In this notation, the top of the stack is to the right.

B.2 Attributes

* C

   The word can only be used in the colon definition.

* I

   It is an immediate word and will be executed
   during compilation unless special action is taken.

* U

   User variable

In the FORTH standard definitions, each word is
assigned a serial number in the range 100 through 999.

B.3 Stack Parameter Definition

* addr, addrl,....   (0....65535)

   Represent the value for one character's address.

* byte (0....255)

   Represent the value of an 8-bit byte.

* char (0....127)

    Represent the value of a 7-bit ASCII code.

* d (-2147483648....2147483647)

    32-bit signed double number.

* flag

    Boolean flag has two logical states: zero = false,
    non-zero = true.

* n (-32768....32767)

    16-bit signed number.

* ud (0....4294967295)

    32-bit unsigned number.

* un (0....65535)

    16-bit unsigned number.

B4 Words

* !   n addr ---   112

    Save n in an address; pronounced "store".

* !CSP

    Save the stack position in CSP; pronounced
    "store CSP".

* #   ud1 --- ud2   158

    Unsigned double number ud1 generates the next-
    output ASCII code. ud2 is the quotient from
    division of ud1 by BASE and reserved for further
    process. Used between <# and #>. Pronounced
    "sharp".

* #>   d --- addr n   190

    Terminate numeric output conversion. It drops d
    and leaves the string address and character count
    n required by TYPE. Pronounced "Sharp-greater".

* #S   ud --- 0 0   209

    Converse all digits of an unsigned double number,
    add it to the numeric output string until the
    remainder equals 0. If the number is originally 0,
    a 0 will add to the output string. The word is only
    used between <# and #>. Pronounced "Sharp-S".

* '   --- addr

    Used in the form:

    '   <name>

    Leave the parameter field address of the next word
    accepted from the input string when executing. In
    compilation, the address is regarded as a literal;
    the value will be placed on the stack in   later
    execution. An error will occur if the word can not
    be found in CONTEXT and FORTH vocabularies. In a
    colon definition, ' <name> is identical to (* '
    <name> *) LITERAL.   Pronounced "tick".

* (I,112

    Use in the form:

    (cccc)

    Accept and ignore the input string until the next
    right parenthesis. As usual, left parenthesis
    must be followed by a blank. It can be used in
    either execution or compilation. An error message
    is displayed if the input string terminates before
    the right parenthesis. Left parenthesis is
    pronounced "paren"; right parenthesis is pronounced
    "close-paren".

* (*   I, 125

    Terminate compilation mode, and execute input
    string context. Pronounced "left-bracket". Refer
    to *).

* (+LOOP)   n ---   c

    A run-time procedure, compiled by +LOOP.

* (.")   c

    A run-time procedure, compiled by ."

* (;CODE)   c

    A run-time procedure, compiled by ;CODE.

* (DO)   c

    A run-rime procedure, compiled by DO; it moves
    loop control parameters to the return stack.

* (ABORT)

    Execute when error occurs and the WARNING is -1.
    Usually, the word executes ABORT. The user may
    change it by a procedure. Refer to ABORT.

* (FIND)

    addr1 addr2 --- addr3 byte flag (found)
    addr1 addr2 --- flag   (not found)

    Search the text at addr1 in the dictionary from
    name field address addr2. If a match is found,
    return the parameter field address addr3, name
    field byte length and a Boolean true. If not
    found, leaves a Boolean false.

* (LINE)

    n1 n2 --- addr n3

    Convert line number n1 and block number n2 to
    pseudo disk buffer address. n3 must equal 32
    indicating length of the entire line.

* (LOOP)   c

    A run-time procedure, compiled by LOOP.

* *   n1 n2 --- n3   138

    Leave the product of n1 times n2; pronounced
    "times".

* *)   126

    Set a compilation mode. The input string text is
    executed immediately. Pronounced "right-bracket".
    Refer to (*.

* */   n1 n2 n3 --- n4   200

    Multiply n1 by n2 and divide the result by n3.
    Leave quotient n4. n4 is the rounded number. Its
    precision is higher than that of n1 n2 * n3 /. The
    product of n1 times n2 is an intermediate 32-bit
    number. Pronounced "times-divide".

* */MOD n1 n2 n3 --- n4 n5   192

    Multiply n1 by n2 and divide the result by n3.
    Leave remainder n4 and quotient n5. As */, the
    intermediate result is a 32-bit number. The sign
    for the remainder is the same as n1. Pronounced
    "times-divide-mod".

* +   n1 n2 --- n3   121

    Plus n1 by n2 and leave the sum n3 on the stack.
    Pronounced "plus".

* +!   n addr ---   157

    Add n to 16-bit number at addr. Pronounced "plus-
    store".

* +-   n1 n2 --- n3

    Assign the sign of n2 to n1 to produce n3.
    Pronounced "plus-minus".

* +BUF

    Execute nothing. Pronounced "plus-buf".

* +LOOP n --- I,C,141

Add loop index to the signed n, and compare the result with the limit. Return to DO to execute until the new index is equal to or larger than the limit (n>0), or until the new index is equal to or smaller than the limit (n<0). When existing loop, drop loop control parameter and continue to execute. Index and limit are singed numbers in the range -32,768 through 32,767. Pronounced "plus-loop". (As conventionally, a negative upper limit is not used.)

* '   n --- 143

Reserve 2 bytes in the dictionary and save n. Pronounced "comma".

* -   n1 n2 --- n3 134

Subtract n2 from n1 and leave the difference n3. Pronounced "minus".

* -->   I

Continue to interpret next screen. Pronounced "next screen".

* -FIND

```
--- pfa b tf (found)
--- ff        (not found)
```

Accept a next text word transferred to HERE from the input stream. Search the same input character in CURRENT from CONTEXT. If found, pfa, length b and true flag are left on the stack; otherwise, a false flag is left.

