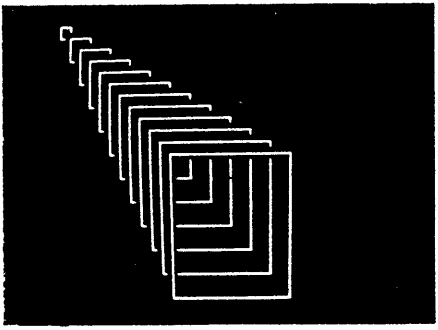
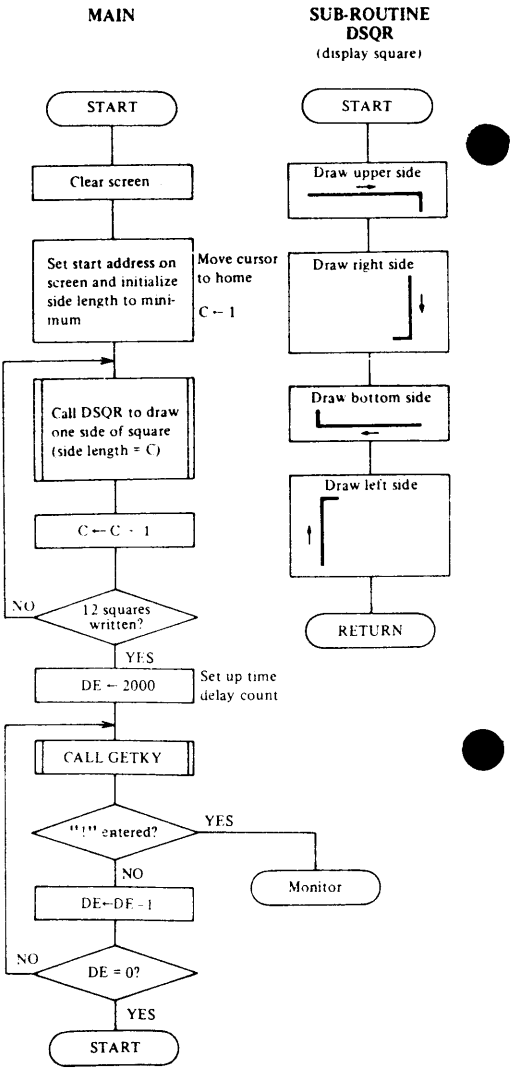
CHAPTER 4
SAMPLE PROGRAMS

4.1 DRAWING AN APPROACHING SQUARE

Let us prepare a program which draws squares which get bigger and bigger as they approach. We want a small square drawn at the upper left corner of the screen to appear as if it were approaching us. This can be accomplished by drawing squares with sides of increasing length one over another with a slight displacement.



The photo above shows a square which apparently approaches us as it grows bigger and bigger.



```
0001: 0000                ;
0002: 0000                ; APPROACHING SQUARE
0003: 0000                ;
0004: 0000 (00E0)         CHR1:   EQU     E0H
0005: 0000 (00CE)         CHR2:   EQU     CEH
0006: 0000 (00FD)         CHR3:   EQU     FDH
0007: 0000 (00DD)         CHR4:   EQU     DDH
0008: 0000 (00CD)         CHR5:   EQU     CDH
0009: 0000 (00D0)         CHR6:   EQU     D0H
0010: 0000 (0000)         MNTR:   EQU     0000H
0011: 0000 (001B)         GETKY:  EQU     001BH
0012: 0000 (0012)         PRNT:   EQU     0012H
0013: 0000                ;
0014: 0000 119500         3DG0:   LD      DE,DSP1
0015: 0003 CD8C00                 CALL    CPRNT
0016: 0006 0E01                   LD      C,1
0017: 0008 41             3DG1:   LD      B,C
0018: 0009 CD3900                 CALL    DSQR
0019: 000C CD8300                 CALL    TMDLY
0020: 000F 3E13                   LD      A,13H
0021: 0011 CD1200                 CALL    PRNT
0022: 0014 3E11                   LD      A,11H
0023: 0016 CD1200                 CALL    PRNT
0024: 0019 0C                     INC     C
0025: 001A 79                     LD      A,C
0026: 001B FE0D                   CP      13
0027: 001D 20E9                   JR      NZ,3DG1
0028: 001F 110020                 LD      DE,2000H
0029: 0022 1B             3DG2:   DEC     DE
0030: 0023 CD1B00                 CALL    GETKY
0031: 0026 FE21                   CP      '!'
0032: 0028 CA3100                 JP      Z,MNTR1
0033: 002B 7A                     LD      A,D
0034: 002C B3                     OR      E
0035: 002D 20F3                   JR      NZ,3DG2
0036: 002F 18CF                   JR      3DG0
0037: 0031 C30000         MNTR1:  JP      MNTR
0038: 0034                ;
0039: 0034                ; SUB ROUTINE
0040: 0034                ;
0041: 0034 3EE0           DSQR0:  LD      A,CHR1
0042: 0036 CD1200                 CALL    PRNT
0043: 0039 10FB           DSQR:   DJNZ    DSQR0
0044: 003B 3ECE                   LD      A,CHR2
0045: 003D 41                     LD      B,C
0046: 003E 1802                   JR      +4
0047: 0040 3EFD           DSQR1:  LD      A,CHR3
0048: 0042 CD1200                 CALL    PRNT
0049: 0045 3E11                   LD      A,11H
0050: 0047 CD1200                 CALL    PRNT
0051: 004A 3E14                   LD      A,14H
0052: 004C CD1200                 CALL    PRNT
0053: 004F 10EF                   DJNZ    DSQR1
0054: 0051 3EDD                   LD      A,CHR4
0055: 0053 41                     LD      B,C
0056: 0054 1802                   JR      +4
0057: 0056 3EE0           DSQR2:  LD      A,CHR1
0058: 0058 CD1200                 CALL    PRNT
0059: 005B 3E14                   LD      A,14H
0060: 005D CD1200                 CALL    PRNT
```

Defines graphic characters for drawing squares.

Symbol	Character
CHR1	⊟
CHR2	◱
CHR3	⊞
CHR4	◲
CHR5	◳
CHR6	◱

Defines monitor subroutine addresses.

Clears screen and positions cursor at initial position.

Determines side length of square to be drawn first.

Draws one square whose side length is specified in C, then determines next position after a delay.

Increments C (which contains the side length). Exits this loop when C reaches 13.

Gets character with a delay.

Returns to monitor if "!" is entered, otherwise, returns to beginning of the program.

Draws top side of square.

Draws right side.

Draws bottom side.

```
0061: 0060 3E14          LD      A,14H
0062: 0062 CD1200        CALL    PRNT
0063: 0065 10EF          DJNZ    DSQR2
0064: 0067 3ECD          LD      A,CHR5
0065: 0069 41            LD      B,C
0066: 006A 1802          JR      +4
0067: 006C 3EFD  DSQR3:  LD      A,CHR3
0068: 006E CD1200        CALL    PRNT
0069: 0071 3E12          LD      A,12H
0070: 0073 CD1200        CALL    PRNT
0071: 0076 3E14          LD      A,14H
0072: 0078 CD1200        CALL    PRNT
0073: 007B 10EF          DJNZ    DSQR3
0074: 007D 3ED0          LD      A,CHR6
0075: 007F CD1200        CALL    PRNT
0076: 0082 C9            RET
0077: 0083               ;
0078: 0083 110020 TMDLY: LD      DE,2000H
0079: 0086 1B            DEC     DE
0080: 0087 7A            LD      A,D
0081: 0088 B3            OR      E
0082: 0089 20FB          JR      NZ,-3
0083: 008B C9            RET
0084: 008C               ;
0085: 008C       CPRNT:  ENT
0086: 008C 1A            LD      A,(DE)
0087: 008D B7            OR      A
0088: 008E C8            RET     Z
0089: 008F CD1200        CALL    PRNT
0090: 0092 13            INC     DE
0091: 0093 18F7          JR      CPRNT
0092: 0095               ;
0093: 0095 16    DSP1:   DEFB    22
0094: 0096 11            DEFB    17
0095: 0097 13            DEFB    19
0096: 0098 00            DEFB    00H
0097: 0099               END
```

Draws left side and returns.

