Appendix A DRIVE NAMES

Drive	Peripheral
A:	RAM disk
B:	ROM capsule-1
C:	ROM capsule-2
D:,E:,F:,G:	Floppy disk drives
H:	Microcassette drive
I:	RAM cartridge
J:	ROM cartridge-1
K:	ROM cartridge-2
SCRN:	LCD screen
LPT0:	Printer
COM0:	RS-232C interface
COM1:	Serial interface (SIO)
COM2:	RS-232C input, or SIO output
COM3:	Cartridge serial
KYBD:	Keyboard
CAS0:	External cassette

Appendix B BASIC COMMANDS, STATEMENTS, AND FUNCTIONS AVAILABLE FOR I/O DEVICES

BASIC commands, statements, and functions which can be used with various I/O devices are given in the table below. Disk-1 and Disk-2 denote one of the following disk drives:
Disk-1: RAM disk, RAM cartridge, or floppy disk drives
Disk-2: ROM capsules, or ROM cartridges

Drive name / Command, Statement, or Function	KYBD:	SCRN:	LPT0:	COMn	Disk-1	Disk-2	H:	CAS0:
CLOSE	○	○	○	○	○	○	○	○
DSKF	×	×	×	×	○	○	Note 2	×
EOF	–	×	×	○	○	○	○	○
GET	×	×	×	×	○	○	Note 3	×
INPUT#	○	×	×	○	○	○	○	○
INPUT$	○	×	×	○	○	○	○	○
LINE INPUT#	○	×	×	○	○	○	○	○
LIST	×	○	○	○	○	×	○	○
LOAD	○	×	×	○	○	○	○	○
LOC	–	×	×	○	○	○	○	×
LOF	–	×	×	○	○	○	○	×
OPEN "I"	○	×	×	○	○	○	Note 3	○
OPEN "O"	×	○	○	○	○	×	Note 3	○
OPEN "R"	×	×	×	×	○	○	Note 3	×
POS	×	○	○	○	○	○	○	○
PRINT#	×	○	○	○	○	×	○	○
PRINT USING#	×	○	○	○	○	×	○	○
PUT	×	×	×	×	○	×	Note 3	×
SAVE	×	Note 1	Note 1	Note 1	○	×	○	○
WIDTH	×	×	○	○	×	×	×	○
WRITE	×	○	○	○	○	×	○	○

○ Available
× Not available
– No meaning

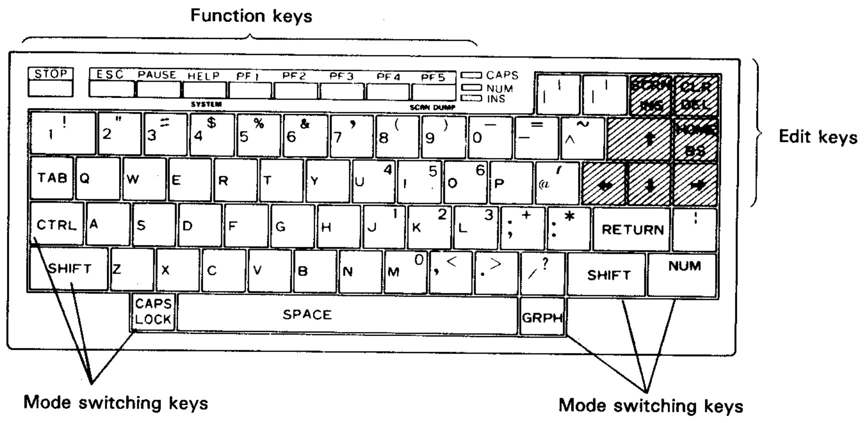
Note 1: Data is output in an ASCII format even if the A option is omitted.
Note 2: Exact value is not returned due to the limitations imposed by cassette tape.
Note 3: There are some restrictions on the use of the file. For example, record numbers must be sequentially assigned, beginning with 1 for a random data file.

Appendix C FORMATTING CHARACTERS

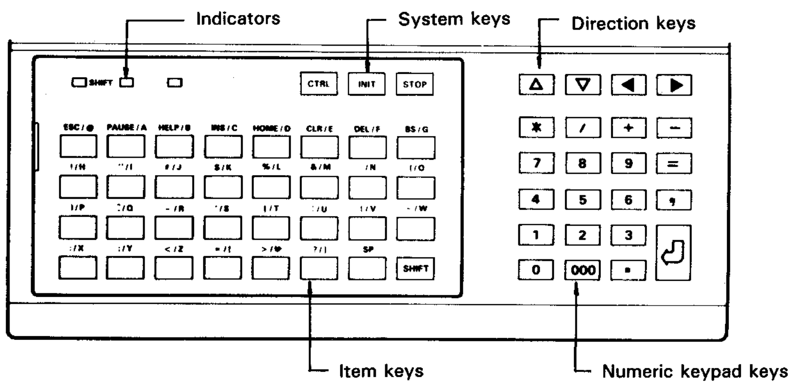
Formatting character	Remarks
!	Causes the first character of a string to be displayed.
&n__spaces&	Causes (n+2) characters to be displayed from a string, starting at the first character.
@	Causes the contents of a string to be displayed as is.
#	Causes the number of digits equal to the number of #'s to be displayed. The result is right-justified.
.	Causes a decimal point to be displayed in the same position when it appears.
+	Causes the sign to be displayed before or after a number.
−	Causes a minus sign to be displayed after a negative number.
* *	Causes leading spaces to the left of a number in the print area to be filled with *'s.
\\\\	Causes \\ (only one) to be displayed to the left of a number in the print area.
* *\\ or \* *	Causes \\ to be displayed just before a number and * to the left of the \\, to fill the leading spaces in the print area.
,	Causes , to be displayed every three digits in the integer part of a number.
∧∧∧∧	Causes a number to be displayed using an exponent.
__ (under line)	Causes the string following the __ to be displayed as is.

Appendix D KEYBOARDS

ASCII keyboard

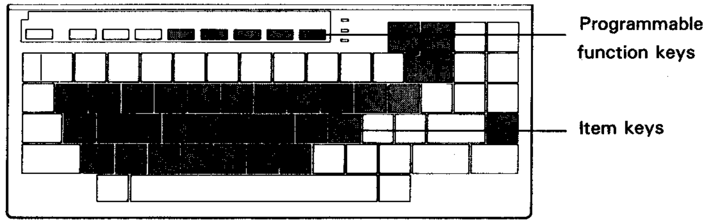


Item keyboard

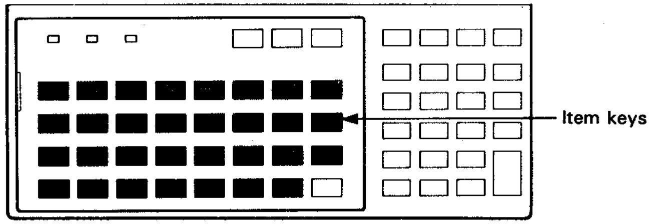


Appendix E PROGRAMMABLE FUNCTION KEYS AND ITEM KEYS

You can assign any string to the programmable function keys and item keys. The keys that can be assigned strings are identified by shaded boxes in the figures below.



ASCII keyboard



Item keyboard

① **Programmable function keys (ASCII keyboard only)**

Load a string for the programmable function keys in the following format:

KEY PF_key_number,string

Specify a number from 1 to 10 for the PF_key_number. 1 to 5 correspond to $\boxed{\text{PF1}}$ to $\boxed{\text{PF5}}$, and 6 to 10 correspond to $\boxed{\text{SHIFT}}$ + $\boxed{\text{PF1}}$ to $\boxed{\text{SHIFT}}$ + $\boxed{\text{PF5}}$, respectively. The number of characters in the string must not exceed 15. Extra characters are ignored. Use the CHR$(n) to define nonprintable characters such as control codes. See Appendix J "CHARACTER CODES" for the control codes.