* -TRAILING 148

addr n1 --- addr n2

Adjust the character count of a text (starting from addr), and remove the trailing blanks, that is, blanks from addr + n2 to addr + n1 - 1. If n1 is negative, an error message is displayed. Pronounced "dash-trailing".

* .   n --- 193

Display n converted from BASE as a single number, followed by a blank. Print a minus sign if it is a negative number. Pronounced "dot".

* ."   I,133

Interpreted or used in a colon-definition in the form:

." cccc"

Accept following text from the input string, terminated by ASCII " (double - quote). In executing, move the text to a selected output device. In compiling, compile it so that the later execution may move the text to a selected output device. At least 127 bytes are allowed for the text. An error message is displayed if input stream stops before the terminating " . Pronounced "dot-quote".

* .CPU

Print the name of CPU (Z80).

* .LINE   n1 n2 ---

Display the text of line number n1 and block number n2.

* .R   n1 n2 ---

Print number n1 in a field of width n2 right justified. No following blank is printed.

* .S   ---

A non-destructive stack printing word used to print current contents of the parameter stack.

* /   n1 n2 --- n3 178

Divide n1 by n2 and remain quotient n3. n3 is rounded toward zero. Pronounced "divide".

* /MOD   n1 n2 --- n3 n4   198

    Divide n1 by n2 and leave the remainder n3 and
    quotient n4. The sign for n3 is as same as n1.
    Pronounced "divide-mod".

* 0 1 2 3   --- n

    These small numbers are used frequently. It is
    necessary to define them as constants.

* 0<   n --- flag   144

    If n < 0, return a true flag. Pronounced "zero-
    less".

* 0=   n --- flag   180

    If n = 0, return a true flag. Pronounced "zero-
    equals".

* 0>   --- flag   118

    If n > 0, return a true flag. Pronounced "zero-
    greater".

* 0BRANCH   flag ---   c

    Execute procedure branches conditionally. If the
    flag is not true, the parameter will be added to
    the interpretive pointer, and branches towards or
    backwards. Compiled by IF, UNTIL, and WHILE.

* 1+   n --- n+1   107

    Add 1 to n according to + operation. Pronounced
    "one-plus".

* 1-   n --- n-1

    Subtract 1 from n according to - operation.
    Pronounced "one-minus".

* :   116

    A definition word, used in the form:

    : <name> ... ;

    Select CONTEXT vocabulary to be identical to
    CURRENT. Build a word <name> in CURRENT and set a
    compile mode. We call it a colon-definition. The
    compiling address of subsequent words (excluding
    immediate words) is saved in the dictionary. When
    <name> is executed , the words in the definition
    will be executed. The immediate word is executed
    immediately. If a word can not be found in CONTEXT
    and FORTH vocabularies, it is regarded as a literal
    for conversion and compilation (using the current
    base). An error message is displayed if failed
    again. Pronounced "colon".

* ;   I,C,196

    Terminate a colon-definition and stop the
    compilation. An error message is displayed if input
    stream terminates before encountering ; .
    Pronounced "semi-colon".

* ;CODE   C,I,206

    Use in the form:

    ; <name> ... ;CODE

    Stop compilation and terminate the definition of
    the word <name>. It is used to define the new word
    <namex> when <name> is later executed in the form:
    <name>, <namex>. The executing address for <namex>
    is included the address after ;CODE in <name>. If
    executing any <namex>, these sequence of machine
    code is executed. Pronounced "semi-colon-code".

* ;INT   C,I

    Used in the form:

    : <name> .... ;INT

    Stop compilation and terminate definition of an
    interrupt handling word <name>.

* ;S   I

    Terminate interpretation of a screen. ;S is a run-
    time word compiled after the colon definition
    which returns execution to the calling procedure.

* <   nl n2 --- flag   139

    It is true if nl < n2. Pronounced "less-than".

* <#   169

    Begin to convert numbers to output format. The
    following word

    <# # #S HOLD SIGN #>

    points out conversion of a double number ASCII code
    string and save it from right to left. Pronounced
    "less-sharp".

* <COMPILE>   I,C,17

    Used in colon-definition in the following form:

    <COMPILE> <name>

    Enforce compilation of the following words. It can
    compile an immediate word to prevent it from being
    executed. Pronounced "bracket-compilation".

* =   nl n2 --- flag   173

    If nl is equal to n2, it is true. Pronounced
    "equals".

* >   nl n2 --- flag   102

    If nl is larger than n2, it is true. Pronounced
    "greater-than".

* >IN   --- addr   U,201

    Leave a variable's address. The variable contains
    the current character offset of input stream in the
    range 0 through 1023. Pronounced "to-in". Refer to
    WORD (." FIND

* >R   n ---   C,200

    Move n to return stack. In a colon definition, each
    >R must be accompanied with another R>.

* ?   addr ---   194

    Display numbers at the address, using the same
    format as . (dot). Pronounced "question-mark".

* ?COMP

    An error message is displayed if not in compile
    mode.

* ?CSP

    An error message is displayed if stack location is
    different from value in CSP.

* ?DUP   n --- n (n)   184

    Copy n if n does not equal 0. Pronounced "query-
    dup".

* ?ERROR   flag n ---

    If Boolean flag is true, print the nth error
    message.

* ?EXEC

    An error message is sent out if not in execution
    mode.

* ?LOADING

    An error message is sent out if not loading.

* ?PAIRS   nl n2 ---

    Error message #19 is sent out if nl does not equal
    n2, which means that some conditional control is
    illegal in compiling.

* ?TERMINAL   ---flag

    Testing any key on the keyboard. A true flag means
    it is operated. This definition is related to the
    devices.