Loads repeat count (2000H) into DE register pair.

Prints message designated by DE register pair.

Data area.

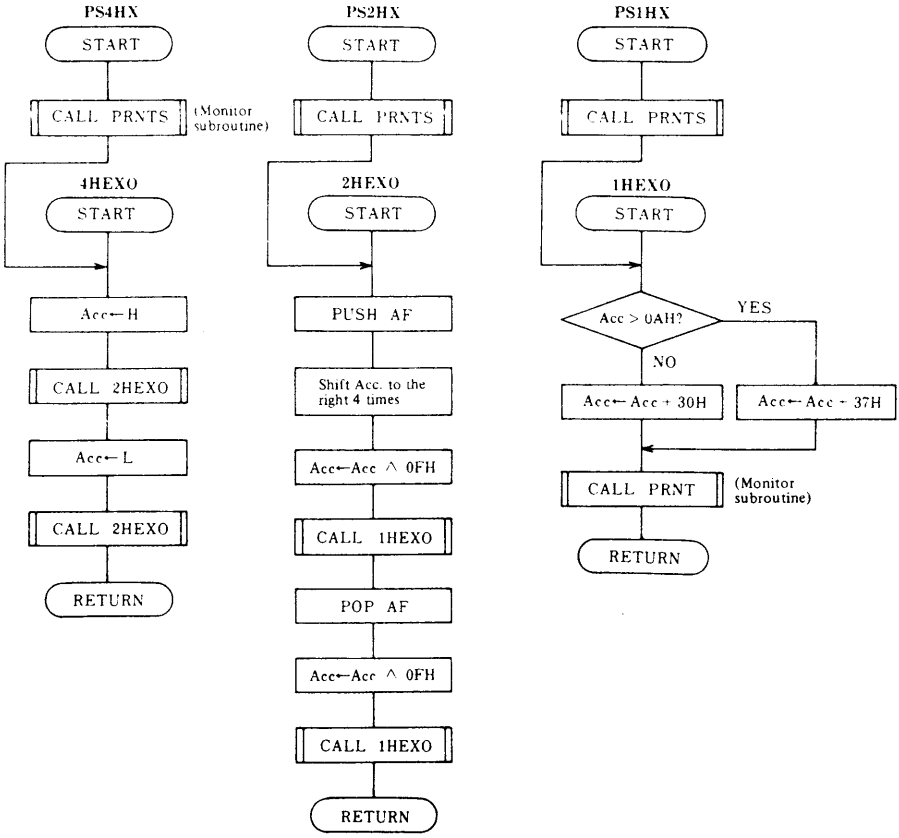
4.2 DISPLAYING BINARY DATA IN HEXADECIMAL REPRESENTATION

Let us construct a subprogram to display binary data in hexadecimal. The subprogram must display the contents of the HL register pair as a 4-digit hexadecimal number, the contents of the accumulator as a 2-digit hexadecimal number, and the lower 4 bits of the accumulator as a 1-digit hexadecimal number. The subprogram must also place a space before the displayed number.

The subprogram has six entry points as follows:

CALL 4HEXO	(4hexa data out)	: Displays the HL contents.
CALL PS4HX	(print space, 4hexa data out)	: Displays a space and the HL contents.
CALL 2HEXO	(2hexa data out)	: Dispalys the Acc. contents.
CALL PS2HX	(print space, 2hexa data out)	: Displays a space and the Acc. contents.
CALL 1HEXO	(1hexa data out)	: Displays the lower 4 bits of Acc.
CALL PS1HX	(print space, 1hexa data out)	: Displays a space and the lower 4 bits of Acc.

The above subprograms are closely related to one another; 4HEXO calls 2HEXO twice and 2HEXO calls 1HEXO twice. The program flows are as shown below.



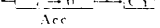
```
** SHARP Z80 ASSEMBLER  1Z-018A V1.0A   PAGE 01

0001: 0000              *          MACRO   SVC
0002: 0000              *          RST     3
0003: 0000              *          DEFB    @1
0004: 0000              *          ENDM
0005: 0000 (0003)         .CRT1C:  EQU     03H
0006: 0000 (0002)         .CR2:    EQU     02H
0007: 0000 (000D)         .INKEY:  EQU     0DH
0008: 0000 (003E)         BELL:    EQU     003EH
0009: 0000 (000C)         PRNTS:   EQU     000CH
0010: 0000                ;
0011: 0000                ;4 HEXA DATA OUT (DESTROYED:A)
0012: 0000                ; CALL 4HEX0
0013: 0000                ; CALL PS4HX (PRINT SPACE)
0014: 0000                ;
0015: 0000                PS4HX:   ENT
0016: 0000 F5                      PUSH    AF
0017: 0001 CD0C00                  CALL    PRNTS
0018: 0004 F1                      POP     AF
0019: 0005                4HEX0:   ENT
0020: 0005 7C                      LD      A,H
0021: 0006 CD1300                  CALL    2HEX0
0022: 0009 7D                      LD      A,L
0023: 000A CD1300                  CALL    2HEX0
0024: 000D C9                      RET
0025: 000E                ;
0026: 000E                ;2 HEXA DATA OUT (DESTROYED:A)
0027: 000E                ;  CALL 2HEX0
0028: 000E                ;  CALL PS2HX
0029: 000E                ;
0030: 000E                PS2HX:   ENT
0031: 000E F5                      PUSH    AF
0032: 000F CD0C00                  CALL    PRNTS
0033: 0012 F1                      POP     AF
0034: 0013                2HEX0:   ENT
0035: 0013 F5                      PUSH    AF
0036: 0014 0F                      RRCA
0037: 0015 0F                      RRCA
0038: 0016 0F                      RRCA
0039: 0017 0F                      RRCA
0040: 0018 E60F                    AND     0FH
0041: 001A CD2900                  CALL    1HEX0
0042: 001D F1                      POP     AF
0043: 001E E60F                    AND     0FH
0044: 0020 CD2900                  CALL    1HEX0
0045: 0023 C9                      RET
0046: 0024                ;
0047: 0024                ;1 HEXA DATA OUT (DESTROYED:A)
0048: 0024                ;  CALL 1HEX0
0049: 0024                ;  CALL PS1HX
0050: 0024                ;
0051: 0024                PS1HX:   ENT
0052: 0024 F5                      PUSH    AF
0053: 0025 CD0C00                  CALL    PRNTS
0054: 0028 F1                      POP     AF
0055: 0029                1HEX0:   ENT
0056: 0029 FE0A                    CP      0AH
0057: 002B 3005                    JR      NC,HXT1
0058: 002D C630                    ADD     A,30H
0059: 002F *               HXT0:    SVC     .CRT1C
      002F DF              *        RST     3
      0030 03              *        DEFB    .CRT1C
      0031                 *        ENDM
0060: 0031 C9                      RET
0061: 0032 C637           HXT1:    ADD     A,37H
0062: 0034 18F9                    JR      HXT0
0063: 0036                         END
```

Calls 2HEX0 with H and L data in Acc.

Rotates Acc to the right 4 times.

RRC A



Acc

Calls 1HEX0 twice to display the lower 4 bits of Acc.

Adds 30H to convert digits 0 to 9 to their ASCII representations.

Adds 37H to convert digits A to F to their ASCII representations.

Monitor subroutines PRNTS and PRNT must be defined in another program unit for the above subroutines to function properly.

