Appendix A: Example Programs

This appendix contains a number of programs that illustrate the use of BDOS functions. Most of them are self explanatory but the following notes describe some of their features.

Reaction timer program

This program uses routines described in chapter 4. It waits for you to press the space bar and then checks how long it takes you to press another key.

Get Console Status is used to break out of the timer loop when the second key has been pressed.

File size calculation program

The function FSIZE calculates the size of a named file and can be used either with the ProPascal program in chapter 9 or in your own assembler programs.

Its input parameter is the address of an FCB and this is passed on the stack. FSIZE will return to you in the register pair HL either the size of the file in sectors or −1 (if the file does not exist).

Disc free space program

The Pascal program provided uses the assembler procedures GET_DPB and GET_MAP and prints the amount of free space on a specified disc. GET_DPB and GET_MAP have been written so as to be of use in your own assembler programs and are suitably annotated.

GET_DPB returns a record containing the DPB for the specified drive. GET_MAP returns the allocation map for this drive. Both procedures expect two parameters: the drive code and an address of memory into which they will place the information required.

```
SIMPLE REACTION TIMER          RML Z80 Ass V 4.1 k    20-Mar-84    Page 1


                0001 *H SIMPLE REACTION TIMER
                0002 COM
                0003 *LM OFF
                0004
0005 =          0005 BDOSE                  EQU     5
0100 =          0006 TPA                    EQU     100H
                0007
0009 =          0008 FN_PRINT_STRING        EQU     9
0001 =          0009 FN_CONSOLE_IN          EQU     1
000B =          0010 FN_CONSOLE_STATUS      EQU     11
0000 =          0011 FN_SYSTEM_RESET        EQU     0
                0012
0024 =          0013 END_OF_STRING          EQU     '$'
000D =          0014 CR                     EQU     0DH
000A =          0015 LF                     EQU     0AH
0020 =          0016 SPACE                  EQU     20H
0001 =          0017 KEY_PRESSED            EQU     1
                0018
0000 =          0019 DELAY_2_MILLISECONDS   EQU     0.
                0020
                0021 MACRO   BDOS    P1
                0022                 LD      C,P1
                0023                 CALL    BDOSE
                0024 ENDM
                0025
                0026
0100            0027         ORG     TPA
0100            0028 START:
0100 31F502     0029         LD      SP,MY_TOP_STACK
                0030
0103 117C01     0031         LD      DE,SIGNON_MSG
                0032         BDOS    FN_PRINT_STRING
                0036
010B            0037 WAIT_FOR_START_KEY:
                0038         BDOS    FN_CONSOLE_IN                  ; WAIT FOR START KEY
0110 FE20       0042         CP      SPACE                          ; IS IT <SPACE> ?
0112 20F7       0043         JR      NZ,WAIT_FOR_START_KEY
                0044
0114 11B901     0045         LD      DE,STOP_MSG
                0046         BDOS    FN_PRINT_STRING
                0050
011C 3E00       0051         LD      A,0                            ; CLEAR TIMER COUNT
011E 32F401     0052         LD      (TICKS),A
0121 32F301     0053         LD      (TOCKS),A
                0054
0124            0055 TIMER_LOOP:
                0056         BDOS    FN_CONSOLE_STATUS              ; TEST IF A KEY HAS BE
0129 FE01       0060         CP      KEY_PRESSED                                   PRESS
012B 281A       0061         JR      Z,STOP_THE_CLOCK
                0062
012D            0063 SHORT_DELAY:
012D 0600       0064         LD      B,DELAY_2_MILLISECONDS         ; SHORT DELAY LOOP
                0065 1$:
012F E3         0066         EX      (SP),HL                        ; WASTE TIME
0130 E3         0067         EX      (SP),HL
```

```
SIMPLE REACTION TIMER          RML Z80 Ass V 4.1 k    20-Mar-84    Page 2


 0131 10FC     0068          DJNZ    1$
               0069
 0133 3AF401   0070          LD      A,(TICKS)                    ; INCREMENT THE TIME
 0136 C601     0071          ADD     A,1
 0138 27       0072          DAA
 0139 32F401   0073          LD      (TICKS),A
 013C 3AF301   0074          LD      A,(TOCKS)
 013F CE00     0075          ADC     A,0
 0141 27       0076          DAA
 0142 32F301   0077          LD      (TOCKS),A
               0078
 0145 18DD     0079          JR      TIMER_LOOP
               0080
 0147          0081 STOP_THE_CLOCK:
 0147 21F301   0082          LD      HL,TOCKS
               0083
 014A 3E00     0084          LD      A,0
 014C ED6F     0085          RLD                                  ; GET TOP NYBBLE INTO A
 014E C630     0086          ADD     A,'0'                        ; CONVERT TO A NUMBER.
 0150 32EE01   0087          LD      (TIME_MSG.4TH_DIGIT),A
               0088
 0153 3E00     0089          LD      A,0
 0155 ED6F     0090          RLD
