—— CONTENTS ——

USING LIBRARY ROUTINES

The FDOS master diskette contains three libraries (MONEQU.LIB,FDOSEQU.LIB and RELO.LIB).

MONEQU.LIB is a file of monitor subroutine addresses defined with the EQU statement. That is, it contains a program such as that shown at right which has been assembled and converted to the library (.LIB) mode.

```
GETL   : EQU  0003H
LETNL  : EQU  0006H
PRNTS  : EQU  000CH
PRNT   : EQU  0012H
MSG    : EQU  0015H
MSGX   : EQU  0018H
GETKY  : EQU  001BH
BRKEY  : EQU  001EH
MELDY  : EQU  0030H
              :
              :
```

Part of the contents of MONEQU . ASC

Monitor subroutines are often used when creating programs with the assembler; in such cases, they are used as described below.

First, the subroutine names are written as is (without addresses defined) as external names when the program is written. These are then assembled with the assembler. When the assembly listing is reviewed at this time, the symbol "E" is affixed to indicate that the names are external names. Next, the program is linked; MONEQU .LIB is linked at this time also. For example,

```
2 > LINK GAMEPRG1, $FD1 ; MONEQU .LIB ↵
         ↑ Program created
```

MONEQU.LIB must be written last at this time.

FDOSEQU.LIB is a file of subroutine addresses in FDOS which are defined with the EQU statement; it is used in the same manner as MONEQU.LIB. Since MONEQU.LIB is contained in FDOSEQU.LIB, it is only necessary to link FDOSEQU.LIB when both monitor and FDOS subroutines are to be used at the same time.

RELO.LIB is a library of subroutines for programs created with the BASIC compiler. It contains subroutines for the four basic arithmetic operations, functional calculations, character string processing, error message display and many others. In other words, whereas MONEQU.LIB and FDOSEQU.LIB are simple collections of EQU statements, RELO.LIB contains actual subroutines.

When the linker is used for linkage with RELO.LIB, it is possible to select only the routines required from the many available for linkage.

RELO.LIB is used in the same manner as MONEQU.LIB and FDOSEQU.LIB. Further, the contents of MONEQU.LIB and FDOSEQU.LIB are included in RELO.LIB.

Source programs MONEQU.ASC and FDOSEQU.ASC are also included on the master diskette along with MONEQU.LIB and FDOSEQU.LIB. It is possible to modify and add to the libraries by regenerating the source programs to recreate the libraries as necessary.

Note:

Detailed procedures for using FDOSEQU.LIB are contained in "LINKING ASSEMBLY PROGRAM WITH FDOS" in Appendix; see "EXAMPLE OF PLOTTER CONTROL APPLICATION" in Programming Utility for details on RELO.LIB.

MONITOR SUBROUTINES (MONEQU.LIB)

Subroutine name (hexadecimal address)	Function	Registers preserved
CALL LETNL ($0006)	To change the line and set the cursor to the beginning of the next line.	All registers except AF
CALL NL ($0009)	Changes the line and sets cursor to its beginning if the cursor is not already located at the beginning of a line.	All registers except AF
CALL PRNTS ($000C)	Displays one space only at the cursor position on the diplay screen.	All registers except AF
CALL PRNT ($0012)	Handles data in A register as ASCII code and displays it on the scresn, starting at the cursor position. However, a carriage return is performed for 0DH and the various cursor control operations are performed for 11H-16H when these are included.	All registers except AF
CALL MSG ($0015)	Displays a message, staring at the cursor position on the screen. The starting address of the message must be specified in the DE register in advance. The message is written in ASCII code and must end in 0DH. A carriage return is not executed, however, cursor control codes (11H-16H) are.	All registers
CALL MSGX ($0018)	Almost the same as MSG, except that cursor control codes are for reverse character display.	All registers
CALL BELL ($003E)	Sounds a tone (approximately 880 Hz) momentarily.	All registers except AF
CALL MELDY ($0030)	Plays music according to music data. The starting address of the music data must be specified in advance in the DE register. As with BASIC, the musical interval and the duration of notes of the musical data are expressed in that order in ASCII code. The end mark must be either 0DH or C8H "■". The melody is over if C flag is 0 when a return is made; if C flag is 1 it indicates that SHIFT + BREAK were pressed.	All registers except AF
CALL XTEMP ($0041)	Sets the musical tempo. The tempo data (01 to 07) is set in and called from A register. 01 : Slowest 04 : Medium speed 07 : Fastest Care must be taken here to ensure that the tempo data is entered in A register in binary code, and not in the ASCII code corresponding to the numbers 1 to 7 (31H to 37H).	All registers
CALL MSTA ($0044)	Continuously sounds a note according to a specified division factor. The division factor *nn'* consists of two bytes of data; *n'* is stored at address 11A1H and *n* is stored at address 11A2H. The relationship between the division factor and the frequency produced is 2 MHz/*nn'*.	BC and DE only
CALL MSTP ($0047)	Discontinues a tone being sounded.	All registers except AF

Subroutine name (hexadecimal address)	Function	Registers preserved
CALL TIMST **($0033)**	Sets the built-in clock. (The clock is activated by this call.) The call conditions are: A register ← 0 (AM), A register ← 1 (PM) DE register ← the time in seconds (2 bytes)	All registers except AF
CALL TIMRD **($003B)**	Reads the value of the built-in clock. The conditions upon return are: A register ← 0 (AM), A register ← 1 (PM) DE register ← the time in seconds (2 bytes)	All registers except AF and DE
CALL BRKEY **($001E)**	Checks whether $\boxed{\text{SHIFT}}$ + $\boxed{\text{BREAK}}$ were pressed. Z flag is set if they were pressed, and Z flag is reset if they were not.	All registers except AF
CALL GETL **($0003)**	Inputs one line entered from the keyboard. The starting address where the data input is to be stored must be specified in advance in the DE register. $\boxed{\text{CR}}$ functions as the end mark. 80 is the maximum number of characters which can be input (including the end mark 0DH). Key input is displayed on the screen and cursor control is also accepted. The BREAK code (1BH) followed by a carriage return code (0DH) is set at the beginning of the address specified in the DE register when $\boxed{\text{SHIFT}}$ + $\boxed{\text{BREAK}}$ are pressed.	All registers
CALL GETKY **($001B)**	Takes one character only into A register from the keyboard in ASCII code. A return is made after 00 is set in A register if no key is pressed when the subroutine is executed. However, key input is not displayed on the screen. Codes which are taken into A register when these special keys are pressed are shown below.	All registers except AF