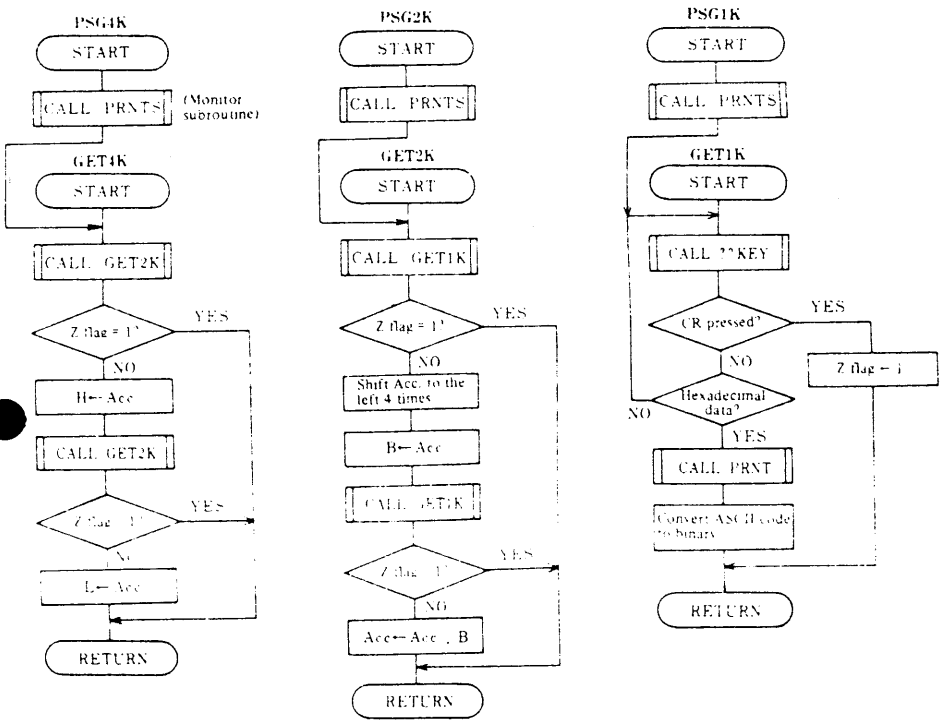
4.3 ENTERING HEXADECIMAL DATA

Let us construct a subprogram to read hexadecimal data from the keyboard, with the cursor to blink when prompting for data. Data is to be entered as one, two, or four digits, and the cursor is to flash until the required number of digits have been entered. A subprogram is to generate a beeper tone when an invalid code is entered, and the subprogram is to return with the Z flag set when a carriage return is entered.

The subprogram has six entry points as follows:

CALL GET4K	(get 4hexa data)	: Enters a 4-digit hexadecimal number into the HL register pair.
CALL PSG4K	(print space, get 4hexa data)	: Prints a space, then enters a 4-digit hexadecimal number into the HL register pair.
CALL GET2K	(get 2hexa data)	: Enters a 2-digit hexadecimal number into Acc.
CALL PSG2K	(print space, get 2hexa data)	: Prints a space, then enters a 2-digit hexadecimal number into Acc.
CALL GET1K	(get 1hexa data)	: Enters a 1-digit hexadecimal number into the lower 4 bits of Acc.
CALL PSG1K	(print space, get 1hexa data)	: Prints a space, then enters a 1-digit Hexadecimal number into lower 4 bits of Acc.

The above subprograms are related to one another: GET4K calls GET2K twice and GET2K calls GET1K twice. GET1K also calls monitor subroutine "??KEY", which waits for character input while flashing the cursor. The flowcharts for these subprograms are as shown below.



```
0001: 0000              *       MACRO   $VC
0002: 0000              *       RST     2
0003: 0000              *       DEFB    @1
0004: 0000              *       ENDM
0005: 0000 (0003)         .CRT1C: EQU     03H
0006: 0000 (0002)         .CR:    EQU     02H
0007: 0000 (000D)         .INKEY: EQU     0DH
0008: 0000 (003E)         BELL:   EQU     003EH
0009: 0000 (000C)         PRNTS:  EQU     000CH
0010: 0000                ;GET 4 CHARACTER(DESTROYED:A,H,L
0011: 0000                ; CALL PSG4K
0012: 0000                ;   EXIT:HL<--XXXX X:HEXA
0013: 0000                ;
0014: 0000                PSG4K:  ENT
0015: 0000 F5                     PUSH    AF
0016: 0001 CD0C00                 CALL    PRNTS
0017: 0004 F1                     POP     AF
0018: 0005                GET4K:  ENT
0019: 0005 CD1500                 CALL    GET2K
0020: 0008 C8                     RET     Z
0021: 0009 67                     LD      H,A
0022: 000A CD1500                 CALL    GET2K
0023: 000D C8                     RET     Z
0024: 000E 6F                     LD      L,A
0025: 000F C9                     RET
0026: 0010                ;
0027: 0010                ; GET 2 CHARACTER(DESTROYED:A)
0028: 0010                ;       CALL GET2K
0029: 0010                ;       CALL PSG2K
0030: 0010                ;         EXIT:A<--XX X:HEXA
0031: 0010                ;
0032: 0010                PSG2K:  ENT
0033: 0010 F5                     PUSH    AF
0034: 0011 CD0C00                 CALL    PRNTS
0035: 0014 F1                     POP     AF
0036: 0015                GET2K:  ENT
0037: 0015 CD2B00                 CALL    GET1K
0038: 0018 C8                     RET     Z
0039: 0019 07                     RLCA
0040: 001A 07                     RLCA
0041: 001B 07                     RLCA
0042: 001C 07                     RLCA
0043: 001D C5                     PUSH    BC
0044: 001E 47                     LD      B,A
0045: 001F CD2B00                 CALL    GET1K
0046: 0022 2802                   JR      Z,+4
0047: 0024 B0                     OR      B
0048: 0025 04                     INC     B
0049: 0026 C1                     POP     BC
0050: 0027 C9                     RET
0051: 0028                ;
0052: 0028                ; GET 1 CHARACTER(DESTROYED:A)
0053: 0028                ;       CALL GET1K
0054: 0028                ;       CALL PSG1K
0055: 0028                ;         EXIT:A<--0X X:HEXA
0056: 0028                ;
0057: 0028                PSG1K:  ENT
0058: 0028 CD0C00                 CALL    PRNTS
0059: 002B                GET1K:  ENT
0060: 002B 3E01                   LD      A,1
0061: 002D *                      $VC     .INKEY
      002D DF             *       RST     3
      002E 0D             *       DEFB    .INKEY
      002F                *       ENDM
0062: 002F FE0D                   CP      0DH
0063: 0031 C8                     RET     Z
0064: 0032 F5                     PUSH    AF
0065: 0033 FE30                   CP      '0'
0066: 0035 381B                   JR      .C,GGG2
0067: 0037 FE3A                   CP      3AH
0068: 0039 3007                   JR      NC,GGG0
0069: 003B *                      $VC     .CRT1C
      003B DF             *       RST     3
      003C 03             *       DEFB    .CRT1C
      003D                *       ENDM
```

Calls GET2K twice.

Loads 4 bits into Acc. by calling GET1K, rotates Acc. 4 bits to the left to move data into the higher 4 bits in Acc., and saves Acc. data into B.

Calls GET1K once more to load 4 bits into Acc. and merges two digits in Acc. by ORing Acc. with B; INC B is used to reset Z flag.

Returns if CR is entered.

Checks whether input data is a hexadecimal character, if so, converts it to binary and loads converted data into the lower 4 bits of Acc.

```
** SHARP Z80 ASSEMBLER  IZ-018A V1.0A   PAGE 02

0070: 003D F1                      POP     AF
0071: 003E D630                    SUB     30H
0072: 0040 180D                    JR      GGG1
0073: 0042 FE41           GGG0:    CP      'A'
0074: 0044 380C                    JR      C,GGG2
0075: 0046 FE47                    CP      47H
0076: 0048 3008                    JR      NC,GGG2
0077: 004A *                       SVC     .CRT1C
      004A DF             *        RST     3
      004B 03             *        DEFB    .CRT1C
      004C                *        ENDM
0078: 004C F1                      POP     AF
0079: 004D D637                    SUB     37H
0080: 004F FEF0           GGG1:    CP      F0H
0081: 0051 C9                      RET
0082: 0052 F1             GGG2:    POP     AF
0083: 0053 CD3E00                  CALL    BELL
0084: 0056 18D3                    JR      GET1K
0085: 0058                         END
```

Returns with Z flag reset.

Generates a beeper tone if an invalid code is input

The above program uses the following subroutines and monitor functions.

PRNTS	000CH	(CALL PRNTS)
BELL	003EH	(CALL BELL)
.INKEY	0DH	(SVC .INKEY)
.CRTIC	03H	(SVC .CRTIC)

For this subprogram to be used as a subroutine, the above addresses must be defined in the calling program.

[Application]

Add a delete function to the above subprogram to make it possible to cancel invalid key entries. For example, if

3 A [CLR]

were entered during execution of the GET4K subroutine, all that would be required to change character A to character B would be to backspace with the DEL key once to delete character A, then to enter character B. Prepare a subroutine which performs as stated above.