* ?STACK   ---

    An error message is sent out if the data stack
    exceeds the limit.

* @   addr --- n   199

    Leave number at the addr on the stack. Pronounced
    "fetch".

* ABORT   101

    Clear the data and return stack, set execution
    mode. Return control to the terminal.

* ABS   n1 --- n2   108

    Leave the absolute value of a number on the stack.
    Pronounced "absolute".

* ALLOT   n ---   154

    Add n bytes of spaces to parameter field of words
    most recently defined.

* AND   n1 n2 --- n3   183

    Leave the result of the logical AND of n1 and n2 on
    the stack.

* AGAIN

    Used in colon definition:

    BEGIN....AGAIN

    Execute words between BEGIN and AGAIN infinitely.

* B/BUF   --- n

    A constant, which leaves number of bytes in each
    buffer on the stack. That is, the bytes count read
    from mass storage by BLOCK.

* B/SCR   --- n

    A constant used to leave the number of blocks in
    each   screen on the stack.

* BACK   addr ---

    In run-time procedure, count the branching offset
    from HERE to addr. Move the offset to the next
    effective address in the dictionary.

* BASE   --- addr   U,115

    Leave the address of a variable on the stack, in
    which the conversion base for numeric input/output
    is stored. The range of the variable is 2 to 70.

* BEGIN   I,C,147

    Used in colon-definition:

    BEGIN ..... AGAIN   or

    BEGIN...FLAG UNTIL   or

    BEGIN...FLAG WHILE...REPEAT

    BEGIN indicates the start of a series of repeatedly
    executed words. BEGIN ... UNTIL repeats until the
    flag is true. BEGIN ... WHILE ...REPEAT repeats
    until the flag is false. When loop finishes, words
    after UNTIL and REPEAT are executed. The flag is
    dropped after testing. BEGIN ... AGAIN constitutes
    an infinite loop.

* BL   --- char   176

    A constant which leaves the ASCII code for "blank"
    on the stack.

* BLANKS   addr n ---

    Fill the   n consecutive memory locations   starting
    from addr with ASCII codes for "blank".

* BLK   --- addr   U,132

    Leave the address of a variable on the   stack.   The
    address saves blocks count in the mass storage and
    regarded as input stream. If the content is Ø,
    input stream is taken from the terminal. Pronounced
    "b-l-k".

* BLOCK   n --- addr

    Leave   the   address of the first byte of the nth
    block on the stack.

* BRANCH   C

    The run-time procedure for unconditional branch. An
    in-line offset is added to IP (interpreter pointer)
    for branching forward or backward.   BRANCH is
    compiled by ELSE, AGAIN, REPEAT.

* BOUNDS   addr n --- addr+n addr

    Convert addr and n to start and end addresses to be
    used by loop.

* BUFFER   n --- addr

    Same as BLOCK.

* C!   n addr ---   219

    Save low order byte of n at the   addr. Pronounced
    "c-store".

* C,   byte ---

    Save the 8-bit character to the next   usable
    dictionary   character.   The   dictionary   pointer
    increments by 1.

* C/L --- n

    A constant which leaves the number of characters in
    each line of the source text 32 on the stack.

* C@   addr --- byte   156

    Leave the contents of the character at the addr on
    the stack. (In 16-bit field, MSC is Ø) Pronounced
    "c-fetch".

* CALL   addr ---

    Transmit control to the machine code subroutine
    (The address is on the stack). The registers are
    input from a reserved memory and saved.

* CFA   addr1 --- addr2

    Convert addr1 (parameter field address) to addr2
    (code field address).

* CMOVE   153

    addr1 addr2 n ---

    Move n bytes starting from addr1 to addr2. The
    contents of addr1 is first moved toward high memory
    address. If n is equal to or less than Ø, nothing
    is moved. Pronounced "c-move".

* CODE   111

    Used in the form:

    CODE <name>... END-CODE

    Build a dictionary word <name>, which is defined by
    following assembly language words.

* COLD

    Cold start procedure used to adjust dictionary
    pointer to the minimum standard and reinitiated by
    ABORT.

* COMPILE   C,146

   When a word containing COMPILE is executed, 16 bits
   after COMPILE's compilation address is copied or
   compiled into the dictionary, that is, COMPILE DUP
   copies DUP's compilation address.

* CONSTANT   n ---   185

   Used in the form:

   n CONSTANT <name>

   Build a word <name> and leave n in its parameter
   field. n is left on the stack when <name> is
   executed later.

* CONTEXT   ---addr   U,151

   Leave on the stack the address of the variable
   pointing out the vocabulary in which dictionary
   search is to be made during interpretation of input
   stream.

* CONVERT   dl addr1 --- d2 addr2   195

   Convert text starting from addr1+1 to its
   corresponding stack number with regard to BASE. The
   new value is added into dl and the result is left
   as d2. addr2 is the first non-convertible byte.

* COUNT   addr --- addr+1 n   159

   Leave the address (addr+1) for the text starting
   from addr, and number of characters on the stack.
   The first byte on the addr must contain the number
   of characters n. The range for n is 0 through 255.

* CR   160

   Cause a carriage-return and line-feed to the
   current output device. Pronounced "c-r".

* CREATE   239

   Used in the form:

   CREATE <name>

   Build a word <name>. No parameter field memory is
   reserved. When <name> is executed later,the address
   of the first character of <name>'s parameter field
   will be left on the stack.

* CSP   --- addr   U

   A user variable, saving stack pointer location
   temporarily for checking compilation error.

* CURRENT   --- addr   U,137

   Leave on the stack address of the variable which
   points out into what vocabulary a new word
   definition is to be compiled into.

* D+   dl d2 --- d3   241

   Leave the sum of dl and d2, d3 on the stack.
   Pronounced "d-plus".