 0157 C630     0091          ADD     A,'0'
 0159 32EF01   0092          LD      (TIME_MSG.3RD_DIGIT),A
               0093
 015C 23       0094          INC     HL
 015D 3E00     0095          LD      A,0
 015F ED6F     0096          RLD
 0161 C630     0097          ADD     A,'0'
 0163 32F001   0098          LD      (TIME_MSG.2ND_DIGIT),A
               0099
 0166 3E00     0100          LD      A,0
 0168 ED6F     0101          RLD
 016A C630     0102          ADD     A,'0'
 016C 32F101   0103          LD      (TIME_MSG.1ST_DIGIT),A
               0104
 016F 11D801   0105          LD      DE,TIME_MSG
               0106          BDOS    FN_PRINT_STRING
               0110
 0177          0111 LE_FINI:
               0112          BDOS    FN_SYSTEM_RESET
               0116
               0117 ;***************
               0118
```

```
DATA AREAS                    RML Z80 Ass V 4.1 k    20-Mar-84    Page 3


                0119 *H DATA AREAS
                0120
  017C          0121 SIGNON_MSG:
  017C 0D0A     0122                DEFB    CR,LF
  017E 52454143 0123                DEFM    'REACTION TIMER PROGRAM'
  0194 0D0A0D0A 0124                DEFB    CR,LF,CR,LF
  0198 50524553 0125                DEFM    'PRESS <SPACE> TO START THE TIMER'
  01B8 24       0126 END_OF_STRING
                0127
  01B9          0128 STOP_MSG:
  01B9 0D0A     0129                DEFB    CR,LF
  01BB 50524553 0130                DEFM    'PRESS <SPACE> TO STOP THE TIMER'
  01DA 24       0131 END_OF_STRING
                0132
  01DB          0133 TIME_MSG:
  01DB 0D0A0D0A 0134                DEFB    CR,LF,CR,LF
  01DF 54494D45 0135                DEFM    'TIME TAKEN IS:
  01EE          0136          TIME_MSG.4TH_DIGIT:
  01EE 00       0137                DEFB    0
  01EF          0138          TIME_MSG.3RD_DIGIT:
  01EF 00       0139                DEFB    0
  01F0          0140          TIME_MSG.2ND_DIGIT:
  01F0 00       0141                DEFB    0
  01F1          0142          TIME_MSG.1ST_DIGIT:
  01F1 00       0143                DEFB    0
  01F2 24       0144 END_OF_STRING
                0145
  01F3          0146 TOCKS:         DEFS    1
  01F4          0147 TICKS:         DEFS    1
                0148
  01F5          0149 STACK:
  01F5          0150                DEFS    256.
  02F5          0151 MY_TOP_STACK:
                0152
  0000          0153 END

Symbols:

0005 BDOSE              0000 CR                 0000 DELAY_2_MILLISECO  0024 END_OF_STRING
0001 FN_CONSOLE_IN      0008 FN_CONSOLE_STATUS  0009 FN_PRINT_STRING    0000 FN_SYSTEM_RESET
0001 KEY_PRESSED        0177 LE_FINI            000A LF                 02F5 MY_TOP_STACK
012D SHORT_DELAY        017C SIGNON_MSG         0020 SPACE              01F5 STACK
0100 START              0189 STOP_MSG           0147 STOP_THE_CLOCK     01F4 TICKS
0124 TIMER_LOOP         01DB TIME_MSG           01F1 TIME_MSG.1ST_DIGI  01F0 TIME_MSG.2ND_DIGI
01EF TIME_MSG.3RD_DIGI  01EE TIME_MSG.4TH_DIGI  01F3 TOCKS              0100 TPA
0108 WAIT_FOR_START_KE

No errors detected
```

```
FSIZE function for Propas        RML Z80 Ass V 4.1 k    20-Mar-84    Page 1


                0001 *H FSIZE function for Propas
                0002
 0011 =         0003 fn_search_first         = 11h
 0023 =         0004 fn_compute_file_size    = 23h
 0021 =         0005 fcb.random_record_offset= 33.
 0005 =         0006 BDOS                    = 5
 FFFF =         0007 break                   = -1
                0008
                0009 global  FSIZE
                0010
 0000           0011 FSIZE:
                0012 ; preserve environment
                0013 ; get parameters
                0014
                0015 ;break
 0000 E1        0016         POP     HL              ; return addr
 0001 223A00    0017         LD      (RETURN_ADDR),HL
 0004 D1        0018         POP     DE              ; addr of FCB
 0005 ED533C00  0019         LD      (fcb_addr),DE
                0020
                0021 ; check to see that the file exists
 0009 0E11      0022         LD      C,fn_search_first
 000B CD0500    0023         CALL    BDOS
 000E 3C        0024         INC     A               ; see if it exists
 000F 281D      0025         JR      Z,no_such_file
                0026
                0027 ; ok so lets find how big it is.
                0028
 0011 ED5B3C00  0029         LD      DE,(fcb_addr)
 0015 0E23      0030         LD      C,fn_compute_file_size
 0017 CD0500    0031         CALL    BDOS
                0032
 001A 2A3C00    0033         LD      HL,(fcb_addr)
 001D 112100    0034         LD      DE,fcb.random_record_offset
 0020 19        0035         ADD     HL,DE
 0021 7E        0036         LD      A,(HL)                  ; get low byte
 0022 4F        0037         LD      C,A
 0023 23        0038         INC     HL                      ; get middle byte
 0024 7E        0039         LD      A,(HL)
 0025 47        0040         LD      B,A
 0026 23        0041         INC     HL                      ; get low byte
 0027 7E        0042         LD      A,(HL)
 0028 5F        0043         LD      E,A