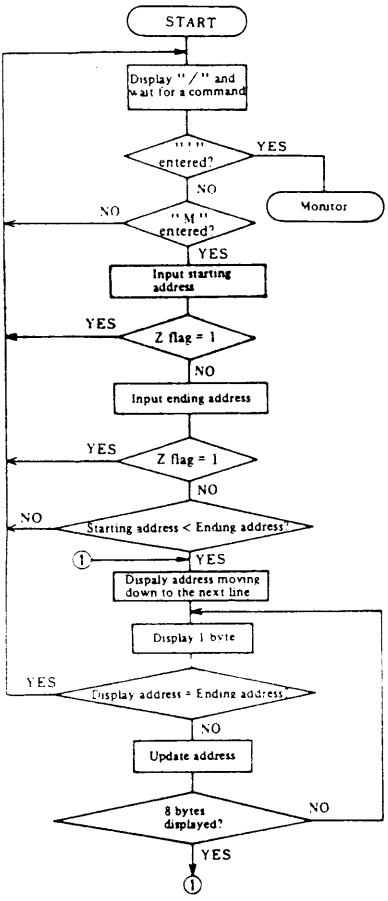
4.4 DISPLAYING A MEMORY BLOCK

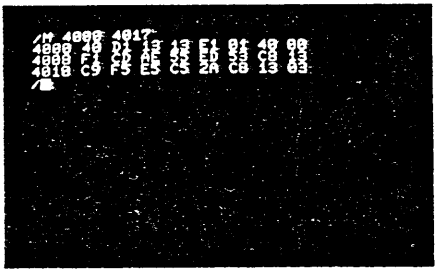
Let us construct a program which uses the hexadecimal data input and output subrpograms described above to display the contents of a specified memory block. The memory block must be specified in the same format as the M symbolic debugger command.

At the beginning of this command, the cursor is to blink to prompt for a command. There are to be two commands: M and !.

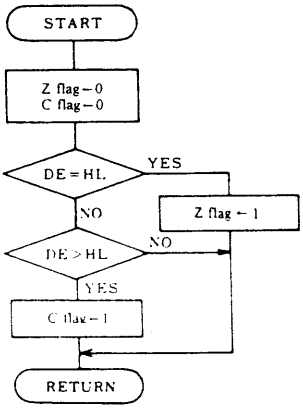
The memory dump is to be started with the M command and control is to be returned to the monitor by the ! command.

When the program is started with the M command, it is to wait for the starting address (a 4-digit hexadecimal number) at which the memory dump is to start. After the starting address is specified, the program is to wait for the ending address after printing a space. After the ending address is specified, the program is to start the memory dump, then return to the command wait state when the memory dump is completed.





Subroutine COMPR



```
0001: 0000                *           MACRO   SVC
0002: 0000                *           RST     3
0003: 0000                *           DEFB    @1
0004: 0000                *           ENDM
0005: 0000 (0003)           .CRT1C:   EQU     03H
0006: 0000 (0002)           .CR2:     EQU     02H
0007: 0000 (000D)           .INKEY:   EQU     0DH
0008: 0000 (003E)           BELL:     EQU     003EH
0009: 0000 (000C)           PRNTS:    EQU     000CH
0010: 0000 (0000)           MNTR:     EQU     0000H
0011: 0000                  ;
0012: 0000                  ; MEMORY DUMP
0013: 0000                  ;    ':GOTO MONITOR
0014: 0000                  ;
0015: 0000                  MEMRY:    ENT
0016: 0000 *                          SVC     .CR2
      0000 DF               *         RST     3
      0001 02               *         DEFB    .CR2
      0002                  *         ENDM
0017: 0002 3E2F                       LD      A,2FH
0018: 0004 *                          SVC     .CRT1C
      0004 DF               *         RST     3
      0005 03               *         DEFB    .CRT1C
      0006                  *         ENDM
0019: 0006 3E01                       LD      A,1
0020: 0008 *                          SVC     .INKEY
      0008 DF               *         RST     3
      0009 0D               *         DEFB    .INKEY
      000A                  *         ENDM
0021: 000A FE21                       CP      '!'
0022: 000C CA0000                     JP      Z,MNTR
0023: 000F FE4D                       CP      'M'
0024: 0011 2805                       JR      Z,+7
0025: 0013 CD3E00           MEMR0:    CALL    BELL
0026: 0016 18E8                       JR      MEMRY
0027: 0018 *                          SVC     .CRT1C
      0018 DF               *         RST     3
      0019 03               *         DEFB    .CRT1C
      001A                  *         ENDM
0028: 001A CD0000         E           CALL    PSG4K
0029: 001D 28F4                       JR      Z,MEMR0
0030: 001F EB                         EX      DE,HL
0031: 0020 CD0000         E           CALL    PSG4K
0032: 0023 28EE                       JR      Z,MEMR0
0033: 0025 CD4000                     CALL    COMPR
0034: 0028 38E9                       JR      C,MEMR0
0035: 002A EB                         EX      DE,HL
0036: 002B *                MEMR1:    SVC     .CR2
      002B DF               *         RST     3
      002C 02               *         DEFB    .CR2
      002D                  *         ENDM
0037: 002D CD0000         E           CALL    4HEX0
0038: 0030 0608                       LD      B,8
0039: 0032 7E               MEMR2:    LD      A,(HL)
0040: 0033 CD0000         E           CALL    PS2HX
0041: 0036 CD4000                     CALL    COMPR
0042: 0039 28D8                       JR      Z,MEMR0
0043: 003B 23                         INC     HL
0044: 003C 10F4                       DJNZ    MEMR2
0045: 003E 18EB                       JR      MEMR1
```

Displays " / " after moving to the next line and waits for a command.

Returns to monitor if " ! " is entered and starts the memory dump if " M " is entered.

Returns to the command wait state if an invalid command is entered.

Input starting and ending addresses of the memory block to be dumped.

Returns to the command wait state if the starting address is greater than the ending address.

Displays address after moving down to the next line.

Displays the memory dump (8 bytes on a line) until the ending address is reached.

```
0046: 0040                ;
0047: 0040                ; COMPARE DE,HL(DESTROYED:A)
0048: 0040                ;   CALL COMPR
0049: 0040                ;     EXIT:DE=HL  Z=1
0050: 0040                ;          DE>HL  C=1
0051: 0040                ;
0052: 0040                COMPR:  ENT
0053: 0040 7C                     LD      A,H
0054: 0041 92                     SUB     D
0055: 0042 C0                     RET     NZ
0056: 0043 7D                     LD      A,L
0057: 0044 93                     SUB     E
0058: 0045 C9                     RET
0059: 0046                        END
```

Returns with Z flag set if DE = HL, and with C flag set if DE > HL.

4.5 WRITING DATA INTO A MEMORY AREA

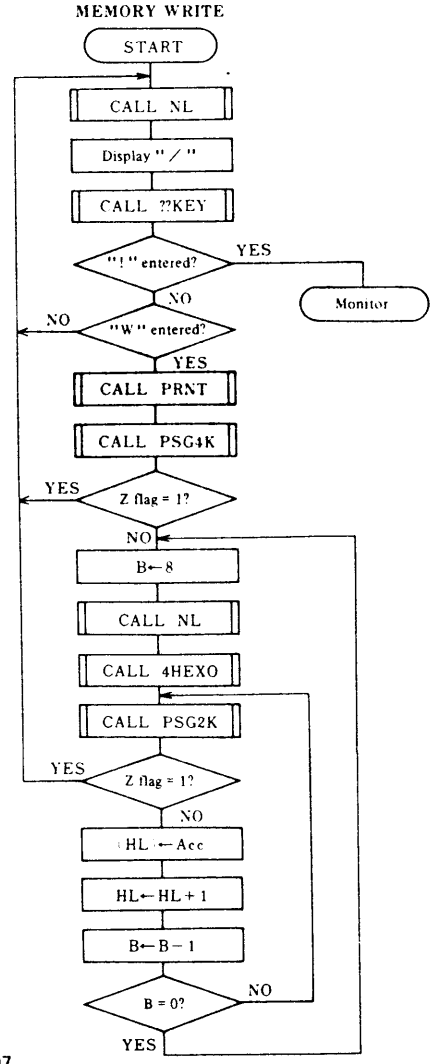
Let us construct a program to write 2-digit hexadecimal numbers into a memory block, starting at a specified address. The memory block is to be specified in the same format as with the W command.