Special key		Code taken into A register
$\boxed{\text{DEL}}$		60 H
$\boxed{\text{INST}}$		61 H
$\boxed{\text{GRPH}}$	graphic mode	62 H
	normal mode	63 H
$\boxed{\text{BREAK}}$		64 H
$\boxed{\text{CR}}$ or $\boxed{\text{ENT}}$		66 H
$\boxed{\text{CTRL}}$ + $\boxed{\text{A}}$ ~ $\boxed{\text{Z}}$		01 H ~ 1 AH
$\boxed{\text{CTRL}}$ + $\boxed{\text{[}}$		1 BH
$\boxed{\text{CTRL}}$ + $\boxed{\backslash}$		1 CH
$\boxed{\text{CTRL}}$ + $\boxed{\text{]}}$		1 DH
$\boxed{\text{CTRL}}$ + $\boxed{\text{^}}$		1 EH
$\boxed{\text{CTRL}}$ + $\boxed{\text{–}}$		1 FH

Subroutine name (hexadecimal address)	Function	Registers preserved
CALL PRTHL **($03BA)**	Displays the contents of the HL register on the display screen as a 4-digit hexadecimal number.	All registers except AF
CALL PRTHX **($03C3)**	Displays the contents of the A register on the display screen as a 2-digit hexadecimal number.	All registers except AF
CALL ASCI **($03DA)**	Converts the contents of the lower 4 bits of A register from hexadecimal. to ASCII code and returns after setting the converted data in A register.	All registers except AF
CALL HEX **($03F9)**	Converts the 8 bits of A register from ASCII code to hexadecimal and returns after setting the converted data in the lower 4 bits of A register. When C flag = 0 upon return A register ← hexadecimal When C flag = 1 upon return A register is not assured	All registers except AF

Subroutine name (hexadecimal address)	Function	Registers preserved
CALL HLHEX ($0410)	Handles a consecutive string of 4 characters in ASCII code as hexadecimal string data and returns after setting the data in the HL register. The call and return conditions are as follows. DE ← starting address of the ASCII string (string "3" "1" "A" "5") CALL HLHEX $\uparrow$—DE C flag = 0 HL ← hexadecimal number (e.g., HL = 31A5H) C flag = 1 HL is not assured.	All registers except AF and HL
CALL 2HEX ($041F)	Handles 2 consecutive ASCII strings as hexadecimal strings and returns after setting the data in A register. The call and return conditions are as follows. DE ← starting address of the ASCII string (e.g., "3" "A") CALL 2HEX $\uparrow$—DE C flag = 0 A register ← hexadecimal number (e.g., A register = 3AH) C flag = 1 A register is not assured.	All registers except AF and DE
CALL ??KEY ($09B3)	Awaits key input while causing the cursor to flash. When a key entry is made it is converted to display code and set in A register, then a return is made.	All registers except AF
CALL ?ADCN ($0BB9)	Converts an ASCII value to display code. Call and return conditions are as follows. A register ← ASCII value CALL ?ADCN A register ← display code	All registers except AF
CALL ?DACN ($0BCE)	Converts a display code to an ASCII value. Call and return conditions are as follows. A register ← display code CALL ?DACN A register ← ASCII value	All registers except AF
CALL ?DPCT ($0DDC)	Controls the display on the display screen. The relationship between A register at the time of the call and control is as follows.	All registers

A reg.	Same function	A reg.	Same function
C0H	Scrolling	C8H	$\boxed{\text{INST}}$
C1H	$\boxed{\downarrow}$	C9H	$\boxed{\text{GRPH}}$ (graphic → normal)
C2H	$\boxed{\uparrow}$	CAH	$\boxed{\text{GRPH}}$ (normal → graphic)
C3H	$\boxed{\rightarrow}$	CCH	$\boxed{\text{CTRL}}$ + $\boxed{@}$ (rev. ↔ norm.)
C4H	$\boxed{\leftarrow}$	CDH	$\boxed{\text{CR}}$ or $\boxed{\text{ENT}}$
C5H	$\boxed{\text{HOME}}$	CEH	$\boxed{\text{CTRL}}$ + $\boxed{\text{D}}$ (roll up)
C6H	$\boxed{\text{CLR}}$	CFH	$\boxed{\text{CTRL}}$ + $\boxed{\text{E}}$ (roll down)
C7H	$\boxed{\text{DEL}}$		

Subroutine name (hexadecimal address)	Function	Registers preserved
CALL ?BLNK ($0DA6)	Checks vertical blanking of the display screen. Waits until the vertical Blanking interval starts and then returns when blanking takes place.	All registers
CALL ?PONT ($0FB1)	Sets the current position of the cursor on the display screen in register HL. The return conditions are as follows. CALL ?PONT HL ← cursor position on the display screen (V-RAM address) (Note) The X-Y coordinates of the cursor are contained in DSPXY (1171 H). The current position of the cursor is loaded as floows. LD HL, (DSPXY); H ← Y coordinate on the screen L ← X coordinate on the screen The cursor position is set as follows. LD (DSPXY), HL	All registers except AF and HL

Subroutine name (hexadecimal address)	Function	Registers preserved
CALL WRINF ($0021)	Writes the current contents of a certain part of the header buffer (described later) onto the tape, starting at the current tape position. Return conditions C flag = 0 No error occurred. C flag = 1 The BREAK key was pressed.	All registers except AF
CALL WRDAT ($0024)	Writes the contents of the specified memory area onto the tape as a CMT data block in accordance with the contents of a certain part of the header buffer. Return conditions C flag = 0 No error occurred. C flag = 1 The BREAK key was pressed.	All registers except AF
CALL RDINF ($0027)	Reads the fist CMT header found starting at the current tape position into a certain part of the header buffer. Return conditions C flag = 0 No error occurred. C flag = 1, A register = 1 A check sum error occurred. C flag = 1, A register = 2 The BREAK key was pressed.	All registers except AF
CALL RDDAT ($002A)	Reads in the CMT data block according to the current contents of a certain part of the header buffer. Return conditions C flag = 0 No error occurred. C flag = 1, A register = 1 A check sum error occurred. C flag = 1, A register = 2 The BREAK key was pressed.	All registers except AF
CALL VERFY ($002D)	Compares the first CMT file found following the current tape position with the contents of the memory area indicated by its header. Return conditions C flag = 0 No error occurred. C flag = 1, A register = 1 A match was not obtained. C flag = 1, A register = 2 The BREAK key was pressed.	All registers except AF

(Note) The contents of the header buffer at the specific addresses are as follows. The buffer starts at address $10F0 and consists of 116 bytes.