Examples:

```
KEY 1,"AUTO"
```

Pressing the $\boxed{\text{PF1}}$ key provides the same effect as typing $\boxed{\text{A}}$ $\boxed{\text{U}}$ $\boxed{\text{T}}$ $\boxed{\text{O}}$.

```
KEY 5,"RUN"+CHR$(13)
```

Pressing the $\boxed{\text{PF5}}$ key provides the same effect as typing $\boxed{\text{R}}$ $\boxed{\text{U}}$ $\boxed{\text{N}}$ and pressing the $\boxed{\text{RETURN}}$ key.

② **Item keys**

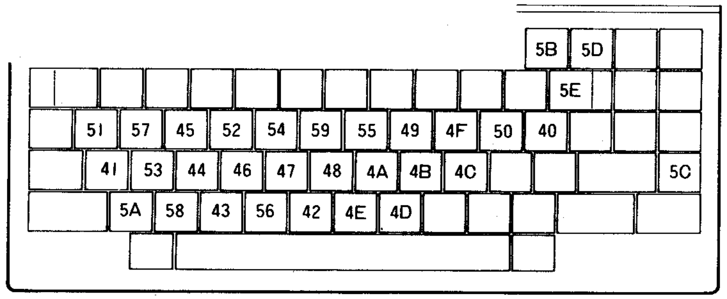
Load a string for the item keys in the following format:

KEY item_key_number,string

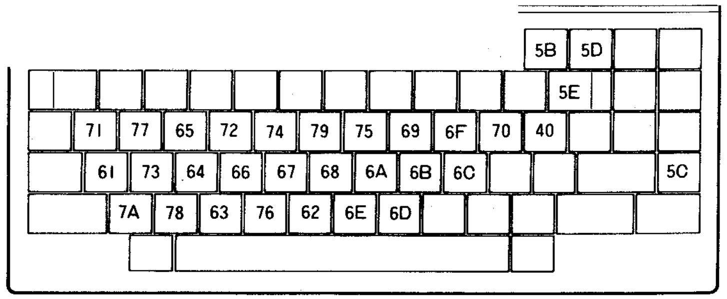
The item_key_number must be within the range of &H40-&H5E, or &H60-&H7E. Keys corresponding to the ASCII and item keyboards and the item key numbers are given below.

ASCII keyboard (numbers in hexadecimal)

CAPS lock mode (the CAPS LED on the keyboard is on)


5B 5D
5E
51 57 45 52 54 59 55 49 4F 50 40
41 53 44 46 47 48 4A 4B 4C 5C
5A 58 43 56 42 4E 4D

Normal mode (three LEDs on the keyboard are off)


5B 5D
5E
71 77 65 72 74 79 75 69 6F 70 40
61 73 64 66 67 68 6A 6B 6C 5C
7A 78 63 76 62 6E 6D

Item keyboard (numbers in hexadecimal)

Normal mode (the SHIFT LED is off)

40	41	42	43	44	45	46	47
48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57
58	59	5A	5B	5C	5D	5E	SHIFT

Shift mode (the SHIFT LED is on)

60	61	62	63	64	65	66	67
68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77
78	79	7A	7B	7C	7D	7E	SHIFT

The number of characters in <string> must not exceed 15. BASIC ignores the 16th and subsequent characters. Use the CHR$(n) function to define non-printable characters such as the control codes.

The item keys on the keyboard are enabled immediately once they are loaded with a string.

On the ASCII keyboard however, you must first switch the key mode to use the function keys. The key mode is set by:

KEY 253: Accepts any defined strings (item key mode)
KEY 254: Ignores any defined strings and accepts normal characters (default mode established when BASIC is started).

You can enter a programmed string by pressing the item key while holding CTRL and SHIFT keys, even if you are in another keyboard mode.
Normal characters will be entered even after the execution of KEY 253, if no item key has been defined.

Strings assigned to item keys on both the item and ASCII keyboards can be cancelled by executing KEY 255.

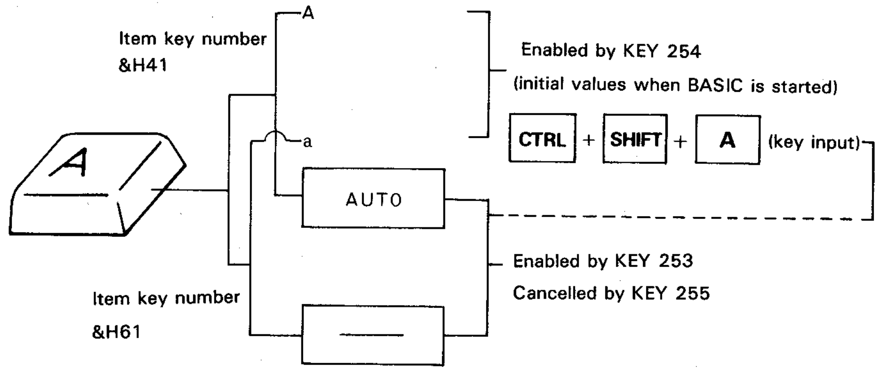
NOTE:
Executing this command on the item keyboard will inhibit entry from the keyboard.

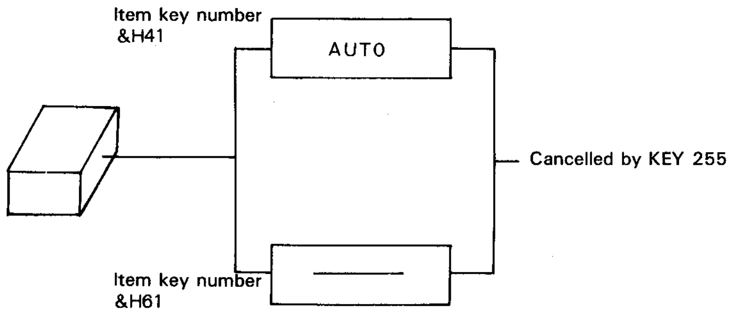
Example:

```
KEY &H41,"AUTO"
```

BASIC will display "AUTO" if you press the A key after executing KEY 253.

If you further press the A key after executing KEY 254, then BASIC will display "A". Pressing the CTRL, SHIFT, and A keys together will also display "AUTO".

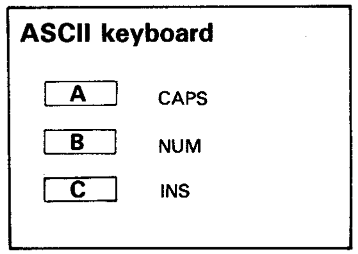


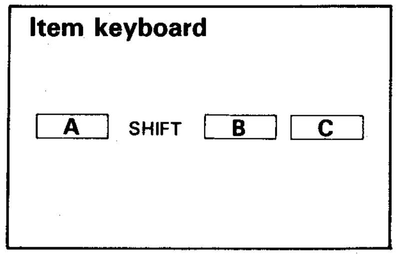


The three LEDs on the ASCII or item keyboard can be turned on and off with a program by using the PRINT statement in the following format:

```
PRINT CHR$(&H1B);CHR$(n)
```

Specify n as a number between &HA0 and &HA5 (160 to 165). The LEDs will turn on and off as follows:





A: Is turned on when the value of n is &HA2 and off when it is &HA3.
B: Is turned on when the value of n is &HA4 and off when it is &HA5.
C: Is turned on when the value of n is &HA0 and off when it is &HA1.