At the beginning of this program, the program is to flash the cursor while waiting for command entry. Memory write is to be started with the W command and control is to be returned to the monitor with the ! command.

When the program is started with the W command, it is to prompt for the starting address (a 4-digit hexadecimal number) of the memory block at which the memory write is to start. After the starting address is specified, the program is to display the specified address on a new line and wait for the operator to enter 2-digit hexadecimal numbers.

Data entry is to be terminated with [CR]

```
/W A000
A000 14 56 78 EF 15 20 01 56
A008 01 AF 31 ■
```



```
** SHARP Z80 ASSEMBLER  1Z-018A V1.0A   PAGE 01

0001: 0000                ; MEMORY WRITE
0002: 0000                ;   W:START
0003: 0000                ;   !:GOTO MONITOR
0004: 0000                ;
0005: 0000              *         MACRO   SVC
0006: 0000              *         RST     3
0007: 0000              *         DEFB    @1
0008: 0000              *         ENDM
0009: 0000 (0003)           .CRT1C: EQU     03H
0010: 0000 (0002)           .CR2:   EQU     02H
0011: 0000 (000D)           .INKEY: EQU     0DH
0012: 0000 (003E)           BELL:   EQU     003EH
0013: 0000 (000C)           PRNTS:  EQU     000CH
0014: 0000 (C000)           MNTR:   EQU     0000H
0015: 0000                  WRITE:  ENT
0016: 0000 *                        SVC     .GR2
      0000 DF             *         RST     3
      0001 02             *         DEFB    .CR2
      0002                *         ENDM
0017: 0002 3E2F                     LD      A,2FH
0018: 0004 *                        SVC     .CRT1C
      0004 DF             *         RST     3
      0005 03             *         DEFB    .CRT1C
      0006                *         ENDM
0019: 0006 3E01                     LD      A,1
0020: 0008 *                        SVC     .INKEY
      0008 DF             *         RST     3
      0009 0D             *         DEFB    .INKEY
      000A                *         ENDM
0021: 000A FE21                     CP      '!'
0022: 000C CA0000                   JP      Z,MNTR
0023: 000F FE57                     CP      'W'
0024: 0011 2805                     JR      Z,+7
0025: 0013 CD3E00           WRITE0: CALL    BELL
0026: 0016 18E8                     JR      WRITE
0027: 0018 *                        SVC     .CRT1C
      0018 DF             *         RST     3
      0019 03             *         DEFB    .CRT1C
      001A                *         ENDM
0028: 001A CD0000         E         CALL    PSG4K
0029: 001D 28F4                     JR      Z,WRITE0
0030: 001F 0608             WRITE1: LD      B,8
0031: 0021 *                        SVC     .CR2
      0021 DF             *         RST     3
      0022 02             *         DEFB    .CR2
      0023                *         ENDM
0032: 0023 CD0000         E         CALL    4HEX0
0033: 0026 CD0000         E WRITE2: CALL    PSG2K
0034: 0029 28E8                     JR      Z,WRITE0
0035: 002B 77                       LD      (HL),A
0036: 002C 23                       INC     HL
0037: 002D 10F7                     DJNZ    WRITE2
0038: 002F 18EE                     JR      WRITE1
0039: 0031                          END
```

Displays " / " on a new line and waits for command entry.

Returns to the monitor if "!" is entered and starts the memory write.

Returns to the beginning of program and sounds a beeper tone if an invalid key is pressed.

Input the starting address at which memory write is to start.

Input 8 bytes on a line and continues the memory write while displaying write addresses.
Data entry is terminated with [CR] .

This program references the following monitor and external subroutines:

BELL	003H	Monitor subroutine	
.CR2	02H	Monitor calls	
.CRTIC	03H		
.INKEY	0DH		
4HEXO	(4hexa data out)		See section 4.2.
PSG2K	(print space, get 2hexa data)		See section 4.3.
PSG4K	(Print space, get 4hexa data)		See section 4.3.

[Application]

Construct a machine-language monitor program which executes the W, M, and ! commands described above, as well as additional execution command G. The G command must input the starting address using the GET4K subroutine and execute the program (by loading the starting address into the program counter (PC)).

/W	X X X X		memory write	
/M	X X X X	Y Y Y Y	memory dump	
/G	X X X X		goto XXXX	. . . . . Load XXXX into program counter.
/!			goto monitor	. . . . . Jump to address 0000H.

APPENDICES

1. MONITOR SUBROUTINES

— MZ-800 monitor subroutines —

The following subroutines are used by the ROM Monitor (9Z-504M). Each subroutine name symbolically represents the function of the corresponding subroutine. These subroutines can be called from user programs.
Registers saved are those whose contents are restored when control is returned to the calling program. The contents of other registers are changed by execution of the subroutine.

Name and entry point (hex.)	Function	Registers saved
CALL LETNL (0006)	Moves the cursor to the beginning of the next line.	All except AF
CALL PRNTS (000C)	Displays a space at the cursor position.	Al lexcept AF
CALL PRNTS (0012)	Displays the character corresponding to the ASCII code stored in the ACC at the cursor position. See Appendix J for the ASCII codes. No character is displayed when code 0D (carriage return) or codes 11 to 16 (the cursor control codes) are entered, but the corresponding function is still performed (a carriage return for 0D and cursor movement for 11 to 16).	All except AF
CALL MSG (0015)	Displays a message, starting at the position of the cursor. The starting address of the area in which the message is stored must be loaded into the DE register before calling this subroutine, and the message must end with a carriage return code (0D). The carriage return is not executed. The cursor is moved if any cursor control codes (11 to 16) are included in the message.	All registers
CALL BELL (003E)	Briefly sounds tone of la (about 880 Hz).	All except AF
CALL MELDY (0030)	Plays a tune according to the music data stored in the memory area starting at the address in the DE register. The music data must be in the same format as that for the MUSIC statement of the BASIC, and must end with 0D or C8. When the tune is completed, control is returned to the calling program with the C flag set to 0. When play is interrupted with the [BREAK] key, Control is returned with the C flag set to 1.	All except AF
CALL XTEMP (0041)	Sets the music tempo according to the tempo data stored in the accumulator (ACC). ACC ← 01 Slowest speed ACC ← 04 Middle speed ACC ← 07 Highest speed Note that the data in the accumulator is not the ASCII codes for 1 to 7 but the binary codes.	All registers
CALL MSTA (0044)	Generates a continuous sound of the specified frequency. The frequency is given by the following equation freq. = 895 kHz / nn'. Here, nn' is a 2-byte number stored in addresses 11A1 and 11A2 (n in 11A2 and n' in 11A1)	BC and DE

Name and entry point (hex.)	Function	Registers saved
CALL MSTP (0047)	Stops the sound generated with the CALL MSTA subroutine.	All except AF
CALL TIMST (0033)	Sets and starts the built-in clock. The registers must be set as follows before this routine is called. ACC ← 0 (AM), ACC ← 1 (PM) DE ← 4-digit hexadecimal number representing the time in seconds.	All except AF
CALL TIMRD (003B)	Reads the built-in clock and returns the time as follows. ACC ← 0 (AM), ACC ← 1 (PM) DE ← 4-digit hexadecimal number representing the time in seconds.	All except AF and DE
CALL BRKEY (001E)	Checks whether the [SHIFT] and [BREAK] keys are both being pressed. The Z flag is set when they are being pressed simultaneously; otherwise, it is reset.	All except AF
CALL GETL (0003)	Reads one line of data from the keyboard and stores it in the memory area starting at the address in the DE register. This routine stops reading data when the [CR] key is pressed, then appends a carraige return code (0D) to the end of the data read. A maximum of 80 characters (including the carriage return code) can be enered in one line. Characters keyed in are echoed back to the display. Cursor control codes can be included in the line. When the [SHIFT] and [BREAK] keys are pressed simultaneously, the BREAK code is stored at the address indicated by the DE register and a carriage return code is sotored in the following address.	All registers
CALL GETKY (001B)	Reads a character code (ASCII) from the keyboard. If no key is pressed, control is returned to the calling program with 00 set in ACC. No provision is made to avoid data read errors due to key bounce, and characters entered are not echoed back to the display. When any of the special keys (such as [DEL] or [CR]) are pressed, this subroutine returns a code to the ACC which is different to the corresponding ASCII code as shown below. Here, display codes are used to address characters stored in the character generator, and are different from the ASCII codes.	All except AF