Address	Contents
IBUFE ($10F0)	This byte indicates one of the following file modes. 01. Object file (machine language program) 02. BASIC text file 03. BASIC data file 04. Source file (ASCII file) 05. Relocatable file (relocatable binary file) A0. PASCAL interpreter text file A1. PASCAL interpreter data file
IBU1 ($10F1~$1101)	These 17 bytes indicate the file name. However, since 0DH is used as the end mark, in actuality the file name is limited to 16 bytes. Example: $\boxed{S}\boxed{A}\boxed{M}\boxed{P}\boxed{L}\boxed{E}\boxed{0D}$
IBU18 ($1102~$1103)	Tese two bytes indicate the byte size of the data block which is to follow.
IBU20 ($1104~$1105)	These two bytes indicate the data address of the data block which is to follow. The loading address of the data block which is to follow is indicated by "CALL RDDAT". The starting address of the memory area which is to be output as the data block is indicated by "CALL WRDAT".
IBU22 ($1106~$1107)	These two bytes indicate the execution address of the data block which is to follow.
IBU24 ($1108~$1163)	These bytes are used for supplemental information, such as comments.

Example

Address	**Content**	
10F0	01	; indicates an object file (machine language program).
10F1	'S'	; the file name is "SAMPLE".
10F2	'A'	
10F3	'M'	
10F4	'P'	
10F5	'L'	
10F6	'E'	
10F7	0D	
10F8	} Variable	
1101		
1102	00	; the size of the file is 2000H bytes.
1103	20	
1104	00	; the data address of the file is 1200H.
1105	12	
1106	50	; the execution address of the file is 1250H.
1107	12	

FDOS SUBROUTINES (FDOSEQU.LIB)

—Outline—

FDOS subroutines can be broadly divided into three groups. That is,

1. CLI (Command Line Interpreter) subroutines
2. IOCS (Input Output Control System) subroutines
3. Utility subroutines

CLI subroutines are used to translate command lines appearing within user programs. That is, when programs are called in which switches and arguments appear in appended format (such as RUN PROG/P FILE1, FILE2 $\boxed{\text{CR}}$), these subroutines translate those switches and arguments.

IOCS subroutines are used to open and close files and devices. Utility subroutines are other general purpose subroutines.

Command lines are strings of characters (which have been converted to intermediate code) which are input from the keyboard as FDOS commands or other character strings in the same format. In the explanation below, except where otherwise indicated, command lines appear in intermediate code. See the table on page 26 for the intermediate code.

—CLI (Command Line Interpreter) Subroutines—

TRS10

Function:	Converts FDOS command lines written in ASCII code into intermediate code.
Input registers:	The HL register contains the starting address of the command line written in ASCII code. The DE register contains the starting address of the area storing the command line converted to intermediate code.
Calling procedure:	CALL TRS10

Output register:	CF = 0 .... Normal
	CF = 1 .... Error (A ← error code)

Note: See the "System Error Messages" in System Command for details. The same applies below.

Registers preserved:	All registers except AF.

. CLI (Command Line Interpreter)

Function:	Translates and executes FDOS command lines.
Input registers:	The HL register contains the command line pointer.

Calling procedure:

```
LD      DE, (RJOB)
PUSH    DE
CALL    .CLI
POP     HL
LD      (RJOB), HL
```

Output registers:	CF = 0 .... Normal
	CF = 1 .... Error (A ← FFH)

Registers preserved:	None

Caution: The LIMIT, RUN, EXEC and DEBUG commands cannot be executed.
See page 25 for the RJOB.

Example of use (DATE/P)

```
       LD      HL, DATE
       LD      DE, (RJOB)
       PUSH    DE
       CALL    .CLI
       POP     HL
       LD      (RJOB), HL
       JP      C, ERROR
        :
DATE : DEFB    B1H  }  Intermediate
       DEFB    88H  }  code for DATE/P
       DEFB    0DH
```

? HEX (Check Hexadecimal)

Function:	Converts a 4-digit hexadecimal data item starting with "$" into sixteen bit, binary notation.
Input registers:	HL contains the pointer; it should specify "$".
Calling procedure:	CALL ?HEX
Output registers:	CF = 1 .... Not a hexadecimal number. (A ← 3, and HL are preserved) CF = 0 .... a hexadecimal number. (DE ← data, HL indicates the address following the hexadecimal number)
Registers preserved:	All registers except AF, DE and HL.

? SEP (Check Separator)

Function:	Checks whether the contents of the address indicated by the HL register are a separator (one of the following: $\boxed{\text{CR}}$ ␣ , : /).
Input registers:	Register HL is the pointer.
Calling procedure:	CALL ?SEP
Output registers:	CF = 1 .... Not a separator. A←3(error code) and the HL register are preserved.
	CF = 0 .... A separator.
	A = 2CH ... The separator is a space or a comma " ␣ ", " , " (the HL register then points to the address following the separator)
	A = 0DH ... The separator is $\boxed{\text{CR}}$ or slant " / " (the HL register points to the separator)
	A = 3AH ... The separator is a colon " : " (the HL register points to the separator)
Registers preserved:	All registers except AF and HL.

? GSW (Check Global Switch)

Function:	Determines whether the global switch on the command line is correct and, if so, stores it in the area within FDOS.
Input registers:	The DE register contains the starting address of the switch table. The HL register contains the command line pointer which points to the global switch.
Calling procedure:	

```
                LD      DE, SWTBL
                CALL    ?GSW
                  :
                  :
        SWTBL : DEFB    SW1      }
                DEFB    SW2      }  List of items which may be used as global switches
                  :              }  (these are written in intermediate code, from 0 to a maximum of 5.
                DEFB    SWn      }  See page 26)
                DEFB    FFH         End of table
```

Output registers:	CF = 1 .... Error (A ← error code)
	CF = 0 .... Normal. The HL register points to the address following the global switch.
Registers preserved:	All registers except AF, DE and HL.

TESW (Test Global Switch)

Function: Determines the presence or absence of the specified global switch. Subroutine "?GSW" must be called before this subroutine is used.

Input registers: None

Calling procedure:

```
CALL  TESW
DEFB  global switch
```

Output registers: CF = 0 .... The specified global switch is present.

CF = 1 .... The specified global switch is not present.

Registers preserved: All registers except AF.

Example: This routine outputs whether or not global switch /P is present to the line printer or the CRT.

```
CALL   TESW
DEFB   88H          ; intermediate code for /P
PUSH   AF
CALL   C, MSG       ; displayed on the CRT if the
                      switch is not present.
POP    AF
CCF                 ; CF ← C̄F̄
CALL   C, PMSG      ; Printed on the line printer if
                      the switch is present.
JP     C, ERROR     ; indicates a line printer error.
```

? LSW (Check Local Switch)

Function: Used to determine the local switch which is attached to the file name on the command line.

Input registers: The HL register is the command line pointer which indicates the start of the file name.