NOTE:
This statement turns the LEDs on and off but does not change the key entry modes.

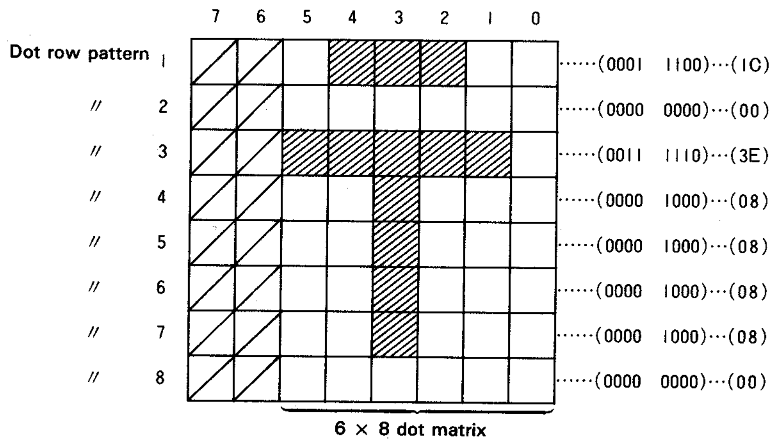
Appendix F USER-DEFINED CHARACTERS

(1) User-defined Characters

The PX-4 allows you to define your own character designs. You can assign up to 30 user-defined characters to any of the unused character codes between &HE2 and &HFF (the other character codes are already assigned letters, symbols, or control characters: see Appendix J). You can display these characters on the screen by using the CHR$ function.

Generating a character pattern

One character can consist of a matrix 6 dots wide by 8 dots long (one of the rectangular segments in the drawing below is referred to as a dot). To define a character pattern, you must compose it with eight bytes of dot data, with each dot row represented by the ASCII code equivalent to the desired binary format, as shown below.



The dot pattern shown above corresponds to the numbers on the right.

0000··· 0	1000··· 8
0001··· 1	1001··· 9
0010··· 2	1010··· A
0011··· 3	1011··· B
0100··· 4	1100··· C
0101··· 5	1101··· D
0110··· 6	1110··· E
0111··· 7	1111··· F

NOTE:
Bits 6 and 7 of each byte are ignored.

Assigning a character pattern to an ASCII character code.

Once you have designed a character pattern, it can be registered in the system as an ASCII character code by using the following statement.

```
PRINT CHR$(&H1B)+CHR$(&HE0)+CHR$(character code)
                           +CHR$(pattern for dot row 1)
                           +CHR$(pattern for dot row 2)
                           +CHR$(pattern for dot row 3)
                           :        :
                           :        :
                           +CHR$(pattern for dot row 8)
```

The ASCII character codes &HE2 to &HFF are available as user-defined characters.

```
5 ' USER-DFEINED CHARACTERS
10 PRINT CHR$(&H1B)+CHR$(&HE0)+CHR$(&HF0
)+CHR$(&H1C)+CHR$(&H0)+CHR$(&H3E)+CHR$(&
H8)+CHR$(&H8)+CHR$(&H8)+CHR$(&H8)+CHR$(&
H0)
20 PRINT CHR$(&HF0)

RUN

〒
Ok
```

A sample program for defining a user-defined character is given below.

You can enter user-defined characters from the keyboard if you load them directly into programmable function keys or item keys.

Example:

Assuming that "〒" is assigned to character code &HF0. You can define "〒" in the system by typing

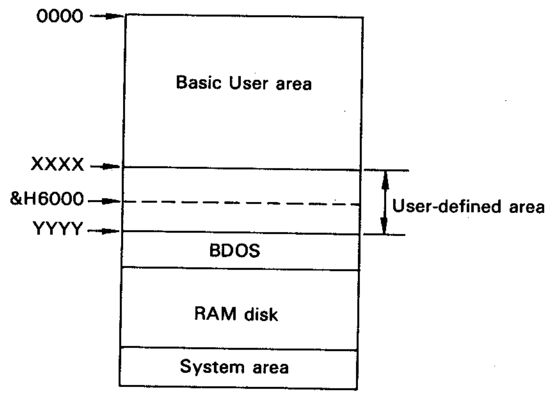
```
KEY 1,CHR$(&HF0)
```

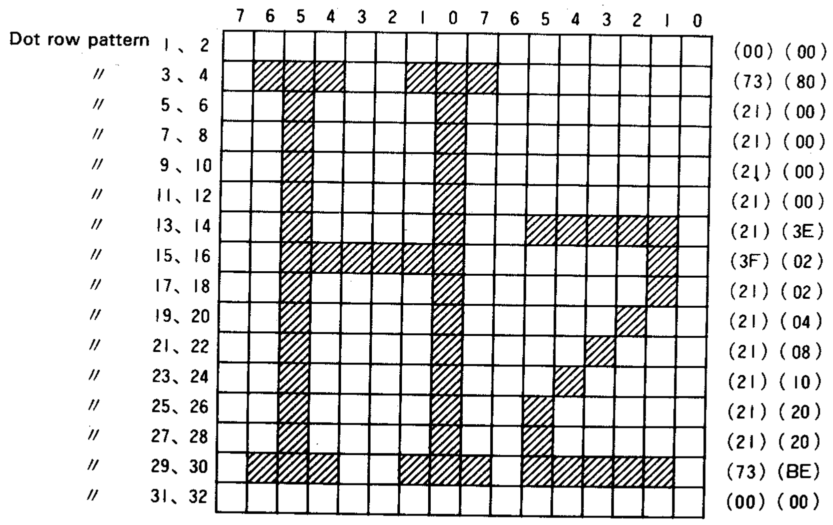
and pressing the PF1

(2) User-defined Pattern

Generating a user-defined pattern

A user-defined pattern consists of a 16 dots by 16 dots matrix and is specified with a total of 32 bytes. The pattern "Hz", for example, can be defined as shown below.



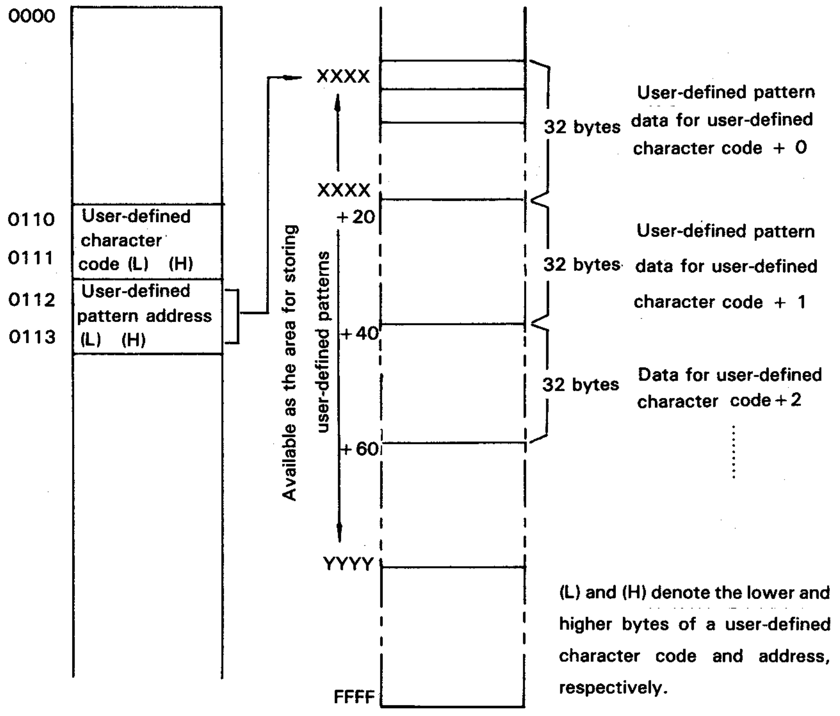


This dot pattern can be stored in memory in the form of binary bytes 00, 00, 73, 80, 21, 00, 21, 00, and so on, in that order.