 0029 AF        0044         XOR     A
 002A 57        0045         LD      D,A
 002B EB        0046         ex      de,hl
 002C 1806      0047         JR      exit
                0048
 002E           0049 no_such_file:
 002E 3EFF      0050         LD      A,-1
 0030 67        0051         LD      H,A
 0031 6F        0052         LD      L,A
 0032 47        0053         LD      B,A
 0033 4F        0054         LD      C,A
                0055
```

```
FSIZE function for Propas        RML Z80 Ass V 4.1 k    20-Mar-84    Page 2


  0034           0056 exit:
  0034 ED5B3A00  0057         LD      DE,(return_addr)
  0038 D5        0058         PUSH    DE
  0039 C9        0059         ret
                 0060
  003A           0061 return_addr:
  003A           0062         defs    2
                 0063
  003C           0064 fcb_addr:
  003C           0065         defs    2
                 0066
  0000           0067 end


Symbols:

0005  BDOS               FFFF  BREAK              0034' EXIT               0021  FCB.RANDOM_RECORD
003C' FCB_ADDR           0023  FN_COMPUTE_FILE_S  0011  FN_SEARCH_FIRST    0000G FSIZE
002E' NO_SUCH_FILE       003A' RETURN_ADDR

No errors detected
```

```
program free_disc_space;
(*
  Program to calculate and display the amount of free space
  on a disc.
*)

const
   max_no_blocks_by_8   = 4095;  (* 32k/8 -1 *)

type
        byte    = 0..255;
        word    = 0..65535;
        dpb     = record
                    no_sectors_a_track  : word;
                    block_shift_factor  : byte;
                    block_mask          : byte;
                    extent_mask         : byte;
                    size_in_blocks      : word;
                    no_dir_entries      : word;
                    reserved            : array[0..3] of byte;
                    no_reserved_tracks  : word;
                  end;

   alloc_map    = array[0..max_no_blocks_by_8] of byte;

var
   free         : integer;
   drive        : byte;
   drive_name   : char;
   my_dpb       : dpb;
   an_alloc_map : alloc_map;

function free_disc_space(a_drive : byte) : integer;
(*
   find free disc space in sectors.
 *)

const
  unallocated   = 0;

var
  block_no      : integer;
  free_space    : word;

  procedure get_map(a_drive : byte; no_blocks_on_disc_div_8 : word;
                    var buffer_for_alloc_map : alloc_map);
  external;

  Procedure GET_DPB(drive : byte; var a_dpb : dpb);
  external;

  function check_block(block_number : integer;
                       var an_alloc_map : alloc_map) : byte;

  (*
     This function tests if a block is allocated or not,
     if it is it returns a '1', otherwise it returns a '0'
     for free block.
  *)
```

```
  var
     a_bit,this_bit     : byte;
     pwr2               : word;

  begin
    pwr2 := 1;
    this_bit := 0;

(*
   calculate bit no from block no (7-BLOCK_NO/8)
*)
    a_bit := 7 - (block_number mod 8);
    while this_bit < a_bit do
     begin
        pwr2 := pwr2*2;
        this_bit := this_bit +1;
      end;

(*    Return the answer  '0' - free, '1' - used.*)

    check_block := ((an_alloc_map[block_number div 8])
                    div pwr2 ) mod 2;
  end;

begin
  free_space := 0;
  get_dpb(a_drive,my_dpb);      (* get disc characteristics *)

 (* get allocation map *)
  get_map(a_drive,(my_dpb.size_in_blocks+8) div 8,an_alloc_map);

  for block_no := 0 to my_dpb.size_in_blocks do
   (* scan allocation map & count free space *)
    if check_block(block_no,an_alloc_map)= unallocated
      then free_space := free_space + 1;
  free_disc_space := (free_space * (my_dpb.block_mask+1)) div 8;
  (* free space in k bytes *)
end;

begin
  writeln('Simple Disc free space program              v1.2a');
  writeln;
  write('Enter drive name (A..P) ?');
  readln(drive_name);
  drive := ord(drive_name) - ord('A');

  free := free_disc_space(drive);       (* find free space *)

  write('Free space on disc ');
  write(drive_name,': is ',free,' kbytes from a total of ');
  write(((my_dpb.size_in_blocks+1)*(my_dpb.block_mask+1)) div 8);
  writeln(' kbytes.');
end.
```

```
Propas Proc Get_DPB              RML Z80 Ass V 4.1 k    20-Mar-84    Page 1


                0001 *H Propas Proc Get_DPB
                0002
                0003 ;***************************************************************************
                0004 ;GET_DPB:
                0005 ;--------
                0006 ;       Procedure to return a record containing the DPB for the specified
                0007 ;drive. Note that in order to ensure that the DPB is correct, the procedure
                0008 ;first performs a GET_CURRENT_DISC, saves the current drive code then resets
                0009 ;the specified drive.  It then selects the specified drive and performs a
                0010 ;BDOS FN_GET_DPB.  On exit it restore the current drive.
                0011 ;
                0012 ;type
                0013 ;       byte    = 0..255;
                0014 ;       word    = 0..65535;
                0015 ;       dpb     = record
                0016 ;                   no_sectors_a_track  : word;