Calling procedure: CALL ?LSW

Output registers: CF = 1 .... Error (A ← error code)

CF = 0 .... Normal

ZF = 1 .... No local switch. (A ← 0)

ZF = 0 .... Local switch is present. (A ← intermediate code for the local switch)

Registers preserved: All registers except AF.

Example: Read-opens (ROPEN) a file with logical number 2 if a local switch is not present; if local switch /O is present the file is write-opened (WOPEN) with logical number 3; otherwise, an error occurs.

```
      EXX
      LD     B, 4        ; default file mode .ASC
      EXX
      CALL   ?LSW
      JP     C, ERROR
      JR     NZ, L2
      LD     C, 2        ; logical number 2
      CALL   ROPEN
      JR     L3
L2:   CP     89H         ; intermediate code for /O
      LD     A, 8        ; error code (il local switch)
      JP     NZ, ERROR
      LD     C, 3        ; logical number 3
      CALL   WOPEN
L3:   JP     C, ERROR
```

ROPEN (Read Open)

Function:	Read-opens a file (including the input/output device).
Input registers:	HL: Pointer which indicates the start of the file name.
	C : Logical number (see note 3)
	B' : Default file mode (see note 1)
Calling procedure:	CALL ROPEN
Output registers:	CF = 1 .... Error (A ← error code)
	CF = 0 .... Normal
	HL: Pointer (indicates the next separator)
	B' : File mode (see note 1)
	C' : File attribute (see note 2)
	L' : Device number
	IY: Starting address of the device table (see note 4)
Registers preserved:	Only registers BC, DE and IX.

Example (when $FD1 ; ABC)

```
        LD      HL, FL
        LD      C, 2 (logical number)
        EXX
        LD      B, 4            ( . ASC)
        EXX
        CALL    ROPEN
        CALL    C, ERR          (see page 23)
        RET     C
          :
FL :    DEFB    90H             ($ FD1)
        DEFM    ' ; ABC '
        DEFB    0DH
```

WOPEN (Write Open)

Fuction:	Write-opens a file (including an input/output device).
Input registers:	HL: Pointer which indicates the start of the file name.
	C : Logical number (see note 3)
	B' : Default file mode (see note 1)
Calling procedure:	CALL WOPEN
Output registers:	CF = 1 .... Error (A ← error code)
	CF = 0 .... Normal
	HL: Pointer (indicates the next separator)
	B' : File mode (see note 1)
	C' : File attribute (only for "0")
	L' : Device number
	IY: Starting address of the device table (see note 4)
Registers preserved:	Only registers BC, DE and IX.

Example ($PTP/PE/LF)

```
        LD      HL, PTP
        LD      C, 3 (logical number)
        EXX
        LD      B, 4            ( . ASC)
        EXX
        CALL    WOPEN
        JP      C, ERROR
          :
PTP :   DEFB    A1H             ($ PTP)
        DEFB    8FH             (/PE)
        DEFB    8CH             (/LF)
        DEFB    0DH
```

MODECK (Filemode Check)

Function:	Checks whether the file mode indicated in register B' for the file opened is correct or not.
Input registers:	Register B' contains the file mode of the opened file.
Calling procedure:	CALL MODECK
	DEFB file mode number (see page 26 concerning file modes)
Output registers:	CF = 0 .... The file mode is correct.
	CF = 1 .... The file mode is not correct. A ← error code.
Registers preserved:	All registers except AF.

(Note 1) The default file mode is the mode which is assumed when no mode is specified in the command line. The numbers enclosed in parentheses indicate the file mode number. (see page 26.)

Example:

Command line	Default file mode	Actual file mode
ABC . ASC	. ASC (4)	. ASC (4)
ABC . LIB	. RB (5)	. LIB (7)
ABC	. OBJ (1)	. OBJ (1)
ABC	. ASC (4)	. ASC (4)

(Note 2) The file attribute indicates the type of tile access, and is expressed as one of the following ASCII codes.

"0"	a file with no attribute.
"R"	a file for which reading is inhibited. (Read protected file)
"W"	a file for which writing is inhibited. (Write protected file)
"P"	a file for which both reading and writing are inhibited. (Permanent file)

However, files with the attribute "P" can be read and written if the file mode is .OBJ, The EXEC command can be executed if the file mode is .ASC.

Normally, the programmer does not need to be aware of file attributes since they are managed by FDOS.

(Note 3) Logical file numbers are numbers within FDOS which have a one-to-one correspondence with physical files opened (including input/output devices). Numbers from 1 to 249 may be used as logical numbers; however, since programs within FDOS use all of the numbers from 128 on, user programs should use only the numbers from 1 to 127 to avoid conflict.

(Note 4) An explanation of the device table is contained in "USER CODED I/O ROUTINES" in Appendix; however, except for special I/O operations, the programmer normally does not need to be aware of the contents of the device table.

GET1L (Get 1 Line)

Function:	Reads in one line from the file whose logical number is specified in the C register. The line read is one which is terminated with 0DH. The data read is stored in the area indicated by the address in the DE register. The length of the line, including 0DH, must be no more than 128 bytes.
Input registers:	The C register contains the logical number. The DE register contains the address of the area in which the data is stored.
Calling procedure:	`CALL GET1L`
Output registers:	CF = 0 .... Normal
	CF = 1, A = 0 .... File end
	CF = 1, A ≠ 0 .... Error (A ← error code)
	IY : Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC, DE, HL and IX.

GET1C (Get 1 Character)

Function:	Reads one byte from the file whose logical number is specified in the C register.
Input registers:	The C register contains the logical number.
Calling procedure:	`CALL GET1C`
Output registers:	CF = 0 .... Normal (A ← data read)
	CF = 1, A = 0 .... File end
	CF = 1, A ≠ 0 .... Error (A ← error code)
	IY : Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC, DE, HL and IX.

GETBL (Get Block)

Functions:	Read data into the address indicated in the DE register from the file whose logical number is specified in the C register; only the number of bytes of data indicated in the HL register are read in.
Input registers:	The C register contains the logical number. The DE register contains the address in which the data is to be stored. The HL register contains the number of bytes of data to be read.
Calling procedure:	`CALL GETBL`
Output registers:	CF = 0 .... Normal } DE ← address of the next block of data to be read
	CF = 1, A = 0 .... File end } HL ← number of bytes of data actually read
	CF = 1, A ≠ 0 .... Error (A ← error code)
	IY: Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC and IX.

? EOF (Check End-of-file)

Function:	Checks for the end of a read-opened file. Z flag becomes "1" when an attempt is made to read beyond the end of data.
Input registers:	The C register contains the logical number.
Calling procedure:	`CALL ?EOF`
Output registers:	CF = 1 .... Error (A ← error code)
	CF = 0, ZF = 1 .... Not file end
	CF = 0, ZF = 0.......... File end
	IY : Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC, DE, HL and IX.