User-defined pattern area configuration

BASIC stores the address of the area used for storing user-defined patterns in addresses &H0112 and &H0113 and the code for the first user-defined pattern in addresses &H0110 and &H0111. The user can specify the storage addresses and character codes of user-defined patterns by using the POKE statement.

User-defined patterns are defined by consecutive user-defined character codes in the area starting at address XXXX designated by the user-defined pattern address stored in memory addresses &H0112 and &H0113.


0000
0110
0111
0112
0113
User-defined character code (L) (H)
User-defined pattern address (L) (H)
Available as the area for storing user-defined patterns
XXXX
XXXX +20
+40
+60
YYYY
FFFF
32 bytes
User-defined pattern data for user-defined character code + 0
32 bytes
User-defined pattern data for user-defined character code + 1
32 bytes
Data for user-defined character code + 2
(L) and (H) denote the lower and higher bytes of a user-defined character code and address, respectively.
All memory addresses in hexadecimal.

User-defined character codes are assigned to codes from 0 to 65535. When storing code 2000 for example, you must load &HD0 in memory address &H0110 and &H07 in memory address &H0111 because the hexadecimal representation of 2000 is &H07D0. (The decimal to hexadeciaml conversion can be accomplished easily by using the BASIC PRINT HEX&(2000) statement.)

When using an area at &H5000 for storing user-defined patterns, load &H00 in memory address &H0112 and &H50 in memory address &H0113. This allows user-defined patterns to be defined in the memory area starting at &H5000 as 32-byte consecutive character codes (2000, 2001, 2002, and so on).

Defining user-defined patterns into the system

To define user-designed patterns in the system, load the pattern bytes at the address for the first byte of the corresponding character code. In the example below, the pattern corresponding to user-defined character code 2000 is loaded into the 32-byte area from &H4000 through &H401F. Next, the pattern corresponding to 2001 is loaded into the 32-byte area from &H4020 through &H403F.

Example:

```
5 'USER-DEFINED CHARACTERS
100 CLS
110 CLEAR ,&H3FFF
120 FOR I=0 TO 31
130   READ A
140   POKE &H4000+I,A
150 NEXT
160
170 POKE &H110,&HD0 :'2000 = &h07D0
180 POKE &H111,&H7
190 POKE &H112,0
200 POKE &H113,&H40
210 FONT(16,16),PSET,2000
220 END
230
240 DATA 0,0,&h73,&h80,&h21,0,&h21,0,&h2
1,0,&h21,0,&h21,&h3E,&h3F,2,&h21,2,&h21,
4,&h21,8,&h21,&h10,&h21,&h20,&h21,&h20,&
h73,&hBE,0,0
```

```
run

  Hz
```

Explanation:
Line 110 reserves the area for storing the user-defined pattern. The FOR NEXT loop from lines 120 through 150 reads user-defined patterns defined on line 240 into the area starting at address &H4000. Lines 170 and 180 load the user-defined character code and lines 190 and 200 load the start address of the pattern area. The program finally prints the user-defined pattern on the screen at line 210.

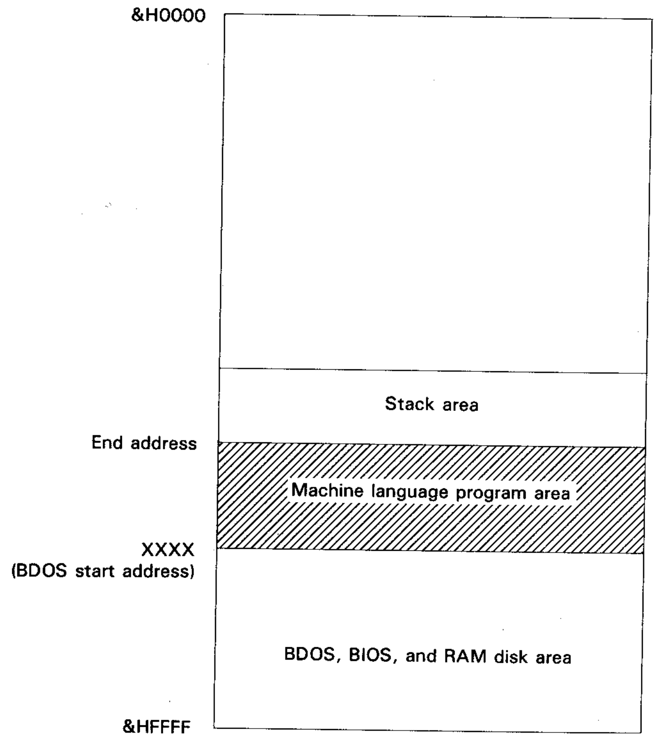
110	User-defined character code	L	User-defined character code	&H07D0 = 2000 (L H2)
111	User-defined character code	H		
112	User-defined pattern address	L	User-defined pattern address	&H4000 (L H)
113	User-defined pattern address	H		

You can store the user-defined patterns onto an auxiliary storage device with the BSAVE command, then load them back into memory with the BLOAD command.

Appendix G MACHINE LANGUAGE PROGRAMS

①Memory allocation

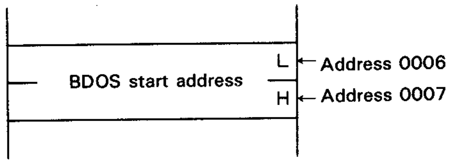
When using machine-language programs or defining user-defined characters, it is necessary to reserve a memory area that does not overlap the one used by BASIC. This is to prevent the BASIC programs or data from being destroyed by the machine language programs, and vice versa. To reserve the memory area, you must specify the end address of the area used by BASIC by using the CLEAR command, or in the /M: parameter when starting BASIC. The machine language program area start address must be lower than the BDOS start address or &H6000, whichever is smaller.



NOTE: All memory addresses are in hexadecimal.

Machine language programs can use the memory area from the address you specified up to XXXX-1. The address XXXX is determined by the sum of the areas available for the RAM disk and user BIOS. The address XXXX is stored in addresses 0006 and 0007.

Example:



L: Lower byte
H: Higher byte

```
PRINT HEX$(PEEK(7)*256+PEEK(6))
6206
Ok
```

The BDOS start address is &H6206 in this example.

Use the POKE command to load short machine language programs, one byte at a time. To load a machine language program file, use the BLOAD command.

- To invoke a machine language program from a BASIC program, use the USR function or CALL statement.

② USR function

To invoke a machine language program with the USR function, specify the start address of the machine language program. Then assign a subroutine number to the machine language program with the DEF USR statement. The subroutine number must be an integer between 0 and 9.

DEF USR<subroutine number> = <start address of the memory area that is to be loaded with the machine language program>

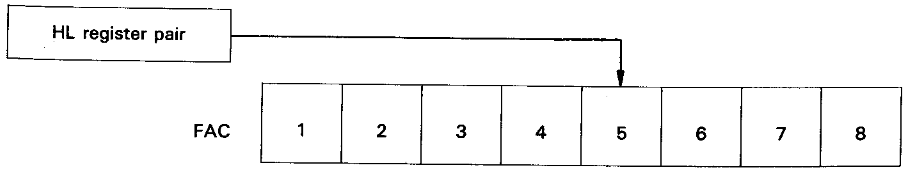
You can now call a machine language program by specifying the subroutine number in the USR function. The format is as follows:

<variable name> = USR[<subroutine number>](argument)

<variable name> is used to store any results from the machine language program execution that are to be returned to the BASIC program. When there are no results to be returned, specify the same variable name for the argument.
Specify for <subroutine number> the same subroutine number that is defined in the DEF USR statement. If the number is omitted, 0 is assumed.
Specify in (argument) the value (of a numeric or string variable) to be passed from the main program to the machine language program. A variable name must be specified even when there are no values to be passed.