                0017 ;                   block_shift_factor  : byte;
                0018 ;                   block_mask          : byte;
                0019 ;                   extent_mask         : byte;
                0020 ;                   size_in_blocks      : word;
                0021 ;                   no_dir_entries      : word;
                0022 ;                   reserved            : array[0..3] of byte;
                0023 ;                   no_reserved_tracks  : word;
                0024 ;                 end;
                0025 ;
                0026 ;Procedure GET_DPB(drive : byte; var a_dpb : dpb);
                0027 ;external;
                0028 ;
                0029 ;***************************************************************************
                0030
0005 =          0031 bdos                    = 5
001F =          0032 fn_get_dpb              = 1fh
0025 =          0033 fn_reset_drive          = 25h
0019 =          0034 fn_get_current_disc     = 19h
000E =          0035 fn_seldisc              = 0eh
                0036
000F =          0037 size_dpb                = 15.
                0038
                0039 global getdpb
                0040
0000            0041 getdpb:
0000            0042 get_dpb:
0000 D1         0043         pop     de                      ; get return addr
0001 ED534400   0044         ld      (return_addr),de
0005 E1         0045         pop     hl                      ; get addr a_dpb to return ans.
0006 224800     0046         ld      (var_a_dpb),hl
0009 F1         0047         pop     af                      ; get drive code
000A 324700     0048         ld      (drive),a
                0049
000D CD5C00     0050         call    bdos_get_current_disc   ; save current drive.
0010 324600     0051         ld      (current_disc),a
                0052
0013 110000     0053         ld      de,0
0016 3A4700     0054         ld      a,(drive)               ; calculate drive to reset
0019 3C         0055         inc     a
```

```
Propas Proc Get_DPB             RML Z80 Ass V 4.1 k    20-Mar-84    Page 2


001A 47         0056          ld      b,a
001B 37         0057          scf
001C            0058 loop:
001C CB13       0059          rl      e
001E CB12       0060          rl      d
0020 10FA       0061          djnz    loop
                0062
0022 CD5000     0063          call    bdos_reset_drive        ; reset & re-select specified drive
                0064
0025 3A4700     0065          ld      a,(drive)
0028 5F         0066          ld      e,a
0029 CD4A00     0067          call    bdos_seldisc
                0068
002C CD5600     0069          call    bdos_get_dpb            ; get dpb & copy data into a_dpb
002F ED5B4800   0070          ld      de,(var_a_dpb)
0033 010F00     0071          ld      bc,size_dpb
0036 EDB0       0072          ldir
                0073
0038 3A4600     0074          ld      a,(current_disc)
003B 5F         0075          ld      e,a
003C CD4A00     0076          call    bdos_seldisc            ; restore original drive.
                0077
003F 2A4400     0078          ld      hl,(return_addr)
0042 E5         0079          push    hl
0043 C9         0080          ret
                0081
                0082 ; data areas
                0083
0044            0084 return_addr:
0044            0085          defs    2
                0086
0046            0087 current_disc:
0046            0088          defs    1
                0089
0047            0090 drive:
0047            0091          defs    1
                0092
0048            0093 var_a_dpb:
0048            0094          defs    2
                0095
                0096 ;------------
                0097
                0098 ; declare bdos functions
                0099
004A            0100 bdos_seldisc:
004A 0E0E       0101          ld      c,fn_seldisc
004C CD0500     0102          call    bdos
004F C9         0103          ret
                0104
0050            0105 bdos_reset_drive:
0050 0E25       0106          ld      c,fn_reset_drive
0052 CD0500     0107          call    bdos
0055 C9         0108          ret
                0109
0056            0110 bdos_get_dpb:
```

```
Propas Proc Get_DPB              RML Z80 Ass V 4.1 k    20-Mar-84    Page 3


  0056 0E1F     0111          ld      c,fn_get_dpb
  0058 CD0500   0112          call    bdos
  005B C9       0113          ret
                0114
  005C          0115 bdos_get_current_disc:
  005C 0E19     0116          ld      c,fn_get_current_disc
  005E CD0500   0117          call    bdos
  0061 C9       0118          ret
                0119
  0000          0120 end


Symbols:

0005  BDOS                005C' BDOS_GET_CURRENT_   0056' BDOS_GET_DPB        0050' BDOS_RESET_DRIVE
004A' BDOS_SELDISC        0046' CURRENT_DISC        0047' DRIVE               0019  FN_GET_CURRENT_DI
001F  FN_GET_DPB          0025  FN_RESET_DRIVE      000E  FN_SELDISC          0000G GETDPB
0000' GET_DPB             001C' LOOP                0044' RETURN_ADDR         000F  SIZE_DPB
0048' VAR_A_DPB

No errors detected
```