* D.R   d n ---

   Display d which is converted according to BASE, in
   a n-character field, right-justified. If it is
   negative, then the minus sign will be printed.
   Pronounced "d-dot-r".

* DABS dl --- d2

   Leave dl's absolute value d2 on the stack. The
   range is 0 through 2,147,483,647. Pronounced "d-
   abs".

* DECIMAL   197

   Set base for numeric input/output to 10.

* DEFINITIONS   155

   Set CURRENT as the CONTEXT vocabulary so that later
   definition will be built in the vocabulary
   previously   selected by CONTEXT.

* DEPTH   ---n   238

   Leave number of 16-bit numbers on the data stack
   on the stack (n is not counted in).

* DI

   Disable interrupt.

* DIGIT

   char n1 --- n2 true-flag (transferable)

   char n1 --- false-flag   (non-transferable)

   Use base n1, convert char into its binary
   equivalent and followed by a true flag. If not,
   leave a false flag.

* DLITERAL   I

   d --- d   (executing)

   d ---   (compiling)

   If compiling, interpret a stack double number to
   literal; later execution including the literal
   will push it to the stack. If executing, the number
   will remain on the stack.

* DNEGATE   245

   d1 ---   -d1

   Leave a double number's two's complement on the
   stack.

* DO   n1 n2 ---   I,C,142

   Used in the form:

   DO...LOOP   or

   DO...+LOOP

   Start a loop which terminates according to control
   parameter. The loop index starts from n2 and
   terminates at n1. The index will increment by a
   signed value at LOOP or +LOOP. DO...LOOP's range is
   determined by terminating words. DO...LOOP can be
   nested. A standard system contains at least three
   levels of nesting.

* DOES>   I,C,168

   Define the run-time action of a word built by a
   high level definition word.

   Used in the form:

   : <name> ... CREATE...DOES>...;
     and then <name> <namex>

   Point out termination of a word <name> and begin
   the definition of the run-time action for words
   that will be defined by <name> later. If executing
   <namex>, words between DOES> and ; are executed.
   The address of <namex>'s parameter field remains on
   the stack. Pronounced "does".

* DP   --- addr   U

   A user variable, the dictionary pointer, which
   contains the address of the next usable memory
   location in the dictionary. The value can be read
   from HERE and changed by ALLOT. Pronounced "d-p".

* DPL   --- addr

   A user variable that saves number of digits to the
   right of the decimal point when double number is
   input.

* DROP n ---   233

   Drop the top number on the stack.

* DUMP   addr n ---

    Print n bytes starting from addr, 4 bytes on   each
    line.

* DUP   n --- n n

    Leave a copied stack top number.

* EDITOR

    Select editor vocabulary as context vocabulary.

* ELSE   I,C,167

    Used in colon definition:

    IF...ELSE...THEN

    If the IF result is true, execute the part between
    IF and ELSE, otherwise that between ELSE and THEN
    is executed.

* EMIT   char---   207

    Transmit the character to the current output
    device.

* EMPTY-BUFFERS

    Clear memory between FIRST and LIMIT - 1.

* ENCLOSE

    addr1 char --- addr1 n1 n2 n3

    Use char as a delimiter. Scan text starting from
    addr1. Three offsets are returned on the stack. n1,
    the   byte offset to the   first   non-delimiter
    character; n2, the offset to the first delimiter
    after the text, and n3, the offset to the character
    not included. The procedure regards NULL's ASCII
    code as unconditional delimiting character. ENCLOSE
    is the primitive for scanning text used by WORD.

* END   flag ---

    Same as UNTIL.

* END-CODE

    Terminate a word definition.   Set the CONTEXT
    vocabulary as CURRENT again. This word definition
    can be used if no error message is displayed.

* ENDIF

    Same as THEN.

* EI

    Enable interrupt.

* ERASE addr n ---

    Clear the n bytes starting from addr.

* ERROR   n--- >n2 n3

    Send out an error message to enter into FORTH
    system again. Leave >IN and BLK on the stack as n2
    and n3 respectively to specify the origin of error.

* EXECUTE   addr ---   163

    Execute a dictionary word. Its compilation address
    is on the stack.

* EXIT   C,117

    Terminate a definition's execution if compiled
    in colon definition. It can not be used in
    DO...LOOP.

* EXPECT   addr n ---   189

    Receive characters from keyboard and transmit them
    to the memory range starting from addr until a
    "return" or the count of n is received. If n < 0 or
    equals to 0, no action occurred. Add one or two
    nulls after the text.

* FENCE   ---addr   U

    A user variable contains an address. The content
    below the address allows no FORGET. The user must
    change the contents of FENCE to forget the contents
    below.

* FILL addr n byte ---   234

   Fill n bytes in the memory starting from addr with
   byte. If n < or = 0, no action occurred.

* FIRST ---addr

   A constant that leaves the first (lowest) block.
   buffer' address on the stack.

* FLD   ---addr

   A user variable that saves a field width of output
   format for numbers.

* FLUSH

   No execution.

* FORGET   186

   Used in the form:

   FORGET <name>

   Delete from the dictionary <name> (in CURRENT
   vocabulary) and the following words. If <name> can
   not be found in CURRENT or FORTH, an error will
   occur.

* FORTH   I,187

   The name of the primary vocabulary. FORTH becomes
   CONTEXT vocabulary upon execution. A new definition
   will be a part of FORTH until a different CURRENT
   vocabulary is built. A user vocabulary is chained
   to FORTH vocabulary upon conclusion and so FORTH is
   considered   to   be   contained in each   user
   vocabulary.

* HERE --- addr   188

   Return the address of the next usable dictionary
   location.

* HEX   162

   Set input/output numbers conversion base to 16
   (hexadecimal).

* HLD   ---addr

   A user variable that contains the last byte's
   address of text in number output conversion
   procedure.