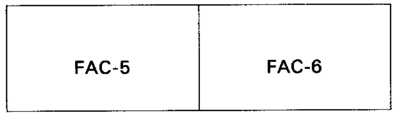
When the USR function is used, register A is loaded with one of the following values which identifies the type of the argument to be passed to the machine language program. If the argument is a number, the HL register pair is loaded with the address of the fifth byte of the FAC (Floating Point Accumulator) memory area, which is used to hold the argument.

Type of argument	Value in register
A Integer	2
String	3
Single-precision real number	4
Double-precision real number	8



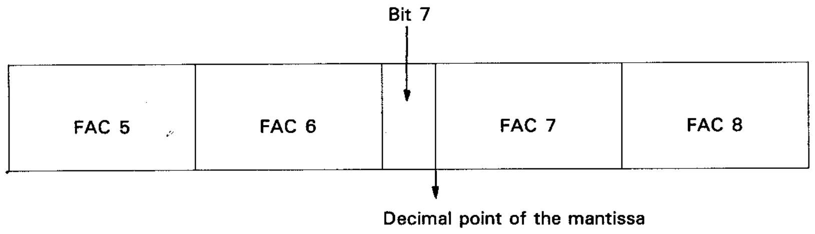
The argument is loaded into the FAC in one of the following ways:

- When the argument is an integer



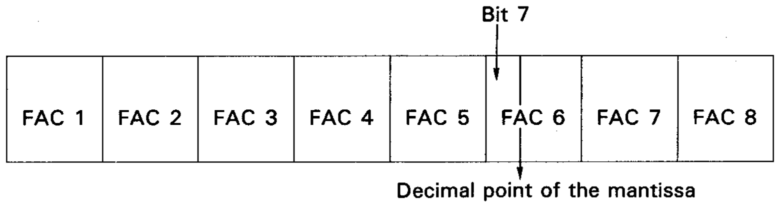
Bytes 5 and 6 of the FAC are loaded with the lower 8 bits and higher 8 bits of the argument, respectively.

- When the argument is a single-precision real number



The highest 7 bits, intermediate 8 bits, and lowest 8 bits of the argument's mantissa are stored into bytes 7, 6, and 5 of the FAC respectively. Bit 7 of byte 7 contains the sign of the argument (0 for positive and 1 for negative). The decimal point of the mantissa is assumed to be placed between bits 6 and 7 of byte 7. Byte 8 is used to hold the value (exponent minus 128).

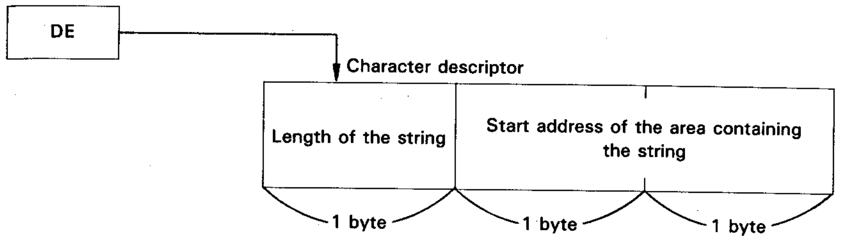
- When the argument is a double-precision real number



Bytes 5 to 8 of the FAC are used in the same way as for a single-precision number. Bytes 1 to 4 contain the lowest 4 bytes of the mantissa.

- When the argument is a string

The start address of the 3-byte area called the "string descriptor" is loaded into the DE register pair. Byte 1 contains the length of the string (0-255); and bytes 2 and 3 contain the start address of the area in which the string is stored.

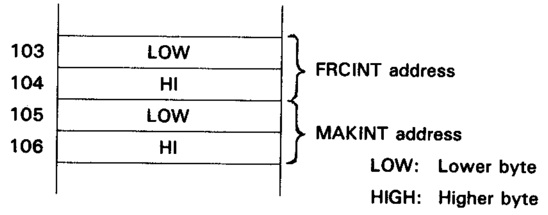


If the argument is a literal string in the program, bytes 2 and 3 of the string descriptor point to the program text.

To make the USR function return a value to BASIC, place the function value in the FAC. Normally, a USR function returns the same type of value as the argument originally passed to the function.

This is easily accomplished by the FRCINT and MAKINT routines provided by PX-4 BASIC. FRCINT converts the value in the FAC to an integer. The converted value is placed into the HL register pair. MAKINT converts the value in the HL register pair into an integer and places it in the FAC. These routines are used to convert to integers values passed from a USR function to a user-defined machine language program, or to return a value to the USR function.

The start addresses of FRCINT and MAKINT are shown in the figure below.



③Invoking a machine language program with the CALL statement

You may also invoke a machine language program from a BASIC program with the CALL statement.

A CALL statement with no argument simply invokes a machine language program and executes it. Control must be returned from the machine language program to the main program via a RET instruction. A CALL statement with arguments passes the addresses of areas that contain them. The method of passing arguments depends on the number of arguments to be passed as follows:

If there are 3 or less arguments, they are placed in registers. The address of the area containing the first argument is placed in the HL register pair, the addresses of the areas containing the second and third arguments, if any, are placed in the DE and BC register pairs, respectively.
If there are more than 3, they are passed as follows:
(a) The address of the area containing the first argument is placed in the HL register pair.
(b) The address of the area containing the second argument is placed in the DE register pair.
(c) For arguments 3, 4,...and so on, the start address of the table containing the start addresses of the arguments are placed in register pair BC. (i.e., BC points to the start address of the area containing the third argument).

④BSAVE, BLOAD

The BSAVE command saves a machine language program or data in memory into a file. The BLOAD command loads a machine language program or data file into memory.

The BSAVE command writes a 5-byte header at the beginning of the file in addition to the data in memory. The BLOAD command cannot load a file which has an illegal header. The 5-byte header has the following format:

```
1      &HFD
2-3    Address
4-5    Length (in bytes)
.      ⎫
.      ⎬ data
.      ⎭
```

The address and length are specified in the BSAVE command. If no address is specified in the BLOAD command, the file will be loaded into the memory space starting at the address originally specified in the BSAVE command.