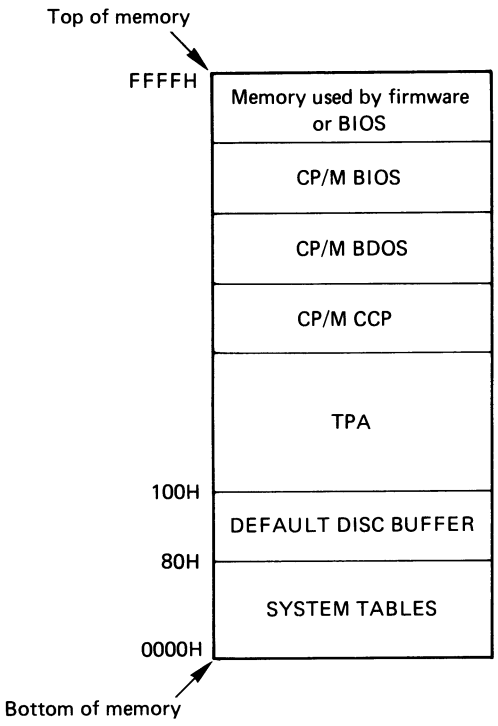
```
Propas Proc GET_MAP             RML Z80 Ass V 4.1 k    20-Mar-84    Page 1


                0001 *h Propas Proc GET_MAP
                0002
                0003 ;*****************************************************************************
                0004 ; GET_MAP
                0005 ; -------
                0006 ;       Returns the allocation map for the specified drive. A call to GET_DPB
                0007 ; should have been called prior to this to a) get the disc size, b) reset the
                0008 ; drive.
                0009 ;
                0010 ;type
                0011 ;       byte    = 0..255;
                0012 ;       word    = 0..65535;
                0013 ;       dpb     = record
                0014 ;                   no_sectors_a_track  : word;
                0015 ;                   block_shift_factor  : byte;
                0016 ;                   block_mask          : byte;
                0017 ;                   extent_mask         : byte;
                0018 ;                   size_in_blocks      : word;
                0019 ;                   no_dir_entries      : word;
                0020 ;                   reserved            : array[0..3] of byte;
                0021 ;                   no_reserved_tracks  : word;
                0022 ;                 end;
                0023 ;
                0024 ;   alloc_map   = array[0..max_no_blocks_by_8] of byte;
                0025 ;
                0026 ;
                0027 ;  procedure get_map(a_drive : byte; no_blocks_on_disc_mod_8 : word;
                0028 ;                          var buffer_for_alloc_map : alloc_map);
                0029 ;  external;
                0030 ;
                0031 ;*****************************************************************************
                0032
0005 =          0033 bdos                    = 5
000E =          0034 fn_seldisc              = 0eh
0019 =          0035 fn_get_current_disc     = 19h
001B =          0036 fn_get_alloc_map        = 1bh
                0037
                0038 global  getmap
                0039
0000            0040 getmap:
0000            0041 get_map:
0000 D1         0042         pop     de                      ; get return addr
0001 ED533700   0043         ld      (return_addr),de
0005 E1         0044         pop     hl                      ; var alloc_map
0006 223D00     0045         ld      (var_alloc_map),hl
0009 E1         0046         pop     hl                      ; no bytes in alloc map
000A 223B00     0047         ld      (map_size),hl
000D F1         0048         pop     af                      ; get drive in a
000E 323900     0049         ld      (drive),a
                0050
0011 CD4500     0051         call    bdos_get_current_disc   ; save current drive
0014 323A00     0052         ld      (current_disc),a
                0053
0017 3A3900     0054         ld      a,(drive)               ; select specified drive
001A 5F         0055         ld      e,a
```

```
Propas Proc GET_MAP              RML Z80 Ass V 4.1 k    20-Mar-84    Page 2


001B CD3F00   0056          call    bdos_seldisc
              0057
001E CD4B00   0058          call    bdos_get_alloc_map      ; get addr alloc map
0021 ED5B3D00 0059          ld      de,(var_alloc_map)      ; copy contents of alloc map
0025 ED4B3B00 0060          ld      bc,(map_size)           ; to BUFFER_FOR_ALLOC_MAP
0029 EDB0     0061          ldir
              0062
002B 3A3A00   0063          ld      a,(current_disc)        ; restore current drive.
002E 5F       0064          ld      e,a
002F CD3F00   0065          call    bdos_seldisc
              0066
0032 2A3700   0067          ld      hl,(return_addr)
0035 E5       0068          push    hl
0036 C9       0069          ret
              0070
              0071 ;----------
              0072
              0073 ; data areas
              0074
0037          0075 return_addr:
0037          0076          defs    2
              0077
0039          0078 drive:
0039          0079          defs    1
              0080
003A          0081 current_disc:
003A          0082          defs    1
              0083
003B          0084 map_size:
003B          0085          defs    2
              0086
003D          0087 var_alloc_map:
003D          0088          defs    2
              0089
              0090
              0091 ;-------------
              0092
              0093 ; bdos routines
              0094
003F          0095 bdos_seldisc:
003F 0E0E     0096          ld      c,fn_seldisc
0041 CD0500   0097          call    bdos
0044 C9       0098          ret
              0099
0045          0100 bdos_get_current_disc:
0045 0E19     0101          ld      c,fn_get_current_disc
0047 CD0500   0102          call    bdos
004A C9       0103          ret
              0104
004B          0105 bdos_get_alloc_map:
004B 0E1B     0106          ld      c,fn_get_alloc_map
004D CD0500   0107          call    bdos
0050 C9       0108          ret
              0109
              0110
```

```
Propas Proc GET_MAP              RML Z80 Ass V 4.1 k    20-Mar-84    Page 3


Symbols:

0005  BDOS                004B' BDOS_GET_ALLOC_MA    0045' BDOS_GET_CURRENT_    003F' BDOS_SELDISC
003A' CURRENT_DISC         0039' DRIVE                001B  FN_GET_ALLOC_MAP     0019  FN_GET_CURRENT_DI
000E  FN_SELDISC           0000G GETMAP               0000' GET_MAP              003B' MAP_SIZE
0037' RETURN_ADDR          003D' VAR_ALLOC_MAP

No errors detected
```