PUT1C (Put 1 Character)

Function:	Outputs one byte of data to the file whose logical number is specified in the C register.
Input registers:	The C register contains the logical number. The A register contains the data to be output.
Calling procedure:	`CALL PUT1C`
Output registers:	CF = 0 .... Normal
	CF = 1 .... Error (A ← error code)
	IY : Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC, DE, HL and IX.

PUT1L (Put 1 Line)

Function:	Outputs the line starting at the address specified in the DE register to the file whose logical number is specified in the C register. Outputs the ending carriage return.
Input registers:	The C register contains the logical number. The DE register contains the starting address of the data to be output.
Calling procedure:	`CALL PUT1L`
Output registers:	CF = 0 .... Normal
	CF = 1 .... Error (A ← error code)
	IY : Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC, DE, HL and IX.

PUTBL (Put Block)

Function:	Outputs the number of bytes of data indicated in the HL register to the file whose logical number is specified in the C register, starting at the address indicated in the DE register.
Input registers:	The C register contains the logical number. The DE register contains the starting address of the data to be output. The HL register contains the number of bytes of data to be output.
Calling procedure:	CALL PUTBL
Output registers:	CF = 0 .... Normal (DE ← address following the end of the block output)
	CF = 1 .... Error (A ← error code)
	IY : Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC and IX. (Register HL is also preserved if C flag = 0)

PUTCR (Put Carriage Return)

Function:	Outputs a carriage return to the file whose logical number is specified in the C register.
Input registers:	The C register contains the logical number.
Calling procedure:	CALL PUTCR
Output registers:	CF = 0 .... Normal
	CF = 1 .... Error (A ← error code)
	IY : Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC, DE, HL and IX.

PUTM (Put Message)

PUTMX

Function:	Outputs the line starting at the address indicated in register DE to the file whose logical number is specified in the C register. PUTM and PUTMX operate in the same manner except for their handling of $CRT and $LPT. Cursor control operations (🅷, ⬆, etc.) are executed only when PUTM is used; when PUTMX is used,they are only displayed or printed as reverse characters. The end code (0DH) is not output.
Input registers:	The C register contains the logical number. The DE register contains the starting address of the data to be output.
Calling procedure:	CALL PUTM or CALL PUTMX
Output registers:	CF = 0 .... Normal
	CF = 1 .... Error (A ← error code)
	IY : Starting address of the device table (see note 4 on page 12)
Registers preserved:	Only registers BC, DE, HL and IX.

CLOSE (Close File)

KILL (Kill File)

Function:	Closes or kills the file whose logical number is specified in the C register. If this subroutine is called when the C register contains 0, all currently opened files will be closed or killed. (This excludes files which were opened by FDOS itself.)
Input registers:	The C register contains 0 or a logical number.
Calling procedure:	`CALL CLOSE` or `CALL KILL`

Output registers:	CF = 0 .... Normal
	CF = 1 .... Error (A ← error code)
	IY : Starting address of the device table (see note 4 on page 12)

Registers preserved:	Only registers BC, DE, HL and IX.

LUCHK (LU Number Check)

Function:	Checks whether a logical number (contained in the C register) has been defined.
Input registers:	The C register contains the logical number.
Calling procedure:	`CALL LUCHK`
Output registers:	CF = 1 .... The logical number has not been defined. CF = 0 .... The logical number has been defined. L' ← device number (see page 26 concerning device numbers) IY ← starting address of the device table. (see note 4 on page 12)
Registers preserved:	All registers except AF, HL, IY, D' and L'.

Example:
```
        LD      C, 5            ; logical number
        CALL    LUCHK
        JP      C, NOTUSE
        EXX
        LD      A, L            ; device number
        EXX
        CP      4
        JP      C, FD
          :
          :
```

MTOFF (Motor Off)

Function:	Stops the motor of the floppy disk drive. (The drive motor is activated automatically when necessary.)
Calling procedure:	`CALL MTOFF`
Registers preserved:	All registers except AF.

BREAK (Check Break Key)

Function:	Checks whether $\boxed{\text{BREAK}}$ has been pressed.
Input registers:	None
Calling procedure:	`CALL BREAK`
Output registers:	CF = 0 .... Not pressed.
	CF = 1 .... Pressed. (In this event, A ← 37. 37 is the error code.)
Registers preserved:	All registers except AF.

HALT (Halt Action with Break Action)

Function:	Checks the keyboard and, if the $\boxed{\text{SPACE}}$ key is pressed, stops execution until the $\boxed{\text{SPACE}}$ key is pressed again. If $\boxed{\text{BREAK}}$ is pressed, A ← 37 and CF ← 1. (37 is the error code.)
Input registers:	None
Calling procedure:	`CALL HALT`
Output registers:	CF = 0 .... Normal
	CF = 1 .... $\boxed{\text{BREAK}}$ was pressed. (In this event, A ← 37.)
Registers preserved:	All registers except AF.

SGETL (Screen Get Line)

Function:	Inputs one line from the keyboard. The line which is actually input is the line in which the cursor is located when $\boxed{\text{CR}}$ is pressed; the maximum number of characters which can be input is 80.
Input registers:	The DE register contains the starting address of the area (80 bytes required) in which the data is to be stored.
Calling procedure:	CALL SGETL
Output registers:	CF = 0 .... Normal
	CF = 1 .... $\boxed{\text{BREAK}}$ was pressed. A ← 0 (not 37)
Registers preserved:	All registers except AF.

LTPNL (Let Printer New Line)
PMSGX (Printer Message X)
PMSG (Printer Message)
PPRNT (Printer Print)
PPAGE (Printer Page)

Function: These are printer control routines. Each routine performs the same function for the printer as does the corresponding monitor subroutine shown below for the CRT.

$\boxed{\text{Printer}}$	$\boxed{\text{CRT}}$
LTPNL (carriage return)	LETNL
PMSGX	MSGX
PMSG	MSG
PPRNT	PRNT
PPAGE	———

Output registers:	CF = 0 .... Normal
	CF = 1 .... Error (A ← error code)
Registers preserved:	All registers except AF and IY.

C&L1
&NL
&PRNT
&NMSG
&MSG
&1L

Function:	Each subroutine directs output to the printer or CRT depending on the presence or absence of the global switch (/P). &NL, &NMSG and &1L include the HALT function (see page 17 for the HALT function).
C&L1	Prepares either the printer or the CRT. This routine must be called before any other routines are used. Further, "?GSW" must be called before this routine is called.
&NL	Performs the same function as LETNL.
&PRNT	Performs the same function as PRNT.
&MSG	Performs the same function as MSG.
&NMSG	Executes &NL, then executes &MSG.
&1L	Executes &MSG, then executes &NL.
Output registers:	CF = 0 .... Normal
	CF = 1 .... Error (A ← error code)
Registers preserved:	All registers except AF and IY.