Appendix H CONTROL CODES

Code		Function	Key usage
Decimal	Hexadecimal		
1	01	Move the cursor to the beginning of the current line.	CTRL + A
2	02	Move the cursor one word to the left.	SHIFT + B
3	03	Stop the program execution. These keys also stop the automatic number generation that is initiated by the AUTO command.	STOP , CTRL + C
5	05	Delete characters to the end of the line.	CTRL + E
6	06	Move the cursor one word to the right.	CTRL + F
8	08	Delete the character to the left.	BS , CTRL + H
9	09	Move the cursor to the next tab position.	TAB, CTRL + I
11	0B	Move the cursor to the home position (upper left corner) of the virtual screen.	HOME , CTRL + K
12	0C	Clear the virtual screen.	CLR , CTRL + L
13	0D	Execute a command or statement.	RETURN CTRL + M
18	12	Toggle between the overwrite and insert modes of the BASIC screen editor.	INS CTRL + S
19	13	Suspend the execution of a BASIC program.	PAUSE , CTRL + S
24	18	Move the cursor to the end of the line.	CTRL + X
26	1A	Delete all characters to the end of the screen.	CTRL + Z

Code		Function	Key usage
Decimal	Hexadecimal		
27	1B	Escape code.	ESC
28	1C	Moves the cursor to the right.	→
29	1D	Moves the cursor to the left.	←
30	1E	Moves the cursor upward.	↑
31	1F	Moves the cursor downward.	↓

Appendix I RESERVED WORDS

ABS
ALARM
ALARM$
AND
ASC
ATN
AUTO

BEEP
BLOAD
BSAVE

CALL
CDBL
CHAIN
CHR$
CINT
CLEAR
CLOSE
CLS
COM
COMMON
CONT
COPY
COS
CSNG
CSRLIN
CVI
CVS
CVD

DATA
DATES
DAY
DEF
DEFINT
DEFSNG
DEFDBL
DEFSTR
DELETE
DIM
DSKF

EDIT
ELSE
END
EOF
ERASE
EQV
ERL
ERR
ERROR
EXP
EXTD

FIELD
FILES
FIX
FN
FONT
FOR
FRE

GET
GO
GOSUB

HEX$

IF
IMP
INKEY$
INP
INPUT
INPUT #
INSTR
INT

KEY
KILL

LEFT$
LEN
LET
LINE
LIST
LLIST
LOAD
LOAD?
LOC
LOCATE
LOF
LOG
LOGIN
LPOS
LPRINT
LSET

MENU
MERGE
MID$
MKD$
MKI$
MKS$
MOD
MOTER
MOUNT

NAME
NEW
NEXT
NOT

OCT$
OFF
ON
OPEN
OPTION
OPTION COUNTRY
OPTION CURRENCY
OR
OUT

PCOPY
PEEK
POINT
POKE
POS
POWER
PRESET
PRINT
PRINT #
PSET
PUT

RANDOMIZE
READ
REM
REMOVE
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN

SAVE
SCREEN
SGN
SIN
SOUND
SPACE$
SQR
STAT
STEP
STOP
STR$
STRING$
SWAP
SYSTEM

TAN
TAPCNT
THEN
TIME$
TITLE
TO
TROFF
TRON

USING
USR

VAL
VARPTR

WAIT
WEND
WHILE
WIDTH
WIND
WRITE

XOR

Appendix J CHARACTER CODES

	Hex. No.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Hex. No.	Binary No.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	0	16	SP 32	0 48	@ 64	P 80	` 96	p 112	┼ 128	○ 144	160	176	192	208	⌂ 224	240
1	0001	1	17	! 33	1 49	A 65	Q 81	a 97	q 113	┴ 129	♠ 145	161	177	193	209	↵ 225	241
2	0010	2	18	" 34	2 50	B 66	R 82	b 98	r 114	┬ 130	♥ 146	162	178	194	210	226	242
3	0011	3	19	# 35	3 51	C 67	S 83	c 99	s 115	┤ 131	♦ 147	163	179	195	211	227	243
4	0100	4	20	$ 36	4 52	D 68	T 84	d 100	t 116	├ 132	♣ 148	164	180	196	212	228	244
5	0101	5	21	% 37	5 53	E 69	U 85	e 101	u 117	─ 133	♪ 149	165	181	197	213	229	245
6	0110	6	22	& 38	6 54	F 70	V 86	f 102	v 118	│ 134	[illegible] 150	166	182	198	214	230	246
7	0111	7	23	' 39	7 55	G 71	W 87	g 103	w 119	┌ 135	[illegible] 151	167	183	199	215	231	247
8	1000	8	24	(40	8 56	H 72	X 88	h 104	x 120	┐ 136	[illegible] 152	168	184	200	216	232	248
9	1001	9	25	) 41	9 57	I 73	Y 89	i 105	y 121	└ 137	[illegible] 153	169	185	201	217	232	249
A	1010	10	26	* 42	: 58	J 74	Z 90	j 106	z 122	┘ 138	[illegible] 154	170	186	202	218	234	250
B	1011	11	27	+ 43	; 59	K 75	[91	k 107	{ 123	▒ 139	↑ 155	171	187	203	219	235	251
C	1100	12	28	, 44	< 60	L 76	\ 92	l 108	¦ 124	■ 140	↓ 156	172	188	204	220	236	252
D	1101	13	29	- 45	= 61	M 77	] 93	m 109	} 125	▬ 141	× 157	173	189	205	221	237	253
E	1110	14	30	. 46	> 62	N 78	^ 94	n 110	~ 126	▌ 142	÷ 158	174	190	206	222	238	254
F	1111	15	31	/ 47	? 63	O 79	_ 95	o 111	⌂ 127	● 143	± 159	175	191	207	223	239	255

ASCII

NOTES:
1. *(0)D through (31)D are control characters.*
2. *(32)D through (127)D are ASCII characters.*
3. *Characters displayed for codes E0 to FF can be defined by the user. For further details, see Appendix H and the Operating Manual.*

Differences between the USASCII character set and the character sets of other countries are as shown below.

Country / Dec. Code	United States	France	Germany	England	Denmark	Sweden	Italy	Spain	Norway
35	#	#	#	£	#	#	#	₧	#
36	$	$	$	$	$	¤	$	$	¤
64	@	à	§	@	É	É	@	@	É
91	[	°	Ä	[	Æ	Ä	°	¡	Æ
92	\	ç	Ö	\	Ø	Ö	\	Ñ	Ø
93	]	§	Ü	]	Å	Å	é	¿	Å
94	^	^	^	^	Ü	Ü	^	^	Ü
96	\`	\`	\`	\`	é	é	ù	\`	é
123	{	é	ä	{	æ	ä	à	¨	æ
124	|	ù	ö	|	ø	ö	ò	ñ	ø
125	}	è	ü	}	å	å	è	}	å
126	~	¨	ß	~	ü	ü	ì	~	ü

Appendix K ***ERROR CODES AND MESSAGES***

BASIC displays an error message and enters the command mode when one of the errors listed in this appendix occurs.

Message	Code	Desciption
/0	11	**Division by zero** An attempt was made to divide by zero. <Possible causes> 1. Zero was used as a divisor. 2. Division was attempted using an undefined variable as the divisor.
AC	72	**Tape access error** An attempt was made to access an access-inhibited file. <Possible causes> 1. An attempt was made to access an access-inhibited microcassette file. 2. An attempt was made to mount a microcassette without executing the REMOVE command to demount the previous one. 3. The REMOVE command was executed while the microcassette was in the demounted condition. 4. The WIND command was executed while the microcassette was in the mounted condition. 5. The TAPCNT function was evaluated when the microcassette was in the mounted condition.
AO	55	**File already open** An OPEN "O" statement was executed for a file which was already open, or a KILL command was executed for a file that was open.

Message	Code	Description
BF	**54**	**Bad file mode** An illegal statement or function was specified for a file. < Possible causes > 1. A PUT or GET statement or the LOF function was executed for a sequential file. 2. A LOAD command was executed for a random file. 3. A file mode other than "I", "R", and "O" was specified in an OPEN statement. 4. An attempt was made to MERGE a file that was not saved in ASCII format.
BN	**52**	**Bad file number** A statement or command references a file that has not been opened. Or, the file number specified in the OPEN statement was outside of the range of file numbers that was specified when BASIC was first started.
BS	**9**	**Bad subscript** The subscript for an array variable was outside the range permitted for that array. 1. The subscript specified was greater than the maximum specified in the DIM statement defining that array. 2. A subscript greater than 10 was used when no array variable was specified by a DIM statement. 3. Zero was used after executing OPTION BASE 1.
CN	**17**	**Can't continue** BASIC cannot continue toexecute the program < Possible causes > 1. Program execution was terminated due to an error. 2. The program was modified while execution was suspended. 3. The STOP key was pressed during execution of an received faster than it can be processed by the computer.
CO	**28**	**Communication buffer overflow** The receive buffer overflowed during receipt of data via the RS-232C interface. This error occurs when data is received faster than it can be processed by the computer.