Appendix B: CP/M Memory Map

The organisation of memory under CP/M is shown in the diagram below. Your programs can be placed in the Transient Program Area (TPA).



Appendix C: ASCII Code Tables

The first table below shows the standard ASCII 7-bit character set. In the second table, we have broken this down to show the decimal value of each key as well as the hexadecimal value. Where relevant, the CTRL code for a character is also shown; this shows how you can generate the character by pressing a combination of the CTRL key and another key.

Most of the actions in the first table are machine-independent; however, some of them are not. You should thus check the implementation on your system and update the second table accordingly.

	0	1	2	3	4	5	6	7
0	NUL	DLE		0	@	P	‾	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	£	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	´	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	↑	n	-
F	SI	US	/	?	O	#	o	DEL

Dec	Hex	Name	Effect	Key Stroke	Dec	Hex	Name	Effect
0	00H	NUL	Do nothing	CTRL/@	64	40H		@
1	01H	SOH	Start of header	CTRL/A	65	41H		A
2	02H	STX	Start of text	CTRL/B	66	42H		B
3	03H	ETX	End of text	CTRL/C	67	43H		C
4	04H	EOT	End of transmission	CTRL/D	68	44H		D
5	05H	ENQ	Enquiry	CTRL/E	69	45H		E
6	06H	ACK	Acknowledge	CTRL/F	70	46H		F
7	07H	BEL	Bell	CTRL/G	71	47H		G
8	08H	BS	Backspace	CTRL/H	72	48H		H
9	09H	HT	Horizontal tab	CTRL/I	73	49H		I
10	0AH	LF	Linefeed	CTRL/J	74	4AH		J
11	0BH	VT	Vertical tab	CTRL/K	75	4BH		K
12	0CH	FF	Form feed	CTRL/L	76	4CH		L
13	0DH	CR	Carriage return	CTRL/M	77	4DH		M
14	0EH	SO	Shift out	CTRL/N	78	4EH		N
15	0FH	SI	Shift in	CTRL/O	79	4FH		O
16	10H	DLE	Data Link Escape	CTRL/P	80	50H		P
17	11H	DC1	XON, Transmission ON	CTRL/Q	81	51H		Q
18	12H	DC2	XON, Transmission ON	CTRL/R	82	52H		R
19	13H	DC3	XOFF, Transmission OFF	CTRL/S	83	53H		S
20	14H	DC4	XOFF,Transmission OFF	CTRL/T	84	54H		T
21	15H	NAK	Negative Acknowledge	CTRL/U	85	55H		U
22	16H	SYN	Synchronous Idle	CTRL/V	86	56H		V
23	17H	ETB	End of Transmission Block	CTRL/W	87	57H		W
24	18H	CAN	Cancel	CTRL/X	88	58H		X
25	19H	EM	End of medium	CTRL/Y	89	59H		Y
26	1AH	SUB	Substitute character	CTRL/Z	90	5AH		Z
27	1BH	ESC	Escape key	CTRL/[	91	5BH		[
28	1CH	FS	File separator	CTRL/\	92	5CH		\
29	1DH	GS	Group separator	CTRL/]	93	5DH		]
30	1EH	RS	Record separator	CTRL/^	94	5EH		^
31	1FH	US	Unit separator	CTRL/_	95	5FH		_
32	20H		Space		96	60H		`
33	21H		!		97	61H		a
34	22H		"		98	62H		b
35	23H		£		99	63H		c
36	24H		$		100	64H		d
37	25H		%		101	65H		e
38	26H		&		102	66H		f
39	27H		'		103	67H		g
40	28H		(		104	68H		h
41	29H		)		105	69H		i
42	2AH		*		106	6AH		j
43	2BH		+		107	6BH		k
44	2CH		,		108	6CH		l
45	2DH		-		109	6DH		m
46	2EH		.		110	6EH		n
47	2FH		/		111	6FH		o
48	30H		0		112	70H		p
49	31H		1		113	71H		q
50	32H		2		114	72H		r
51	33H		3		115	73H		s
52	34H		4		116	74H		t
53	35H		5		117	75H		u
54	36H		6		118	76H		v
55	37H		7		119	77H		w
56	38H		8		120	78H		x
57	39H		9		121	79H		y
58	3AH		:		122	7AH		z
59	3BH		;		123	7BH		{
60	3CH		<		124	7CH		|
61	3DH		=		125	7DH		}
62	3EH		>		126	7EH		~
63	3FH		?		127	7FH	DEL	Backspace and delete

Appendix D: BDOS Functions

This appendix starts with a summary of the BDOS-calling mechanism. The BDOS functions are then listed on this page and the next, first in numeric order and then in their appropriate groups.

The rest of the appendix contains detailed descriptions of all functions. The functions themselves are laid out in alphabetic order, with the function code in decimal at the top of each description and in hexadecimal at the bottom.

BDOS-calling mechanism

- Load C register with function code
- Load registers D and E if necessary
- Call BDOS at address 0005H
- Pick up any information returned by BDOS in registers A, H and L

Remember: the BDOS corrupts 99 per cent of household registers!

Summary of BDOS functions

Table D.1 BDOS functions in numeric order

Fn. No.	Function	Fn. No.	Function
0	System Reset	20	Read Sequential
1	Console Input	21	Write Sequential
2	Console Output	22	Create (Make) File
3	Reader Input	23	Rename File
4	Punch Output	24	Return Logged-in Drives
5	List Output	25	Return Current Disc
6	Direct Console I/O	26	Set DMA Address
7	Get I/O Byte	27	Get Allocation Address
8	Set I/O Byte	28	Write Protect Disc
9	Print String	29	Get Read-Only Indicators
10	Read Console Buffer	30	Set File Attributes
11	Get Console Status	31	Get Disc Parameters Address
12	Return Version Number	32	Set/Get User Code
13	Reset Disc System	33	Read Random
14	Select Disc	34	Write Random
15	Open File	35	Compute File Size
16	Close File	36	Set Random Sector
17	Search for First Entry	37	Reset Drive
18	Search for Next Entry	38/39	Reserved
19	Delete File	40	Write Random with Zero Fill