See "LINKING ASSEMBLY PROGRAM WITH FDOS" in Appendix for an example of use.

CHKACC (Check Acc)

Function:	Checks whether the contents of A register (accumulator) match any of several different given data items.
Input registers:	A contains the data items to be checked.
Calling procedure:	See below

```
CALL   CHKACC
DEFB   n         ;number of data items (1-255)
DEFB   data 1  ⎫
DEFB   data 2  ⎪   n items of data to be compared
DEFB   data 3  ⎬     DEFM ' ......... ' may be used with ASCII.
               ⎪
DEFB   data n  ⎭
```

Output registers:	ZF = 1 .... One of the data items matches the contents of A.
	ZF = 0 .... No match was found.
Registers preserved:	All registers except the flags.

MULT (Multiply)

Function:	Multiplies the contents of the DE register and the HL register (handling them as 16-bit unsigned integers) and places the result in the DE register.
Input registers:	DE, HL
Calling procedure:	`CALL MULT`
Output registers:	CF = 1 .... Overflow (result cannot be expressed in 16 bits)
	CF = 0 .... Normal. The DE register contains the result of the calculation.
Registers preserved:	All registers except AF, DE and HL.

SOUND (Warning Sound)

Function:	Produces the sound "A0+ARA+AR" to indicate that an error has occurred.
Calling procedure:	`CALL SOUND`
Registers preserved:	All registers.

BINARY (Convert ASCII to Binary)

Function:	Converts an ASCII numeric string into a 16-bit unsigned integer.
Input registers:	The HL register contains the starting address of the ASCII numeric string.
Calling procedure:	CALL BINARY
Output registers:	CF = 1 .... Overflow (cannot be expressed within 16 bits)
	CF = 0 .... Normal. The DE register contains the converted data. The HL register contains the address following the end of the numeric string. If the ASCII characters indicated by HL are not a numeric string, CF ← 0 and DE ← 0.
Registers preserved:	All registers except AF, DE and HL.

Example:

```
                LD      HL, BUFFER
                CALL    BINARY
                JP      C, ERROR      ; if CF = 0, DE becomes 400H.
                  :                               HL points to 0DH.
        BUFFER: DEFM    ' 1024 '
                DEFB    0DH           ; must be an ASCII code for other than ' 0 ' – ' 9 '.
```

CASCII (Convert Binary to ASCII)

Function:	Converts a 16-bit unsigned integer into an ASCII numeric string.
Input registers:	The HL register contains the 16-bit unsigned integer. The DE register contains the address of the area in which the ASCII numeric string is to be stored.
Calling procedure:	CALL CASCII
Output registers:	The DE register contains the ending address of the ASCII numeric string obtained.
Registers preserved:	All registers except AF and DE.

Example:

```
                LD      HL, 1024
                LD      DE, BUFFER
                CALL    CASCII
                  :
        BUFFER: DEFS    10            ; after conversion the ASCII numeric string ' 1024 '
                                        is stored.
```

CLEAR (Clear Area)

Function:	Loads a continuous area in the memory with zeros. (The memory area must be 255 bytes or less.)
Input registers:	None
Calling procedure:	`CALL CLEAR` `DEFB length` ; number of bytes to be cleared. `DEFW address` ; the memory is cleared starting at this address.
Output registers:	None
Registers preserved:	All registers.

CHLDE (Compare HL, DE)

Function:	Compares the contents of the HL register with the contents of the DE register.
Input registers:	HL and DE
Calling procedure:	`CALL CHLDE`

Output registers:	FLAG ← HL − DE; that is CF = 0, ZF = 0 .... HL > DE
	CF = 1, ZF = 0 .... HL < DE
	CF = 0, ZF = 1 .... HL = DE

Registers preserved:	All registers except AF.

LCHK (Limit Check)

Function:	Compares the last usable memory area (the address indicated by the stack pointer minus 256) with the contents of the HL register.
Input registers:	HL
Calling procedure:	`CALL LCHK`

Output registers:	CF = 0 .... HL <= SP−256
	CF = 1 .... HL > SP−256
	At this time, A ← 21. 21 is an error code. (memory protect error)

Registers preserved:	All registers except AF.

ERR (Display Error Message)

Function: Displays an error message (see the System Error Messages in System Command for details). The contents of the C register and the IY register must be preserved from the time the error occurs until this routine is called. Further, the CLOSE or KILL routine must not be called during that time (otherwise, the contents of the error message may be incorrect).

Input registers: The A register contains the error code (no error message is output if the error code is FFH).

The C register contains the logical number.
The IY register contains the starting address of the device table (see note 4 on page 12) } These may not be necessary depending on the type of error.

Calling procedure: CALL ERR

Output registers: A ← FFH

CF ← 1

Registers preserved: All registers except AF.

Example:

```
CALL   SGETL       (Page 18)
CALL   NC, & 1L    (Page 19)
JR     NC, – 6
CALL   C, ERR
RET    C
```

ERRX

Function: This function displays a colon (" : "), followed by the contents of the area from the address following a specified 0DH until the next 0DH; the specified 0DH is the one which is the (ACC–1)th from the address indicated in the DE register.

Input registers: The DE register contains the starting address of the message block.
The A register contains a number (1–255).

Calling procedure: CALL ERRX

Output registers: A ← FFH
CF ← 1

Registers preserved: All registers except AF.

Example:

```
ERMSG:  DEFM   ' SYNTAX '
        DEFB   0DH
        DEFM   ' OVERFLOW '
        DEFB   0DH
        DEFM   ' IL DATA '
        DEFB   0DH
          :
        LD     A, 2
        LD     DE, ERMSG
        CALL   ERRX
```

This displays ' : OVERFLOW '.

ERWAIT (Display Error Message and Wait Space Key)

Function:

1. Calls subroutine ERR if A ≠ 0.
2. Displays the contents of the area starting with the address indicated in the DE register until 0DH.
3. Displays ", ⇩ space key" if A = 0.
4. Waits until SPACE or BREAK is pressed.

Input registers: A and DE

Calling procedure: CALL ERWAIT

Output registers: CF = 0 .... SPACE was pressed.

CF = 1 .... BREAK was pressed. (A ← 37)

Registers preserved: All registers except AF.

LIMIT (Limit of Memory)

Number of bytes:	2
Meaning:	Contains the last address plus one of RAM mounted.