Message	Code	Description
DD	**10**	**Duplicate definition** An array was defined more than once. < Possible causes > 1. A second DIM statement was executed for an array without freeing the memory area allocated to that array by the ERASE statement. 2. An undefined array was used, then an attempt was made to redimension that array with a DIM statement. 3. The OPTION BASE statement was used more than once. 4. An OPTION BASE statement was executed after an array had been dimensioned by a DIM statement or after the array variable had been used.
DF	**61**	**Disk full** All the disk storage space is in use.
DR	**70**	**Disk read error** An error occurred while a file on a disk was being input.
DS	**66**	**Direct statement in file** A program line with no line number was encountered during the execution of a LOAD or a MERGE command. An attempt was made to execute a LOAD command for a data file or a machine language program.
DT	**24**	**Device time out** A peripheral device was not ready for input or output processing. 1. CTS was not held high and transmission via the RS-232C interface was not enabled within the needed period of time after an OPEN "O" statement was executed. Or, the transmitter was not ready within the needed period of time after a DSR send check was specified with the "c" option at interface open time. 2. The STOP key was pressed while output to the RS-232C interface was being deferred for some reason. 3. DSR or DCD did not go ON within the needed period of time after an OPEN"I" statement was executed. Both the DSR receive check and DCD check were specified with the "c" option at interface open time. 4. The printer was not ready when output to the printer was attempted.

Message	Code	Description
DU	68	**Device unavailable** An attempt was made to access a peripheral device not connected to the computer.
DW	71	**Disk write error** An error occurred while a file was being written to a disk device.
FA	25	**Device fault** DSR or DCD went OFF during input from the RS-232C interface after a DSR check or DCD check was specified with the "c" option in the OPEN"I" statement at interface open time.
FC	5	**Illegal function call** A statement or function was incorrectly specified. < Possible causes > 1. Specification of a negative number as an array variable subscript 2. Specification of a negative number as the argument in the LOG function 3. Specification of a negative number as the argument of the SQR function 4. Specification of a non-integer exponent with a negative mantissa 5. A call to a USR function for which the start address of the machine language program has not been defined 6. An incorrectly specified argument in any of the following statements or functions: MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, ON...GOSUB, ON...GOTO, ASC, SCREEN, VARPTR, LOGIN, STAT 7. Specification of a non-existent line number in a DELETE statement 8. Attempting to erase a non-existent array variable with an ERASE statement 9. Execution of an EDIT command for a program that was saved on an auxiliary storage device with a SAVE command with the "p" option then loaded into the program area again 10. Execution of a RENUM command with a number greater than 65529. Or, attempting to change the order of the program lines with a RENUM command 11. Execution of a SWAP statement for a variable to which no value has been given by a LET statement

Message	Code	Description
FD	64	**Bad file descriptor** An illegal file descriptor was specified in a LOAD, SAVE, KILL command, or OPEN statement.
FE	58	**File already exists** The new file name specified in a NAME statement is already being used by another file on the disk.
FN	26	**FOR without NEXT** A FOR statement was encountered without a corresponding NEXT.
FO	50	**Field overflow** The size of the random file variables specified in a FIELD statement exceeds the size of a record specified by an OPEN statement.
ID	12	**Illegal direct** A statement that is illegal in the direct mode was entered in direct mode.
IE	82	**Input past end** An attempt was made to input a file which was empty or a file from which all data had been read. (To avoid this error, use the EOF function.)
IE	62	**Input past end** The STOP key was pressed while the RS-232C interface was waiting for input with an INPUT#, INPUT$, or a similar command.
IO	57	**Device I/O error** An error occurred involving input or output to a peripheral device. < Possible causes > 1. An I/O error occurred during access to a disk device. 2. An error occurred during input from the RS-232C interface. (In this case, the error condition will be reset if input is continued, but there is no assurance that data received will be correct.) 3. The printer power was off or a fault occurred when data was output to the printer.
IT	51	**Internal error** An internal malfunction occurred in BASIC.
LO	23	**Line buffer overflow** A line that did not fit in the line buffer was input.
LS	15	**String too long** The length of a string variable exceeds 255 characters.
MF	67	**Too many files** An attempt was made to create a new disk file after all directory entries were full.
MO	22	**Missing operand** An operand required for an expression is missing.

Message	Code	Description
NE	53	**File not exist** The file name specified in a LOAD, KILL, NAME, or OPEN statement does not exist on the disk.
NF	1	**NEXT without FOR** A NEXT statement was executed without a corresponding FOR statement. < Possible causes > 1. A FOR statement corresponding to the NEXT was not executed previously. 2. The variable in a NEXT statement does not correspond to any previously executed FOR statement variable. 3. Execution branched to a point within a FOR/NEXT loop from elsewhere in the program.
NR	19	**No RESUME** No RESUME statement was included in an error processing routine. All error processing routine must conclude with an END or RESUME statement.
OD	4	**Out of data** A READ statement was executed when there was no unread data remaining in the program's DATA statement. < Possible causes > 1. The number of data items in a DATA statement was smaller than that of the variables in a READ statement. 2. Incorrect specification of a RESTORE statement. 3. Incorrect delimiting punctuation used in a DATA statement
OM	7	**Out of memory** The memory available is insufficient for executing a program. < Possible causes > 1. The program is too long. 2. The program uses too many variables. 3. The suscript range specified in a DIM statement is too large. 4. An e4xpression has too many levels of parentheses. 5. FOR...NEXT loop or GOSUB...RETURN sequences are nested to too many levels.
OS	14	**Out of string space** Insufficient memory space is available for storing string variables.

Message	Code	Description
OV	14	**Overflow** A numeric value was encountered whose magnitude exceeds the limits prescribed by BASIC. < Possible causes > 1. The result of an integer calculation was outside the range −32768 to 32767. 2. The result of a single or double precision number calculation was outside the range −1.70141E38 to 1.70141E38. 3. The argument specified for the CINT function was outside the range −32768 to 32767. 4. The argument specified for the HEX$ or OCT$ function was outside the range −32768 to 65535.
RG	3	**RETURN without GOSUB** A RETURN statement was encountered which did not correspond to previously executed GOSUB statement. < Possible causes > 1. Execution was transferred to a subroutine by a GOTO statement (GOSUB must be used). 2. The line number specified in a RUN command indicated a line in a subroutine. 3. The main program included no GOTO or END statement at the end and the program flow moved into a subroutine.
RN	63	**Bad record number** The record number specified in a PUT or GET statement was zero.
RW	20	**RESUME without error** A RESUME statement was executed outside an error processing routine. < Possible causes > 1. A RESUME statement was encountered in an error processing routine to which the program was passed by a GOTO or GOSUB statement. 2. A RESUME statement was encountered in an error processing routine that was executed due to the absence of an END statement between the main and error processing routines.