Table D.2 BDOS function groups

Console and simple I/O functions		*Disc functions*	
Console Input	1	Reset Disc System	13
Console Output	2	Select Disc	14
Reader Input	3	Return Logged-in Drives	24
Punch Output	4	Return Current Disc	25
List Output	5	Get Allocation Address	27
Direct Console I/O	6	Write Protect Disc	28
Get I/O Byte	7	Get Read-Only Indicators	29
Set I/O Byte	8	Get Disc Parameters Address	31
Print String	9	Set/Get User Code	32
Read Console Buffer	10	Reset Drive	37
Get Console Status	11		
File handling functions		*Miscellaneous functions*	
Open File	15	System Reset	0
Close File	16	Return Version Number	12
Search for First Entry	17		
Search for Next Entry	18		
Delete File	19		
Read Sequential	20		
Write Sequential	21		
Create (Make) File	22		
Rename File	23		
Set DMA Address	26		
Set File Attributes	30		
Read Random	33		
Write Random	34		
Compute File Size	35		
Set Random Sector	36		
Write Random with Zero Fill	40		

16 CLOSE FILE

Registers on entry	*Registers on exit*
C: 10H **DE:** FCB address	**A:** Directory code

Effect

This function is the reverse of Open File (function 15). It closes the file described by the FCB whose address is held in DE by writing the enclosed details in the appropriate disc directory. Wildcard characters (?) can be used in the filename.

The directory code returned is in the range 0 to 3 for a successful operation. If the filename cannot be found in the directory, FFH (255 decimal) is returned.

The FCB must have been used for an Open file or Create File function.

Example

```
BDOS                =   0005H
FN_CLOSE_FILE       =   16
DATAFILE_FCB:       ...

                    ...
                    ...
                    LD   DE,DATAFILE_FCB
                    LD   C,FN_CLOSE_FILE
                    CALL BDOS

                    INC  A
                    JN   Z,EXIT
FAILED_CLOSE:       ...             ;Identify failed file
                    ...             ;with message
EXIT:               ...
```

COMPUTE FILE SIZE 35

Registers on entry	*On exit*
C: 23H **DE:** FCB address	Bytes 33 to 35 of FCB set to block after end of file

Effect

This function allows you to calculate the size of a file. It returns the number of 128-byte sectors in the file by setting the record pointer in the FCB to the sector immediately after the end of the file. You can then easily append data to the end of the file.

The register pair DE must point to a 36-byte FCB for the file in question, and the FCB must not contain any wildcard characters (?).

On return, bytes 33 and 35 of the FCB contain the file size as a 16-bit value; in effect, this is the sector address of the sector following the end of the file. If byte 35 contains 1 on return, the file contains the maximum sector count (65536).

Compute File Size will always give you an answer whether or not the file exists. Thus, you should check that the file exists by using Search for First.

You can append data to the end of an existing file by merely calling Compute File Size to set the random record position to the end of the file, then performing a sequence of random writes starting at the sector address contained in the record pointer.

When you write a file sequentially, the number of sectors of data in the file corresponds to the number returned by Compute File Size. However, if you create the file in random mode and leave 'holes' in it, Compute File Size will give a larger number than the number of data sectors in the file. If, for example, you write only the last sector of an 8 Megabyte file in random mode (that is, sector 65535), the file size is 65536 but only one sector of data is allocated.

23H

35 COMPUTE FILE SIZE

Example

```
BDOS                    =    0005H
FN_COMPUTE_FILE_SIZE    =    35
DATAFILE_FCB      ...

                  ...
                  ...
                  LD   DE,DATAFILE_FCB
                  LD   C,FN_COMPUTE_FILE_SIZE
                  CALL BDOS

                  ...       ;File size is in bytes 33 & 34
                  ...       ;of DATAFILE_FCB.
```

CONSOLE INPUT 1

Registers on entry	*Registers on exit*
C: 01H	A: ASCII character

Effect

Reads a character from the keyboard into register A.

Printable characters, together with some of the special control characters (RETURN, linefeed and backspace) are echoed to the screen. Tab characters (CTRL/I) are expanded so as to move the cursor to the next tab position. Other control characters are not echoed to the screen.

Control is not returned to your program until a character has been typed in. Thus, function 1 is not suitable for real-time use. If you are writing a real-time program you should use Direct Console I/O (function 6) instead.

Example

```
BDOS            =    0005H
FN_CONSOLE_IN   =    1
                ...
                ...
                LD   C,FN_CONSOLE_IN
                CALL BDOS
                                ;ASCII character
                                 ;is in register A.
                ...
                ...
```

2 CONSOLE OUTPUT

Registers on entry	*Registers on exit*
C: 02H **E:** ASCII character	

Effect

The ASCII character in register E is sent to the screen. Tabs are expanded and checks are made for start/stop scrolling (CTRL/S) from the keyboard.

Example

```
BDOS            =   0005H
FN_CONSOLE_OUT  =   2
                ...
                ...
                LD  E,'C'
                LD  C,FN_CONSOLE_OUT
                CALL BDOS
                ...
                ...
```

CREATE (MAKE) FILE 22

Registers on entry	*Registers on exit*
C: 16H **DE:** FCB address	**A:** Directory code

Effect

Creates an entry in the directory for a file named in the FCB addressed by DE. This FCB must name a file that does not currently exist in the directory. You can make sure that the file does not exist by using a preceding Delete File function.