ISTACK (Initial Stack Pointer)

Number of bytes:	2
Meaning:	Contains the last address plus one of the memory area which is available to FDOS. This data is used by FDOS for initialization of the stack pointer. The contents of ISTACK may be changed by the FDOS LIMIT instruction. The contents of ISTACK must not be changed by any other means.

ZMAX

Number of bytes: 2

Meaning: Contains the last address of the area being used by FDOS (excluding the stack). The contents of ZMAX may be changed depending on the next subroutine called. (ROPEN, WOPEN, CLOSE, KILL, . CLI)

Caution: The area which may be used within the user program as free area is as follows.

1. [Lowest address] = [value contained in ZMAX when the user program was entered]
   + [number of files which are simultaneously opened (ROPEN or WOPEN)] x 350
   + [number of files which are simultaneously write-opened] x 142
   + [number of floppy disk units used] x 256

   [Maximum address] = [stack pointer (SP)] $-\alpha$, α is approximately 256.
2. From ISTACK to LIMIT−1.
3. Area reserved by the DEFS statement within the assembly program.

.DNAME (Default File Name)

Number of bytes:	17
Meaning:	The file name and succeeding 0DH contained in this area will be used as the default file name when the file name is omitted. For example, when this area contains "ABCD $\boxed{\text{CR}}$", "$FD3" appearing on the command line will be interpreted as "$FD3;ABCD".

BDRIVE (Boot Drive)

Number of bytes:	1
Meaning:	Contains the default drive number minus 1 (0–3). The default drive number is the number which appears to the left of the prompt " > " when FDOS is in the command wait state.

MAXDVR (Maximum Drive)

Number of bytes:	1
Meaning:	Contains the number of floppy disk dirves connected (1–4).

TODAY

Number of bytes:	7
Meaning:	Contains the month, day and year followed by 0DH; each element of the date is indicated with a two-digit ASCII code.

RJOB (Running Job Pointer)

Number of bytes:	2
Meaning:	Area which indicates how far command line interpretation has proceeded. When command lines are interpreted in a user program, the address following that of the last command line interpreted must be placed in RJOB.

—CLI Intermediate Code Table—

Switch

ASCII	Intermediate code
	80 H
/D	81 H
/C	82 H
/E	83 H
/G	84 H
/L	85 H
/N	86 H
/S	87 H
/P	88 H
/O	89 H
/T	8AH
	8BH
/LF	8CH
/PN	8DH
/PO	8EH
/PE	8FH

Device name

ASCII	Device number	Intermediate code	ASCII	Device number	Intermediate code
$FD1	0	90 H	$LPT	16	A0H
$FD2	1	91 H	$PTP	17	A1H
$FD3	2	92 H	$CRT	18	A2H
$FD4	3	93 H		19	A3H
$CMT	4	94 H	$SOA	20	A4H
$MEM	5	95 H	$SOB	21	A5H
	6	96 H		22	A6H
	7	97 H		23	A7H
	8	98 H	$USR1	24	A8H
	9	99 H	$USR2	25	A9H
$PTR	10	9AH	$USR3	26	AAH
	11	9BH	$USR4	27	ABH
$KB	12	9CH		28	ACH
$SIA	13	9DH		29	ADH
$SIB	14	9EH		30	AEH
	15	9FH		31	AFH

Built-in commands

ASCII	Intermediate code	ASCII	Intermediate code
RUN	B0H		C2H
DATE	B1H		C3H
XFER	B2H	BOOT	C4H
DIR	B3H		
	B4H		
RENAME	B5H		
DELETE	B6H		
TYPE	B7H		
CHATR	B8H		
FREE	B9H		
MON	BAH		
TIME	BBH		
EXEC	BCH		
	BDH		
	BEH		
POKE	BFH		
	C0H		
	C1H		

File mode

ASCII	File mode number	Intermediate code
. *	255	F0H
		F1H
. OBJ	1	F2H
. BTX	2	F3H
	3	F4H
. ASC	4	F5H
. RB	5	F6H
	6	F7H
. LIB	7	F8H
	8	F9H
	9	FAH
. SYS	10	FBH
	11	FCH
	12	FDH
	13	FEH
	14	FFH

Other

Codes other than those shown in this table are expressed as is in ASCII code. However, this applies only to 01H-7FH. The codes for some small characters and graphic characters are the same as CLI intermediate codes; therefore, they cannot be used.

BASIC RELOCATABLE LIBRARY (RELO.LIB)

The basic relocatable library contains a collection of subroutines which are required by programs created using the basic compiler. These routines are useful when basic program subroutines (external functions, external commands, etc.) are created using the assembler.

Routines contained in RELO . LIB can only be used as basic subroutines; they cannot be executed as independent assembly programs.

. .INT0
. .INT1
. .INT2 (Convert Floating to Fixed)

Function:	Converts a real number expressed in 5 bytes into a 16-bit integer. The absolute value or any decimal fraction is discarded. (Examples: 1.5 → 1 −2.7 → −2)
	. . INT0 .... The input range is from −32768 ~ 32767
	. . INT1 .... The input range is from 0 ~ 255
	. . INT2 .... The input range is from −32768 ~ 65535
Input registers:	The HL register contains the starting address of the 5 byte real number.
Calling procedure:	CALL . . INT0 CALL . . INT1 CALL . . INT2
Output registers:	HL ← integer
Error processing:	. . INT0 .... Upon overflow, CF ← 1.
	. . INT1 .... Upon overflow, JP ER3.
	. . INT2 .... Upon overflow, CF ← 1.
Registers preserved:	All registers except AF and HL.

Note: The . .FLT0 and CONST subroutines (described below) are used to create the 5-byte real number.

. .FLT0 (Convert Fixed to Floating)

Function:	Converts a 16-bit signed integer into a 5-byte real number.
Input registers:	The HL register contains the 16-bit signed integer. The DE register contains the starting address of the area in which the real number is stored.
Calling procedure:	CALL . .FLT0
Registers preserved:	All registers except AF, DE and HL.

CASC' (Change ASCII)

Function:	Converts a 16-bit unsigned integer into an ASCII character string and appends 0DH to the end of it.
Input registers:	The HL register contains the 16-bit unsigned integer. The DE register contains the starting address of the area in which the ASCII character string is stored.
Calling procedure:	CALL CASC'
Registers preserved:	All registers except AF.

.MOVE' (Move String)

Function:	Converts a character string from type 1 to type 2. The converted character string is stored in an area called .WORD. (The type 1 and type 2 character string formats are explained on page 31.)
Input registers:	The HL register contains the starting address of the character string (type 1).
Calling procedure:	CALL .MOVE'
Output registers:	The DE register contains the starting address of the converted character string. (The address of .WORD)
Registers preserved:	All registers except AF, BC, DE and HL.