* HOLD char---   175

   Insert char to number output stream. It can only
   be used between <# and #>.

* I   --- n   C,136

   Copy loop index to the data stack. It can only be
   used in DO-LOOP.

* ID.   addr ---

   Print a definition name from the name field
   address.

* IF   flag---   I,C,210

   Used in colon definition:

   flag IF.. ELSE... THEN
   flag IF...THEN

   If the flag is true, execute the words after   IF,
   and the words following ELSE are skipped. The ELSE
   part is optional. If flag is false, words between
   IF and ELSE, or between IF and THEN (when no ELSE
   is used), are skipped IF-ELSE-THEN can be nested.

* IMMEDIATE   103

   Indicate the most recently built dictionary entry
   as a word which will be executed when encountered
   in compiling and not compiled.

* IM0

   Select interrupt mode 0.

* IM1

   Select interrupt mode 1.

* IM2

    Select interrupt mode 2.

* INDEX n1 n2 ---

    Print the first line of each screen over the range
    n1 to n2. This is used to inspect comment lines of
    a number of text screens.

* INTERPRET

    The outer text interpreter which sequentially
    executes or compiles text from the input stream
    (terminal or mass storage) depending on STATE. If
    the word name cannot be found after a search of
    CONTEXT and then FORTH, it is converted to a number
    according to the current base. That search also
    failing, an error message echoing the name with a
    "?" will be given. Text input will be aken
    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.

* INTFLAG   --- addr

    A user variable that saves an interpretive
    interrupt flag.

* INTVECT   --- addr

    A user variable that saves CFA of an interpretive
    interrupt handling word.

* J   --- n   C,225

    Return the outer loop index to the stack. Used only
    in the form:

        DO...DO...J...LOOP...LOOP

* KEY   ---char   100

    Leave the ASCII code of the next usable character
    from current input device on the stack.

* LATEST   ---addr

    Leave the top-most word's name field address in
    CURRENT vocabulary on the stack.

* LEAVE   C,213

    Set the loop limit to be the same as the current
    index to terminate DO - LOOP at the next LOOP or
    +LOOP. The index itself does not change and
    execution will continue normally until the
    terminating word is encountered.

* LFA   addr1---addr2

    Convert addr1 (parameter field address of a
    definition) to addr2 (link field address).

* LIMIT --- n

    A constant that leaves the highest memory location
    address of a block buffer on the stack.

* LIST n ---   109

    Print the ASCII contents of screen n on the current
    output device. Set SCR to n, a unsigned number.

* LIT   --- n   C

    In colon definition, LIT is automatically compiled
    before each 16-bit literal encountered in input
    text is compiled. Later execution of LIT will push
    the contents of the following two bytes on the
    stack.

* LITERAL n ---   I,215

    In compilation, regard stack value n as 16-bit
    literal; n will remain on the stack if later
    executed.

* LOAD n ---   202

    Regard screen n as input stream for interpretation;
    reserve current input stream (>IN and BLK)
    locators. If interpreter is not terminated
    explicitly, it will be terminated when input stream
    exhausts. Control returns to input stream
    containing LOAD, determined by >IN and BLK.

* LOOP   I,C,124

   Increment the DO-LOOP index by 1. The loop is
   terminated if new index value equals or is larger
   than limit. Limit and index are signed numbers. The
   range is -32,768 through 32,767.

* M*   n1 n2 --- d

   A mixed arithmetic operation word, which leaves d,
   the product of n1 times n2 on the stack.

* M/   d n1 --- n2 n3

   A mixed arithmetic operation word which leaves on
   the stack remainder n2 and quotient n3 of the
   division of d by n1. The sign of the remainder is
   the same as the dividend.

* M/MOD   ud1 un2 --- un3 ud4

   An unsigned mixed arithmetic operation word, which
   leaves quotient un4 and remainder un3 that un1
   divides un2 on the stack.

* MAX   n1 n2 --- n3   218

   Leave the larger number of n1, n2 on the stack.
   Pronounced "max".

* MESSAGE n ---

   If WARNING = 0, send out MSG# n. If WARNING = 1,
   send out the text of the nth line in the 4th block
   of pseudo disk.

* MIN   n1 n2 --- n3   127

   Leave the smaller number on the stack. Pronounced
   "min".

* MOD   n1 n2 --- n3   104

   Divide n1 by n2. The sign of the remainder n3 is
   the same as n1. Pronounced "mod".

-124-

* MON

   Jump to MPF-IP monitor program.

* NEGATE   n --- -n   177

   Leave two's complement of a number on the stack,
   that is, the difference of 0 and n.

* NEXT

   Used in low level definition:

   CODE <NAME> .... NEXT END-CODE

   Compile directly the word that jumps to FORTH inner
   interpreter (JP 2078) in the dictionary.

* NFA   addr1 --- addr2

   Convert parameter field address of a definition
   addr1 to the name field address addr2.

* NOT   .   flag1 --- flag2   165

   Convert flag1's boolean value. Same as 0=.

* NUMBER   addr --- d

   Convert the string at addr with a preceding count
   to a double number.

* OFFSET   --- addr

   A user variable that saves pseudo disk memory's
   offset. Adding the offset to get the desired
   address when BLOCK is executed.

* OR   n1 n2 --- n3   223

   Leave the result n3 from OR operation of n1 and n2
   on the stack.

* OUT   --- addr

   A user variable that saves output buffer's offset.
   The value of OUT increments each time EMIT is
   executed.

-125-

* OVER   n1 n2 --- n1 n2 n1   170

     Copy the second number on the top of stack.

* P!   n addr ---

     Transmit data n to I/O port addr.

* P@   addr --- n

     Input data n from I/O port addr.