Message	Code	Description
SN	2	**Syntax error** A statement does not conform to the BASIC syntax rules. <Possible causes> 1. Wrong reserved word(s) 2. Unmatched parentheses 3. Wrong delimiting punctuation (commas, periods, colons, or semi-colons) 4. Variable name beginning with a character other than a alphabetic character 5. Reserved words used as the first characters of a variable name 6. Wrong number or type of arguments specified in a function or statement 7. Type of a value included in a DATA statement did not match the corresponding variable in the list of variables specified in a READ statement.
ST	16	**String formula too complex** A string expression has too many levels of parentheses.
TM	13	**Type mismatch** A string was assigned to a numeric variable or a numeric value to a string variable. <Possible causes> 1. An attempt was made to assign a numeric value to a string variable. 2. An attempt was made to assign a string to a numeric variable. 3. The wrong type of value was specified as the argument of a function.
UF	18	**Undefined user function** A call was made to an undefined user function. <Possible causes> 1. The letters FN were used at the beginning of a variable name. 2. The function name was specified incorrectly in the DEFFN statement or when the function was called. 3. The user function was called before the corresponding DEFFN statement was executed.

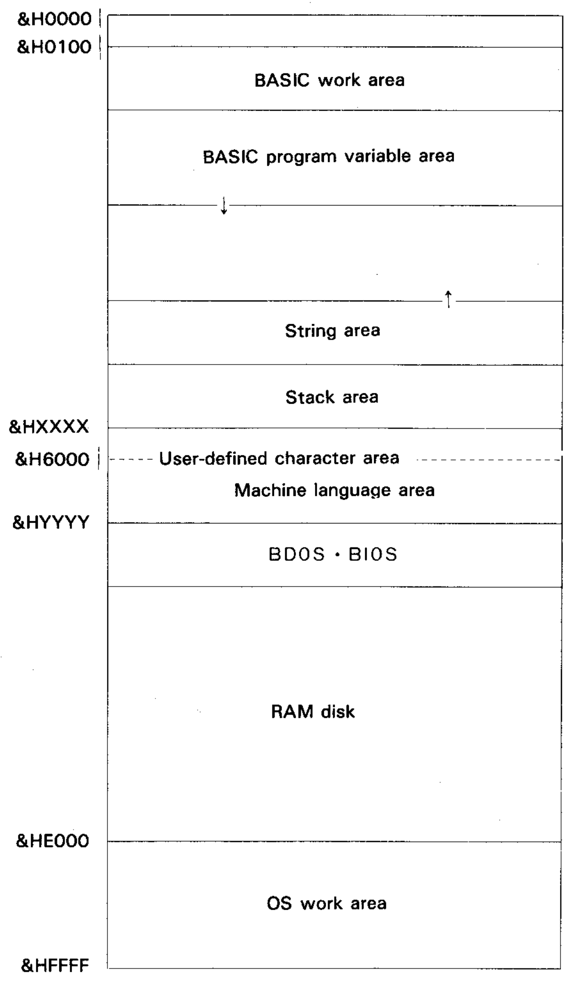
Message	Code	Description
UL	8	**Undefined line number** A non-existent line number was specified in one of the following commands or statements: GOTO, GOSUB, RESTORE, RUN, RENUM.
UP	21	**Unprintable error** No error message has been provided for the error codes, 27, 31-49, 56, 59, 60, 73-255.
WE	29	**WHILE without WEND** A WHILE statement was encountered without a corresponding WEND.
WH	30	**WEND without WHILE** A WEND statement was encountered without a corresponding WHILE.
WP	69	**Disk write protect** An attempt was made to write a file to a disk which was write-protected by a write protect tab.

Appendix L DERIVED FUNCTIONS

Functions that are not intrinsic to PX-4 BASIC may be calculated as follows:

Function	HC-40/41 BASIC Equivalent
SECANT	SEC(X) = 1/COS(X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = COS(X)/SIN(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SQR(− X * X + 1))
INVERSE COSINE	ARCCOS(X) = − ATN(X/X * X + 1)) + 1.570796326794897
INVERSE SECANT	ARCSEC(X) = ATN(SQR(X * X − 1)) + (SGN(X) − 1 * 1.570796326794897
INVERSE COSECANT	ARCCSC(X) = ATN(1/SQR(X * X − 1)) + (SGN(X) − 1) * 1.570796326794897
INVERSE COTANGENT	ARCCOT(X) = − ATN(X) + 1.570596326794897
HYPERBOLIC SINE	SINH(X) = − (EXP(X) − EXP(− X))/2
HYPERBOLIC COSINE	COSH(X) = EXP(X) + EXP(− X))/2
HYPERBOLIC TANGENT	TANH(X) = − EXP(− X)/(EXP(X) + EXP(⊋ − X)) * 2 + 1
HYPERBOLIC SECANT	SECH(X) = 2/(EXP(X) + EXP(− X))
HYPERBOLIC COSECANT	CSCH(X) = 2/(EXP(X) − EXP(− X))
HYPERBOLIC COTANGENT	COTH(X) = EXP(− X)/(EXP(X) − EXP (− X)) * 2 + 1
INVERSE HYPERBOLIC SINE	ARCSONH(X) = LOG(X + SQR(X * X + 1))
INVERSE HYPERBOLIC COSINE	ARCCOSH(X) = LOG(X + SQR(X * X − 1))
INVERSE HYPERBOLIC TANGENT	ARCTANH(X) = LOG(1 + X)/(1 − X))/2
INVERSE HYPERBOLIC SECANT	ARCSECH(X) = LOG((SQR(− X * X + 1) + 1)/X)
INVERSE HYPERBOLIC COSECANT	ARCCSH(X) = LOG((SGN(X) * SQR(X * X + 1) + 1/X)
INVERSE HYPERBOLIC COTANGENT	ARCCOTH(X) = LOG((X + 1)/(X − 1))/2

Appendix M MEMORY MAP

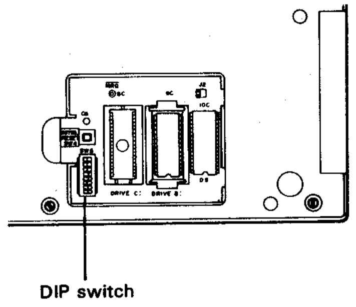


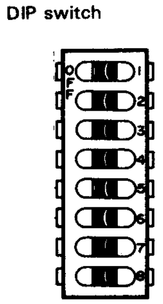
&HXXXX:	Must be specified in the /M option when starting BASIC, or with the CLEAR command. Must be less than &H6000 or &HYYYY, whichever is smaller.
&HYYYY:	This value depends on the sum of the sizes of the RAM disk and user BIOS areas. See APPENDIX G "Machine Language Programs" for further information.

Appendix N DIP SWITCH SETTING

The DIP swith located inside the ROM capsule compartment is used to select some of system configurations.
To change DIP switch setting:

1) Turn off the power switch.
2) Change the desired setting by a ball-point pen, twizers or small screw driver.
3) Push the reset switch on the right side of the computer.
4) Turn on the power switch.





1. Country selection

International character set can be selected by setting of four switches.
It can be also selected by using CONFIG.COM of CP/M utilities. See Operating Mannal for details.

Keyboard type	SW-1	SW-2	SW-3	SW-4
ASCII	ON	ON	ON	ON
France	OFF	ON	ON	ON
German	ON	OFF	ON	ON
England	OFF	OFF	ON	ON
Denmark	ON	ON	OFF	ON
Sweden	OFF	ON	OFF	ON
Italy	ON	OFF	OFF	ON
Spain	OFF	OFF	OFF	ON
Norway	OFF	ON	ON	OFF

2. LST: device selection

When an external printer or optional Cartridge Printer is used, select appropriate setting according to the device to be used.
When the RS-232C or serial interface is used, confirm the parameters such as baud rate match with those of the device connected. Use CONFIG.COM to select interface parameters.

Device name	SW-5	SW-6
Serial interface	OFF	OFF
RS-232C interface	OFF	ON
Cartridge printer	ON	OFF
Printer interface	ON	ON