Special key read with GETKY

Special key	Code loaded in ACC	Display code
DEL	60	C7
INST	61	C8
ALPHA	62	C9
BREAK	64	CB
CR	66	CD
↓	11	C1
↑	12	C2
→	13	C3
←	14	C4
HOME	15	C5
CLR	16	C6

Name and entry point (hex.)	Function	Registers saved
CALL ASC (03DA)	Loads the ASCII character corresponding to the hexadecimal number represented by the lower 4 bits of data in ACC.	All except AF
CALL HEX (03F9)	Converts the 8 data bits stored in the ACC into a hexadecimla number (assuming that the data is an ASCII character), then loads the hexadecimal number in the lower 4 bits of ACC. The C flag is set to 0 when a hexadecimal number is loaded in ACC; otherwise, it is set to 1.	All except AF
CALL HLHEX (0410)	Converts a string of 4 ASCII characters into a hexadecimal number and loads it in the HL register. The call and return conditions are as follows. DE ← Starting address of the memory area which contains the ASCII character string. (e.g., "3" "1" "A" "5") CALL HLHEX ←DE CF = 0 HL←hexadecimal number (e.g., HL = $31A5_H$) CF = 1 The contents of HL are not guarenteed.	All except AF and HL
CALL 2HEX (041F)	Converts a string of 2 ASCII characters into a hexadecimal number and loads it into the ACC. The call and return conditions are as follows. DE ← Starting address of the memory area which contains the ASCII character string. (e.g., "3" "A") CALL 2HEX ←DE CF = 0 ACC←hexadecimal number (e.g., ACC = $3A_H$) CF = 1 The contents of the ACC are not guaranteed.	All except AF and DE
CALL ??KEY (09B3)	Blinks the cursor to prompt for key input. When a key is pressed, the corresponding display code is loaded into the ACC and control is returned to the calling program.	All except AF
CALL ?ADCN (0BB9)	Converts ASCII codes into display codes. The call and return conditions are as follows. ACC ← ASCII code CALL ?ADCN ACC ← Display code	All except AF
CALL ?DACN (0BCE)	Converts display codes into ASCII codes. The call and return conditions are as follows. ACC ← Display codes CALL ?DACN ACC ← ASCII code	All except AF
CALL ?BLNK (0DA6)	Detects the vertical blanking period. Control is returned to the calling program when the vertical blanking period is entered.	All registers
CALL ?DPCT (0DDC)	Controls display as follows. ACC: Control — $C0_H$ Scrolling; $C1_H$ Same as the ↓ key.; $C2_H$ Same as the ↑ key.; $C3_H$ Same as the → key.; $C4_H$ Same as the ← key.; $C5_H$ Same as the HOME key.; $C6_H$ Same as the CLR key.; $C7_H$ Same as the DEL key.; $C8_H$ Same as the INST key.; $C9_H$ Same as the ALPHA key.; CD_H Same as the CR key.	All registers
CALL ?POINT (0FB1)	Loads the current cursor location into the HL register. The return conditions are as follows. CALL ?POINT HL ← Cursor location (binary)	All except AF and HL

— MZ-800 monitor call —

Functions of this monitor can be called using their function numbers in the same manner as function calls.

In the following explanation, two-digit hexadecimal numbers printed in Gothic are the function numbers and the characters at their right are the function names.

The table below lists the main monitor variables related to the monitor calls.

Monitor variable	Address in hexadecimal	Length in bytes	Function
SYSSTA	004D	2	Hot start address of the utilities using this monitor.
ERRORP	004F	2	Address of the error handling routine of the utilities using this monitor
ELMD	1000	1	File mode 1: Object file 2: BASIC text file 3: Source file 4: Relocatable file
ELMD1	1001	17	File name (up to 16 characters) and end mark 0DH.
ELMD20	1014	2	File size in bytes
ELMD22	1016	2	Load address
ELMD24	1018	2	Excrution address
ZLOG	1042	1	Logical number
ZRWX	1043	1	File open type 1: Read open 2: Write open
TEXTST	1070	2	Starting address of the text area of utilities using this monitor.
POOL	1072	2	Starting address of the work area of this monitor.
VARST	1074	2	Starting address of the variable area of utilities using this monitor.
TMPEND	107A	2	Ending address of the temporary area of utilities using this monitor.
TEMLMT	107E	2	Ending address of the memory area used by this monitor.
FILOUT	1091	1	Data is output to the CRT if the value at this address is zero and to the printer if it is 1. (The device specification is effective for monitor functions .&CR, .&lC, .&lCX and .&MSG)

00 .MONOP

Function: Returns to the RAM monitor.
Input registers: None
Output registers: None
Registers saved: None

01 .CR1

Function: Starts a new line independent of cursor position on a line.
Input registers: None
Output registers: None
Registers saved: Primary registers only

02 .CR2

Function:	Starts a new line if the cursor is not at the beginning of a line.
Input registers:	None
Output registers:	None
Registers saved:	Primary register pairs only

03 .CRT1C

Function:	Outputs a character to the CRT. Control codes are executed.
Input register:	ACC: = Output data
Output registers:	None
Registers saved:	Primary register pairs only

04 .CRT1X

Function:	Outputs a character to the CRT. Control codes are displayed in reverse video.
Input register:	ACC: = Output data
Output registers:	None
Registers saved:	Primary register pairs only

05 .CRTMS

Function:	Outputs a character string. The end code is 00H. Control codes are executed. (Same as .CRT1C)
Input register:	DE: = Pointer position of the character string
Output registers:	None
Registers saved:	Primary register pairs only

06 .LPT0T

Function:	Outputs a character to the printer without code conversion.
Input register:	ACC: = Output data
Output registers:	None
Registers saved:	Primary register pairs only

07 .LPT1C

Function:	Outputs a character to the printer converting its code into that of the printer used with the MZ computer. The PRINT/P statement of BASIC uses this function.
Input register:	ACC: = Output data
Output registers:	None
Registers saved:	Primary register pairs only

08 .&CR

Function: Starts a new line on the CRT or printer according to the value of variable FILOUT (address 1091). Set the value of variable FILOUT to 1 to select the CRT or to 0 to select the printer in advance.

Input registers: None

Output registers: None

Registers saved: Primary register pairs only

09 .&1C

Function: Outputs a character to the CRT or printer according to the value of variable FILOUT. Control codes are executed when they are output to the CRT (same as .CRT1C). When this function outputs a character to the printer, its function is the same as .LPR1C.

Input register: ACC: = Output data

Output registers: None

Registers saved: Primary register pairs only

0A .&1CX

Function: Outputs a character to the CRT or printer according to the value of variable FILOUT. Control codes output to the CRT are displayed in reverse video (same as CRT1X). When the character is output to the printer, it is output in the same manner as with the PINT/P statement of BASIC.

Input register: ACC: = Output data

Output registers: None

Registers saved: Primary register pairs only

0B .&MSG

Function: Outputs a character string to the CRT or printer. The end code is 00H. The switching condition between the CRT and printer is the same as that of the .&CRT. This function executes control codes when it outputs a character string to the CRT (same as .&CRT1C). When it outputs a character string to the printer, its function is the same as .LPT1C.

Input register: DE: = Pointer position of the character string

Output registers: None

Registers saved: Primary register pairs only

0C .GETL

Function: Inputs one line of data from the keyboard and adds an end code 00H to the end of the data.

Input register: DE: = Starting address of the buffer in which the input data is stored.

Output register: CF: = 1 when [SHIFT] + [BREAK] are depressed

Registers saved: Primary register pairs except AF

0D .INKEY

Function:	Inputs a character from the keyboard.
Input registers:	A: = 0 Real time key scan (same as the GET statement of BASIC) A: = 1 Waits for key input blinking the cursor. A: = FFH Unlike when A: = 0, inputs only once if a key is depressed and held.
Output register:	A: = MZ ASCII code
Registers saved:	Primary register pairs except AF

0E .BREAK

Function:	Detects [SHIFT] + [BREAK].
Input register:	None
Output registers:	ZF: = 1 when [SHIFT] + [BREAK] are pressed.
Registers saved:	Primary register pairs except AF.