* PAD   --- addr   226

     Save the scratch area's address of the intermediate character string. Its minimum capacity is 64 bytes (from addr to addr + 63).

* PFA   addr1 --- addr2

     Convert name field address of a definition addr1 to its parameter field address addr2.

* PICK   n1 --- n2   240

     Return the value of n1th number n2 on the stack. An error will occur if n1 is smaller than 1. 2 PICK is the same as OVER. (1 ... n).

* PREV   --- addr

     A user variable.

* QUERY   235

     Accept input (80 bytes at most) from the keyboard or when a "return" is encountered, store into the terminal input buffer. Using WORD to set >IN and BLK to 0. So that the buffer may accept text as input string.

* QUIT   211

     Clear return stack, set execution mode, and return control to terminal. No message is given.

* R#   --- addr

     A user variable that saves the position of cursor in the screen during compiling.

* R/W   addr n flag ---

     Delete three items on the stack. No other action is taken.

* R>   --- n   C,110

     Transfer n from the return stack to the data stack. Pronounced "r-from".

* R0   --- addr

     A user variable that saves the initial value of the return stack pointer.

* R@   --- n   C,228

     Copy the top value of the return stack to the data stack. Pronounced "r-fetch".

* RA   --- addr

     A user variable, temporary address for register A.

* RAF   --- addr

     A user variable, temporary address for register pair AF.

* RAF'   --- addr

     A user variable, temporary address for register pair AF'.

* RB   --- addr

     A user variable, temporary address for register B.

* RBC   --- addr

     A user variable, temporary address for register pair BC.

* RBC'   --- addr

   A user variable, temporary address for register pair BC'.

* RC   --- addr

   A user variable, temporary address for register C.

* RD   --- addr

   A user variable, temporary address for register D.

* RDE   --- addr

   A user variable, temporary address for register pair DE.

* RDE'   --- addr

   A user variable, temporary address for register pair DE'.

* RE   --- addr

   A user variable, temporary address for register E.

* REPEAT   I,C,120

   Used in colon definition in the following form:

   BEGIN... WHILE... REPEAT

   REPEAT returns control to just after its corresponding BEGIN in execution.

* RF   --- addr

   A user variable, temporary address for register F.

* RH   --- addr

   A user variable, temporary address for register H.

* RHL   --- addr

   A user variable, temporary address for register pair HL.

* RHL'   --- addr

   A user variable, temporary address for register pair HL'.

* RL   --- addr

   A user variable, temporary address for register L.

* RIX   --- addr

   A user variable, temporary address for register pair IX.

* RIY   --- addr

   A user variable, temporary address for register pair IY.

* ROLL   n ---   236

   Extract the nth number to the top of the stack and move the remaining values to the vacated locations. (1...63).

   1 ROLL : NO OPERATION
   2 ROLL = SWAP
   3 ROLL = ROT

* ROT   n1 n2 n3 --- n2 n3 n1   212

   Rotate the top three numbers on the stack. Place the third to the topmost. Pronounced "rote".

* RP@   --- addr

   Return the address of return stack location to the top of parameter stack as it was before execution of RP@. Pronounced "r-p-fetch".

* RP!

   Set initial value of the return stack pointer. This is a computer-dependent procedure.

* S->D   n --- d

    Change a single number to a double number.

* S0   --- addr

    A user variable that saves initial value of data
    stack pointer.

* SCR   --- addr   U,217

    Leave the address of a variable which contains the
    number of the screen most recently listed on the
    stack. Pronounced "s-c-r".

* SIGN   n ---   C,140

    Insert ASCII "-" into number output string if n is
    negative.

* SMUDGE

    Change smudge bit of the name field address when
    defining a new word to validate the definition.

* SP!

    Set initial value of the stack pointer. Pronounced
    "s-p-store".

* SP@   --- addr   214

    Return address of the stack location to the top of
    the stack, as it was before execution of SP@.
    Pronounced "s-p-fetch".

* SPACE   232

    Send ASCII "blank" to the current output device.

* SPACES n ---   231

    Send n spaces to the current output device if n >
    0, otherwise no action is taken.

* STATE   --- addr   U,164

    Leave the address of a variable which contains
    compilation condition on the stack. The compilation
    begins if the content does not equal 0.

* SWAP   n1 n2 --- n2 n1   230

    Exchange the top two numbers on the stack.

* TASK

    A dummy word.

* THEN   I,C,161

    Used in colon definition in the following form:

    IF...ELSE...THEN or IF...THEN

    THEN must follow ELSE or IF.

* TIB   --- addr

    A user variable containing address of terminal
    input buffer.

* TOGGLE addr byte ---

    Use bit pattern byte to complement the contents of
    addr.

* TRAVERSE   addr1 n --- addr2

    Move across name field address of a variable. addr1
    is either address of length byte or the address of
    the last byte. The motion is toward high memory
    address if n = 1; toward low memory address if n =
    -1. addr2 is address of the other end of the name
    field address.

* TREAD

    Read data of a file from the tape.

* TWRITE   n1 n2 ---

   Save data between blocks n1 and n2 in a file on the
   tape.

* TYPE   addr n ---   222

   Send n bytes starting from addr to the current
   output device if n > 0.

* U*   un1 un2 --- un3   242

   Multiply un1 by un2 and leave the product un3. All
   numbers are unsigned. Pronounced "u-times".

* U.   un ---   106

   Convert un according to BASE as an unsigned number
   and print it in a free-field format, with one
   trailing blank. Pronounced "u-dot".

* U/MOD   ud1 un2 --- un3 un4   243

   Divide ud1 by un2. The remainder is un3; the
   quotient   is un4. All values are   unsigned.
   Pronounced "u-divide-mod".

* U<   un1 un2 --- flag   150

   Leave flag after comparison of un1 and un2. Un1 and
   un2 are 16-bit unsigned numbers. Pronounce "u-less-
   than".