FASCX (Convert Floating to ASCII)

Function:	Converts a 5-byte real number into an ASCII character string and appends 0DH to the end of it.
Input registers:	The HL register contains the starting address of the real number. The DE register contains the starting address of the area in which the ASCII character string is stored.
Calling procedure:	CALL FASCX
Registers preserved:	None

CONST (Convert ASCII to Const)

Function:	Converts a constant expressed in ASCII code into a 5-byte real number.
Input registers:	The HL register contains the starting address of the constant expressed in ASCII code. The DE register contains the starting address of the area in which the result is stored.
Calling procedure:	CALL CONST
Output registers:	The HL register contains the first address following the constant converted.
Registers preserved:	None
Error processing:	JP ER3

CHCOND (Character Condition)

Function:	Compares the two character strings (type 1.)
Input registers:	The HL and DE registers contain the strarting addresses of each of the two character strings being compared.
Calling procedure:	CALL CHCOND

Output registers:	FLAG ← (DE) – (HL)
	that is,
	CF = 0, ZF = 0 .... (DE) > (HL)
	CF = 1, ZF = 0 .... (DE) < (HL)
	CF = 0, ZF = 1 .... (DE) = (HL)

Registers preserved:	All registers except AF, BC, DE and HL.

ER1 ER13
ER2 ER14
ER3 ER21
ER4 ER24
ER5 ER37
ER6 ER64

Function:	Error message display routine used during BASIC program execution. See the Error Message table in the BASIC compiler instruction manual (available separately).
Calling procedure:	JP ER1 (SYNTAX ERROR), etc.

BEERR (Basic Executing Error)

Function:	Error message display routine used during BASIC program execution.

Calling procedure:	`CALL BEERR`
	`DEFB` error code (error number in BASIC)
	`DEFM ' ERROR MESSAGE '`
	`DEFB 0DH`
	→ No return made.

BABORT (Basic Abort)

Function:	When a system error occurs during BASIC program execution, this routine displays the applicable error message and interrupts execution.
Input registers:	The A register contains the error code (system error number). The C register is the logical number. The IY register contains the starting address of the device table (see note 4 on page 12). } May not be required depending upon the type of error.
Calling procedure:	`JP BABORT`

Example:	`LD C, 2`
	`CALL GET1C`
	`JP C, BABORT`

Caution:	BEERR is a routine which displays ∗ER nn: message in linenumber (where nn is the error number in BASIC compiler) when an error occurs in a BASIC program; BABORT is a routine which displays –ERR message in linenumber when an error occurs at the FDOS level. ON ERROR processing will be performed in both cases, if specified.

. .STOP

Function:	Interrupts BASIC program execution. (Corresponds to the STOP instruction of the BASIC compiler.)
Calling procedure:	JP . .STOP

...END

Function:	Terminates BASIC program execution. (This corresponds to the END instruction of the BASIC compiler.)
Calling procedure:	JP . . END

.WORD

Function:	257-byte general purpose area.

—Type 1 and Type 2 Character String Formats—

There are two types of character strings which are handled by BASIC; these should be used as appropriate.

Type 1

```
DEFB  length          (character string length:  0 ~ 255)
DEFM  '........'
```

Type 2

```
DEFM  '........'
DEFB  0DH
```

INDEX OF LIBRARY NAMES

Name	Type	Page
&1L	UTYL	19
&MSG	″	19
&NMSG	″	19
&NL	″	19
&PRNT	″	19
END	RELO	31
..FLT0	″	27
..INT0	″	27
..INT1	″	27
..INT2	″	27
..STOP	″	31
.CLI	CLI	8
.DNAME	VAR	25
.MOVE′	RELO	28
.WORD	″	31
2HEX	MON	4
??KEY	″	4
?DPCT	″	4
?EOF	IOCS	14
?GSW	CLI	9
?LSW	″	10
?HEX	″	8
?PONT	MON	5
?SEP	CLI	9
ASCI	MON	3
BABORT	RELO	30
BEERR	″	30
BELL	MON	2
BDRIVE	VAR	25
BINARY	UTYL	21
BREAK	″	17
BRKEY	MON	3
C & L1	UTYL	19
CASC′	RELO	28
CASCII	UTYL	21
CHCOND	RELO	29
CHKACC	UTYL	20
CHLDE	″	22
CLEAR	″	22
CLOSE	IOCS	16
CONST	RELO	29
ER1	RELO	29
ER2	″	29
ER3	″	29
ER4	″	29
ER5	″	29
ER6	″	29
ER13	″	29
ER14	″	29
ER21	″	29
ER24	″	29
ER37	″	29
ER64	″	29
ERR	UTYL	23
ERRX	″	23
ERWAIT	″	23
FASCX	RELO	28
GET1C	IOCS	13
GET1L	″	13
GETBL	″	13
GETL	MON	3
GETKY	″	3
HALT	UTYL	17
HEX	MON	3
HLHEX	″	4
IBU1	″	6
IBU18	″	6
IBU20	″	6
IBU22	″	6
IBU24	″	6
IBUFE	″	6
ISTACK	VAR	24
KILL	IOCS	16
LCHK	UTYL	22
LETNL	MON	2
LIMIT	VAR	24
LTPNL	UTYL	18
LUCHK	IOCS	16
MAXDVR	VAR	25
MELDY	MON	2
MODECK	IOCS	12
MSG	MON	2
MSGX	″	2
MTOFF	UTYL	17
MULT	″	20
NL	MON	2
PMSG	UTYL	18
PMSGX	″	18
PPAGE	″	18
PPRNT	″	18
PRNT	MON	2
PRNTS	″	2
PRTHL	″	3
PRTHX	″	3
PUSHR	″	5
PUSHR2	″	5
PUT1C	IOCS	14
PUT1L	″	14
PUTBL	″	15
PUTCR	″	15
PUTM	″	15
PUTMX	″	15
RDDAT	MON	5
RDINF	″	5
RJOB	VAR	25
ROPEN	IOCS	11
SGETL	UTYL	18
SOUND	″	20
TESW	CLI	10
TIMRD	MON	2
TIMST	″	2
TODAY	VAR	25
TRS10	CLI	8
VERFY	MON	5
WOPEN	IOCS	11
WRDAT	MON	5
WRINF	″	5
XTEMP	″	2
ZMAX	VAR	24

Type: MON .... Monitor subroutine
CLI .... CLI subroutine
IOCS .... IOCS subroutine
UTYL .... Utility subroutine
VAR .... FDOS common variable
RELO .... BASIC relocatable library

(CLI, IOCS, UTYL, VAR:) FDOS subroutines