0F .HALT

Function:	Waits for [SPACE] to be subsequently depressed if it is pressed. If [SHIFT] + [BREAK] are pressed next, this function transfers control to the address identified by ERRORP..
Input registers:	None
Output registers:	None
Registers saved:	Primary register pairs except AF

10 .DI

Function:	Stops spooling or music, and inhibits interrupt.
Input registers:	None
Output registers:	None
Registers saved:	Primary register Pairs except AF

11 .EI

Function:	Starts spooling or enables interrupt.
Input registers:	None
Output registers:	None
Registers saved:	Primary register pairs except AF

17 .COUNT

Function:	Counts the number of characters of the specified character string. The string must end with an end code 00H.
Input register:	DE: = Pointer position of the chararter string
Output registers:	ACC: = The length of the string
Registers saved:	Primary register pairs except AF

1B .ERRX

Function: Displays an error message.

Input register: ACC: = Error code (the same as the error number listed in the error message table of the OWNER'S MANUAL). When the value of the 7-th bit of the accumulator is 1, the related device name is also displayed.

Output registers: None

Registers saved: Primary register pairs only

2C .DEVNM

Function: Interprets (specifies) the device name.

Input registers: DE: = Pointer position of the chararter string whirh indicates the device name.

B: = The length of the string

Output registers: HL: = Pointer position next to the end of the device name which has been interpreted.

DE: = The starting address of the device table

ACC: = Device identification number (unit number)

Registers saved: None

2D .DEVFN

Function: Interprets the device name and file name.

Input register: DE: = Pointer position in the device name and file name string.

Output registers: None

Registers saved: None

2E .LUCHK

Function: Checks whether the logical number is defined.

Input register: ACC: = Logical number

Output registers: ACC: = 1 (read opened)

ACC: = 2 (write opened)

ACC: = 3 (read/write opened)

CF: = 1 (not opened)

Registers saved: Primary registers except AF

2F .LOPEN

Function: Opens files which are not divided into blocks as object files. To execute this function, the device name and filename must be specified with function .DEVFN in advance.

Input registers: None

Output registers: None

Registers saved: None

30 .LOADF

Function: Loads files which are not divided into blocks such as object files. To execute this function, the file must have been opened with functions .DEVFN and .LOPEN in advance.

Input registers: HL: = Loading address

Output registers: None

Registers saved: None

31 .SAVEF

Function: Saves files which are not divided into blocks such as object files. To execute this function, the file name must be specified with function .DEVFN.

Input registers: DE: = Starting address of the memory area to be saved
ELMD20(1014H): = File size in bytes
ELMD22(1916H): = Load address
ELMD24(1018H): = Execution address

Output registers: None

Registers saved: None

32 .VRFYF

Function: Compare the contents of the specified memory area with a file which are not divided into blocks such as object files. To executes this function, the file must be opened with .DEVFN and .LOPEN in advance.

Input registers: None

Output registers: None

Registers saved: None

33 .RWOPN

Function: Opens to read or write a file which are divided into blocks such as source files (files in ASCII code). To execute this function, the device name and file name must be specified with function .DEVFN in advance.
ZRWX (1043H): = 1 for read-open
ZRWX: = 2 for write-open

Input registers: None

Output registers: None

Registers saved: Primary register pairs except AF

35 .INMSG

Function: Inputs one line of data of the file opened which has been opened with function call .RWOPN.

Input registers: DE: = Starting address of the input buffer

Output register: B: = Input file size in bytes
CF: = 1 when the file end (EOF) has been detected.

Registers saved: DE, HL

37 .PRSTR

Function: Writes the specified bytes of data (max. 255 characters) into the file which has been write opened with .RWOPN.

Input registers: DE: = Starting address of the data to be written.
B: = Data size in bytes

Output registers: None

Registers saved: Primary register pairs except AF

38 .CLKL

Function: Closes or kills the files opened.

Input registers: ACC: = Logical number of the file to be closed or killed
(When ACC: = 0, all files opened are closed or killed.)
B: = 0 for kill and B< >0 for close

Output registers: None

Registers saved: Primary register pairs except AF

39 .DIR

Function: Displays or prints out the information concerning files stored on the disk or the contents of the directory. The device name must be specified with monitor call .DEVNM in advance.

Input registers: ACC: = 0 Inputs the contents of directory into the directory buffer in the monitor.
ACC< >0 Outputs the directory in the directory buffer to the device specified with the value in the ACC.
ACC: = 88H To the CRT
ACC: = 89H To the printer
Otherwise, the directory is output to the file or device specified by the logical number set.

Output registers: None

Registers saved: Primary register pairs except AF

3A .SETDF

Function: Sets the default device.

Input registers: DE: = Starting address of the device table
ACC: = Device identification number (unit number)
These are output registers set by .DEVNM.

Output registers: None

Registers save: Primary register pairs only

3C .FINIT

Function: Initializes the I/O handler routine in the monitor (this function is used by the INIT statement of BASIC). The device name must be specified with monitor call .DEVNM in advance.

Input registers: None

Output registers: None

Registers save: None

43 .ERCVR

Function: Performs the error recovery operation and stops the motor of the MZ disk or floppy disk.

Input registers: None

Output registers: None

Registers saved: None

— Examples of use of monitor calls —

In following examples, it is assumed that the SVC macro has been defined.

```
MACRO SVC
RST 3
DEFB @1        → A function number is assigned to parameter @1.
ENDM
```

When you create programs using this monitor, please add the program below to those programs at the top of them.

```
LD HL, ERADR          ; Sets the address of the error handling routine of the program.
LD (ERRORP), HL
LD HL, hot-start      ; Sets the hot start address of the program.
LD HL, last           ; Sets the last address of the program.
LD (TEXTST), HL
LD (POOL), HL
LD (HL), 0
INC (HL)
INC HL
LD (VARST), HL
LD (TMPEND), HL
LD DE, 6000           ; Sets 600H for a floppy disk and 800H for the MZ disk.
ADD HL, DE
LD (MEMLMT), HL
LD SP, HL             ; Sets HL to the initial value of the stack pointer.
```

Creates error handling routine ERADR mentioned above as follows

```
ERADR: OR   A
       JR   Z, break-adr ; Jumps to [SHIFT] + [BREAK] handling routine.
       CP   80H
       JR   Z, break-adr
       SVC .ERR          ; Displays an error message.
```

Loading or verifying an object file

```
LD DE, FILE, X          ; Set name of object file to be loaded in DE.
SVC .COUNT              ; Set length of file name in B and returns.
SVC .DEVFN              ; Interpret (specify) device name and file name.
SVC .LOPEN              ; Open the file.
LD A, (ELDM)            ; Set file mode of file opened in ACC.
CP 1                    ; Object file?
JP NZ, error            ; Jump to error handling routine if not object file.
LD HL, (ELMD22)         ; Set load address in HL.
SVC .LOADF (or SVC .VRFYF)  ; Load or verify the file.
    :
FILE. X: DEFM "QD: SAMPLE"
         DEFB 0         ; Set the end of the file name to 0.
```

Saving an object file

```
LD DE, FILE. X          ; Set the name of the file to be saved in DE.
SVC .COUNT              ; Set the length of the file name in B and returns.
SVC .DEVFN              ; Interpret (specify) the device name and file name.
LD A, 1
LD (ELMD),A             ; Set the file mode to the object file mode.
LD HL, length
LD (ELMD20), HL         ; Set the file length in bytes.
LD HL, loading-adr
LD (ELMD22), HL         ; Set the load address of the file.
LD HL, exer-adr
LD (ELMD24), HL         ; Set the execution address of the file.
LD DE, save-adr
SVC .SAVEF              ; Save the file.
    :
FILE. X: DEFM "QD:SAMPLE"
         DEFB 0         ; Add 0 to end of filename.
```