* U?TERMINAL --- addr

   A user variable that saves ?TERMINAL's CFA.

* UABORT   --- addr

   A user variable that saves ABORT's CFA.

* UB/SCR   --- addr

   A user variable that saves block number of blocks
   in each screen.

* UC/L   --- addr

   A user variable that saves number of bytes in   a
   line.

* UCR   --- addr

   A user variable that saves the word CR's CFA.

* UEMIT   --- addr

   A user variable that saves EMIT's CFA.

* UFIRST   --- addr

   A user variable that saves FIRST's value.

* UKEY   --- addr

   A user variable that saves KEY's CFA.

* ULIMIT   --- addr

   A user variable that saves LIMIT's value.

* UNTIL   flag ---   I,C,237

   Used in a colon definition that indicates the   end
   of BEGIN-UNTIL loop. The loop ends if the flag is
   true. The execution returns to the first word after
   BEGIN. BEGIN ... UNTIL may be nested.

* UPDATE

   Executing nothing.

* UR/W   --- addr

   A user variable that saves R/W's CFA.

* USE   --- addr

   A user variable.

* USER   n ---

   A defining word, used in the form:

n USER <name>

Build a user varialbe <name>. n in parameter
field is the offset relative to the user area
pointer. The real address can be obtained from n.
(offset + starting address of user area.)

* VOC-LINK   --- addr

A user variable that contains a field address of
newly   built vocabulary. All vocabulary
names are linked to these fields.

* VARIABLE   227

A defining word, used in the form:

VARIABLE <name>

Build dictionary word <name> and reserve two bytes
of memory locations in parameter field. The
initial value must be set in use. It will put the
memory address on the stack when <name> is executed
later.

* VLIST

Print names of definitions in CONTEXT vocabulary.
Pressing any key will terminate printing.

* VOCABULARY   208

A defining word, used in the form:

VOCABULARY <name>

Build (in current vocabulary) a dictionary word
<name>, which points out a table of order for new
words definitions. It will become a CONTEXT
vocabulary when <name> is later executed. A new
definition will be listed in word table when <name>
becomes CURRENT vocabulary. the new vocabulary is
chained to FORTH, that is, searching FORTH
vocabulary after searching a vocabulary.

* WARM

Reset initial values of variables S0, R0, TIB,
WIDTH, WARNING, FENCE, and then enter into ABORT.
The word does not influence the created words in
the dictionary.

-134-

* WARNING   --- addr

A user variable that saves value for controlling
error message output and execution procedure in
occurrence of error.

* WHILE   flag ---   I,C,149

Used in colon definition:

BEGIN...flag WHILE...REPEAT

Select conditional execution according to flag. If
the flag is true, execute until REPEAT (it will
return to the words after BEGIN). If the flag is
false, exit the construction and execute words
after REPEAT.

* WIDTH   --- addr

A user variable that contains the maximum number of
characters reserved in compiling names of defini-
tions. The range is from 1 to 31. Number of
characters for names and the original characters
are reserved according to WIDTH's value. The value
can be freely changed between 1 and 31.

* WORD   char --- addr   181

Accept characters from input string until non-
zero delimiter is encountered or the whole strin is
input. These characters are saved as packed string.
The total number is at the address of the first
byte. The delimiting character encountered is saved
at the end of the text. Its length is 0 if input
string terminates. The starting address of the
packed string remained on the stack.

* XOR   n1 n2 --- n3   174

Leave results of bitwise XOR operation of n1 and
n2. Pronounced "x-or".

-135-

B.5 Double Number Words

* **2!** d addr ---

  Save d in 4 consecutive bytes starting from addr. Pronunced "two-store".

* **2@** addr --- d

  Place 4 consecutive bytes starting from addr on the stack. Pronounced "two-fetch".

* **2DROP** d ---

  Drop double numbers on the top of the stack. Pronounced "two-drop".

* **2DUP** d --- d d

  Duplicate the top double number on the stack Pronounced "two-dup".

* **2OVER** d1 d2 --- d1 d2 d1

  Copy the second double number of the stack to the top. Pronounced "two-over".

* **2SWAP** d1 d2 --- d2 d1

  Exchange the top two double numbers of the stack. Pronounced "two-swap".

* **D+** d1 d2 --- d3   241

  Add d1 to d2 and leave the sum d3 on the stack. Pronounced "d-plus".

* **D+-** d1 n --- d2

  Add sign of n to d1; d1 becomes d2 and d2 is left on the stack. Pronounced "d-plus-minus".

* **D.** d ---   246

  Print d converted according to BASE in free-field, followed by a space. Pronounced "d-dot".

* **D.R** d n ---

  Print d converted according to BASE in n-byte field, right justified. Pronounced "d-dot-r".

* **D<** d1 d2 --- flag   244

  The rlag is true if d1 < d2. Pronounced "d-less".

* **DABS** d1 --- d2

  Leave d1's absolute value on the stack.

* **DNEGATE** d --- -d   245

  Leave two's complement of a double number, that is, the difference of Ø and -d on the stack. Pronounced "d-negate".

B.6 Editing Words

* **Control-I** n ---

  Move cursor n bytes. Control-I means that pressing control and I immediately.

* **-TEXT** addr1 n addr2 --- f

  Return a true flag if the first n bytes in the two strings starting from addr1 and addr2 are the same. Otherwise, a false flag is returned.

* **CLEAR** n ---

  Clear the n-th block.

* **COPY** n1 n2 ---

  Copy data in block n1 to n2.

* **D**

  Used in the form:

  D xxx

  Delete string xxx after the cursor.

* F

    Used in the form:

    F   xxx

    Search string xxx from behind the cursor, and place
    the cursor after the found string. The cursor
    returns to the beginning of line 0 if the target
    string is not found.

* I

    Used in the form:

    I   xxx

    Insert the string to the cursor's current location.

* L

    Reprint text in the current block.

* LINE   n --- addr

    Place the address of the first byte of the nth line
    in the current block on the stack.

* MATCH   addr1 n1 addr2 n2 --- f n3

    The text to be searched begins at addr1 and is n1
    bytes long. The string to be matched begins   at
    addr2 and is n2 bytes long. The boolean flag   is
    true if a match is found, n3 is then the   cursor
    advancement to the end of the found string. If   no
    match is found, f will be false and n3 be n1.

* N

    Search text saved in PAD.

* P

    Used in the form:

    P   xxx

    Place the string xxx at the line the cursor
    is.

* T   n ---

    Print the nth line and move the cursor to the
    beginning of the line.

* TEXT   c ---

    Use the character c as the delimiter to move a text
    string to PAD from input buffer. If the string
    contains less than 64 characters, it will be padded
    with blanks to make a total of 64 characters.

* TILL

    Used in the form:

    TILL xxx

    Delete text between the cursor and xxx.

* TOP

    Move the cursor to the beginning of line 0.

* U

    Used in the form:

    U   xxx

    Insert xxx under the current line, the subsequent
    lines are moved below one line.

* X

    Delete the current line. Deleted text is saved in
    PAD. Pad the last line with blank.

C MPF-IP FORTH
Error Messages

MSG#	REASONS
Ø	Not existing.
1	Data stack empty.
2	Dictionary space is full.
4	Words name defined more than once.
7	Data stack is full.
17	The word can not be used outside the definition.
18	The word can only be executed immediately, can not be used in the definition.
19	Unpaired conditional.
20	The definition is not finished.
21	The word is in the dictionary protected range, can not be deleted.
22	The word can only be used in LOAD.
24	Vocabulary error.

D User Area RAM Map

Address (Hex)	Number of Bytes	Name	Description
FE00-FE05	6	-	Available for user
FE06-FE07	2	S0	Initial value of the data stack pointer
FE08-FE09	2	R0	Initial value of the return stack pointer
FE0A-FE0B	2	TIB	Address of the terminal input buffer
FE0C-FE0D	2	WIDTH	Number of letters saved in names
FE0E-FE0F	2	WARNING	Error message control number
FE10-FE11	2	FENCE	Dictionary FORGET protection point
FE12-FE13	2	DP	The dictionary pointer
FE14-FE15	2	VOC-LINK	Most recently created vocabulary
FE16-FE17	2	BLK	Current block number under interpretation
FE18-FE19	2	>IN	Byte offset within the current input text buffer
FE1A-FE1B	2	OUT	Offset in the text output buffer

FE1C-FE1D	2	SCR	Screen number most recently referenced by LIST
FE1E-FE1F	2	OFFSET	Block offset for disk drives
FE20-FE21	2	CONTEXT	Pointer to the vocabulary within which dictionary search will first begin
FE22-FE23	2	CURRENT	Pointer to the vocabulary within which new definitions are to be added
FE24-FE25	2	STATE	Contains the state of compilation
FE26-FE27	2	BASE	Current I/O base
FE28-FE29	2	DPL	Number of digits to the right of the decimal point on double integer input
FE2A-FE2B	2	FLD	Field width for formatted number output
FE2C-FE2D	2	CSP	Check stack pointer
FE2E-FE2F	2	R#	Location of editor cursor in a text block
FE30-FE31	2	HLD	Address of current output
FE32-FE37	6	FLAST	FORTH vocabulary data initialized to FORTH vocabulary
FE38-FE3D	6	ELAST	Editor vocabulary data initialized to EDITOR vocabulary
FE3E	1	CRFLAG	Carriage return flag
FE3F	1	-	Available for user

FE40-FE42	3	PAT	I/O port fetch routine (input)
FE43-FE45	3	PST	I/O port store routine (output)
FE46-FE47	2	RPP	Return stack pointer
FE48-FE49	2	USE	Mass storage buffer address to use
FE4A-FE4B	2	PREV	Mass storage buffer address just used
FE4C	1	INTFLAG	Interrupt flag
FE4D	1	-	Available for user
FE4E-FE4F	2	INTVECT	Interrupt vector
FE50-FE51	2	U?TERMINAL	Code field address of word ?TERMINAL
FE52-FE53	2	UKEY	Code field address of input word KEY
FE54-FE55	2	UEMIT	Code field address of output word EMIT
FE56-FE57	2	UCR	Code field address of word CR
FE58-FE59	2	UR/W	Code field address of word R/W
FE5A-FE5B	2	UABORT	Code field address of word ABORT
FE5C-FE5D	2	UC/L	Number of characters per input line
FE5E-FE5F	2	UFIRST	Start of pseudo disk buffer
FE60-FE61	2	ULIMIT	End of pseudo disk buffer
FE62-FE63	2	UB/BUF	Number of bytes per block

FE64-FE65	2	UB/SCR	Number of buffers per block
FE66-FE67	2	-	Available for user
FE68-FE69	2	RAF	Register AF
FE6A-FE6B	2	RBC	Register BC
FE6C-FE6D	2	RDE	Register DE
FE6E-FE6F	2	RHL	Register HL
FE70-FE71	2	RIX	Register IX
FE72-FE73	2	RIY	Register IY
FE74-FE75	2	RAF'	Register AF'
FE76-FE77	2	RBC'	Register BC'
FE78-FE79	2	RDE'	Register DE'
FE7A-FE7B	2	RHL'	Register HL'
FE7C	1	-	Available for user
FE7D	1	JPCODE	JMP code (C3) for word CALL
FE7E-FE7F	2	JPVECT	JMP vector for word CALL
FE80-FE9F	32	-	Available for user