Opening a source file (ASCII file)

```
LD A, open-mode          ; Read-open if the open mode is 1 and write-opens if it is 2.
LD (ZRWX), A
LD A, 3
LD (ELMD), A             ; Set the file mode to the source file mode.
LD A, 1
LD (ZLOG), A             ; Set the logical number to 1.
LD DE, FILE. X           ; Set the name of the file to be opened in DE.
SVC .COUNT               ; Sets the file length in bytes in B.
SVC .DEVFN               ; Interpret (specify) the device name and file name.
SVC .RWOPN               ; Open the file.
LD A, (ELMD)             ; Set the file mode opened in A.
CP 3                     ; Source file?
JP NZ, error             ; Jump to the error handling routine if not source file.
  :
FILE. X : DEFM "QD: SAMPLE"
          DEFB 0         ; Set the end of the file name to 0.
                           Add 0 to end of filename.
```

Inputting one line of source file (ASCII file)

```
  :                      ; The source file is assumed to be read-opened.
LD DE, buffer-adr        ; Set the starting address of the input buffer in DE.
SVC .INMSG               ; Input one line.
JP C, eof                ; Perform the file end processing if CF: = .
  :                      ; Set the length of the line read in bytes in B.
                           (0D at the end of the line is not included.)
```

Output of a source file (ASCII file)

```
  :                      ; The source file is assumed to have been opened.
LD DE, save-adr          ; Starting address of the memory area to be saved.
LD B, length             ; File size in bytes (including 0D at the end of each line).
SVC .PRSTR
  :
```

Closing source files (ASCII files)

```
LD A, logiral-number     ; Specified file only if the logical number is not zero and all files if it
                           is zero.
LD B, FFH
SVC .CLKL                ; Closes the file(s).
```

Killing source file

```
LD A, logical-number      ; Specified file only if the logical number is not zero and all files if it
                            is zero.
LD B, 0
SVC .CLKL                 ; Kill the file(s).
```

Setting a default device

```
LD DE, device-name        ; Set the pointer position in the default device name in DE.
?SVC .COUNT               ; Sets the length of the device name in B and returns.
SVC .DEVNM                ; Interpret (specify) the device name.
SVC .SETDF                ; Set the device specified by the value in ACC to the current device.
```

Display or Print out of directory

```
LD DE, device name        ; Set the pointer position of the device name whose directory is to
                            be displayed or printed out in DE.
SVC .COUNT                ; Set the length of the device name in B and returns.
SVC .DEVNM                ; Interpret (specify) the device name.
LD B, A
XOR A                     ; Read the directory into the directory buffer in the monitor.
SVC .DIR
LD A, B
LD A, 88H (or 89H)
SVC .DIR                  ; Output the directory to the CRT if the contents of ACC is 88H or
                            to the printer if it is 89H.
```

Initialization of device (used by INIT statement of BASIC)

```
LD DE, device-name        ; Set the pointer position for the device name to be initialized in DE.
SVC .COUNT                ; Set the length of the device name in B and returns.
SVC .DEVNM                ; Interpret (specify) the device name.
SVC .FINIT                ; Initialize the device.
```

2. MAKING BACKUP COPY OF THE MZ-800 SYSTEM PROGRAM

It is possible that you may accidentally damage the tape which contains the SYSTEM PROGRAM. When this happens, you cannot use the computer. To avoid this, make a copy of the SYSTEM PROGRAM. After making the backup copy, store the original tape in a safe place and use the backup copy for daily use.

Backup procedures are as follows.

1) Prepare a new cassette tape.
2) Turn on the MZ-800 and press the [M] key to start the monitor.
3) Load the tape which contains the SYSTEM PROGRAM into the data recorder, then enter the following monitor command.
   * GE807 [CR]
4) When "↓ PLAY" is displayed, press the [PLAY] button of the data recorder to read the SYSTEM PROGRAM into memory.
5) When the prompt (*) appears, replace the tape with the new one and enter the following monitor command.
   *GE80A [CR]
6) When "↓ RECORD.PLAY" is displayed, press the [RECORD] button of the data recorder to write the SYSTEM PROGRAM to the new tape.
7) When the prompt (*) appears, rewind the tape. Then, enter the following monitor command.
   * GE80D [CR]
8) When "↓ PLAY" is displayed, press the [PLAY] button to verify the tape contents.
9) When the message "CHECK SUM ER." is displayed, repeat steps 3 to 8.
   When the message "OK." is displayed, copying is completed.

COPYING OF MZ-700 SYSTEM PROGRAM

Please follow the procedure below mentioned to copy the SYSTEM PROGRAM tape.

1) Power on MZ-700 (→monitor state)
2) Partial memory should be modified by the use of monitor command M (memory correction) as follows:

```
*MCF00
CF00   CD      CF16   CD
CF01   27      CF17   21
CF02   00      CF18   00
CF03   38      CF19   38
CF04   03      CF1A   03
CF05   CD      CF1B   CD
CF06   2A      CF1C   24
CF07   00      CF1D   00
CF08   DA      CF1E   C3
CF09   FE      CF1F   08
CF0A   00      CF20   CF
CF0B   C3
CF0C   AD
CF0D   00
CF0E   CD
CF0F   27
CF10   00
CF11   38
CF12   F5
CF13   C3
CF14   CB
CF15   0F
```

[SHIFT] + [BREAK] to be keyed in.

NOTE: The content of memory from CF00 to CF15 may not always be as above mentioned.

3) The cassette to be read (copyed from) should be set to the tape recorder.
4) Key in the monitor command J (Jump) As follows:

```
* JCF00 [CR]
↓ PLAY
```

NOTE: If a button of the tape recorder is still pushed on play indication will appear.

5) Confirming the "↓ PLAY" indication above mentioned, push PLAY button and load the content of SYSTEM PROGRAM tape. On this occasion, no indication like FILE NAME, etc. will be shown. When ERROR occured, please restart from the item 1) again.
6) Set a new cassette to which the SYSTEM PROGRAM should be written into the recorder and execute [REWIND] .

7) Key in as follows:

*JCF16 [CR]

8) ↓ RECORD. PLAY
[STOP] button should be pushed beforehand.

9) Push [RECORD] button. The copy will start and the following indication will appear:

WRITING MZ-1Z018C

On the occasion of MZ-711, item 9) should be effectuated after setting the external tape recorder in recording state.

10) After the sound "Pit Pit", the copy will be terminated.
11) The monitor state will be recovered by pushing the rear RESET SW.
12) Rewind the tape and push [STOP] button.
13) Key in as follows:

*JCF0E [CR]
↓PLAY

14) Push [PLAY] button of the recorder and the "VERIFY" function will be executed. When successful verified, the indication of "OK!" will appear though no other indication like FILE NAME etc. will appear. When error occured, please restart from the item 4).
15) Please make sure to enable the write protection of the cassette by removing the nail.

3. CODE TABLES

— ASCII code table —

MSD is an abbreviation for most significant digit, and represents the upper 4 bits of each code. LSD is an abbreviation for least significant digit, and represents the lower 4 bits of each code. Codes 11_H to 16_H are cursor control codes. For example, executing CALL PRINT (a monitor subrountine) with 15_H loaded into the ACC returns the cursor to the home position. ("H" is not displayed.)

LSD \ MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000		SP	0	@	P					q	n				
1	0001		!	1	A	Q					a					
2	0010		"	2	B	R					e	z				
3	0011		#	3	C	S						w	m			
4	0100		$	4	D	T					~	s				
5	0101		%	5	E	U						u				
6	0110		&	6	F	V	¥				t	i	→			
7	0111		'	7	G	W					g	o				
8	1000		(	8	H	X					h	o				
9	0001		)	9	I	Y					k	A				
A	1010		*	:	J	Z					b	f	o			
B	1011		+	;	K	[		°			x	v	a			£
C	1100		,	<	L	\					d					
D	1101	CR	−	=	M	]					r	u	y			
E	1110		.	>	N	↑					p	β				
F	1111		/	?	O	←					c	j				π

— Display code table —

The display codes are used to address character patterns stored in the character generator. These codes must be transferred to video-RAM to display characters.

Monitor subroutines PRNT (0012_H) and MSG (0015_H) convert ASCII codes into display codes and transfer them to the V-RAM location indicated for the cursor.

Codes $C1_H$ to $C6_H$ are for controlling the cursor.