Create File also activates the FCB for file operations; thus, a subsequent Open File function is not needed.

After a successful operation, a value in the range 0 to 3 is returned in the A register. For an unsuccessful operation, the value FFH (255 decimal) is returned, indicating that the directory is full.

Note that if you attempt to create a file that already exists, you will corrupt the disc structure.

Example

```
BDOS                    =    0005H
FN_CREATE_FILE          =    22
DATAFILE_FCB:     ...

                  ...
                  ...
                  LD   DE,DATAFILE_FCB
                  LD   C,FN_CREATE_FILE
                  CALL BDOS

                  INC  A
                  JN   Z,PROCESS
FAILED_CREATION:  ...                    ;Directory full
                  ...
PROCESS:          ...
```

19 DELETE FILE

Registers on entry	*Registers on exit*
C: 13H **DE:** FCB address	**A:** Directory code

Effect

This function removes all files that match the FCB addressed by register pair DE. The filename and filetype positions can contain wildcard characters (?) but the drive code must be unambiguous.

If the file or files cannot be found, the value FFH (255 decimal) will be returned. Otherwise, a value in the range 0 to 3 will be returned.

Example

```
BDOS                =   0005H
FN_DELETE_FILE      =   19
DATAFILE_FCB:   ...

                ...
                ...
                LD   DE,DATAFILE_FCB
                LD   C,FN_DELETE_FILE
                CALL BDOS

                INC  A
                JN   Z,EXIT
FAILED_DELETION: ...                    ;Identify failed file
                ...                     ;with message
EXIT:           ...
```

DIRECT CONSOLE I/O 6

Registers on entry	*Registers on exit*
C: 06H E: 0FFH (input) char. (output)	A: ASCII character or status

Effect

This function allows you to display output on the screen or read from the keyboard. It bypasses BDOS control character functions such as CTRL/C, CTRL/S, CTRL/Q and CTRL/P.

Upon entry, register E should contain FFH (indicating that you want console input), FEH (indicating that you want the console status), or an ASCII character that you want to output.

When you put FFH in register E, Direct Console I/O returns zero in the A register if no character is ready, otherwise register A contains the next character input. Input is not echoed to the screen.

Versions of CP/M later than 2.2 accept FEH in register E as being a request for the keyboard status. They will return either 0 if no character is waiting, or 1 if a character has been input.

If the value in E is not FFH or FEH, CP/M assumes that it is a valid ASCII character and sends it to the screen. In this case, the contents of the A register are undefined.

You must not mix Direct Console I/O and other console I/O functions.

06H

6 DIRECT CONSOLE I/O

Example

The following code outputs the character Y to the screen using Direct Console I/O.

```
BDOS            =   0005H
FN_DIRECT_IO    =   6
                ...
                ...
                LD  E,'Y'           ;Load character to be output
                LD  C,FN_DIRECT_IO
                CALL BDOS
                ...
                ...
```

The following code uses Direct Console I/O to input the character CTRL/C without rebooting CP/M.

```
BDOS            =   0005H
FN_DIRECT_IO    =   6
CTRL_C          =   3
INPUT_MODE      =   0FFH
                ...
                ...
WAIT_FOR_CHAR:  LD  E,INPUT_MODE    ;Indicate input request
                LD  C,FN_DIRECT_IO
                CALL BDOS
                JR  Z,WAIT_FOR_CHAR ;Character input?

                CP  CTRL_C          ;Test for CTRL/C
                JP  Z,EXIT
PROCESS_CHAR:   ...
                ...
EXIT:           ...
                ...
```

GET ALLOCATION ADDRESS 27

Registers on entry	*Registers on exit*
C: 1BH	**HL:** Allocation address

Effect

This function returns the address in memory of the allocation map for the currently selected drive.

An allocation map is a record of which allocation blocks on a disc are in use and which ones are not. There is an allocation map for each disc and it consists of a bit map, each bit of which corresponds to an allocation block.

Note that the allocation information may be invalid if the disc selected has been marked as read-only.

Example

```
BDOS                    =    0005H
FN_GET_ALLOCATION_ADDRESS =  27
                  ...
                  ...
                  ...
                  LD   C,FN_GET_ALLOCATION_ADDRESS
                  CALL BDOS
                  ...         ;Allocation map address is in
                  ...         ;register pair HL
                  ...
```

1BH

11 GET CONSOLE STATUS

Registers on entry	*Registers on exit*
C: 0BH	A: Status information

Effect

This function allows you to test if a character has been input at the keyboard but to continue processing if it has not.

If a character has been typed, the function returns the value 1; otherwise, 0 is returned.

Example

```
TIMER_LOOP:       LD   C,FN_CONSOLE_STATUS   ;Test if a key has
                                             ;been pressed.
                  CALL BDOS

                  OR   A                     ;Set flags:
                                             ;  - Z  no key pressed
                                             ;  - NZ key pressed.
                  JR   NZ,STOP_THE_CLOCK
                  CALL INCREMENT_THE_CLOCK
                  JR   TIMER_LOOP

STOP_THE_CLOCK:   ...
                  ...
```

GET DISC PARAMETERS ADDRESS 31

Registers on entry	*Registers on exit*
C: 1FH	**HL:** DPB address

Effect

This function returns the address of the BIOS Disc Parameter Block (DPB) in register pair HL. Full details of the DPB contents are given in chapter 6.

Example

```
BDOS                    =    0005H
FN_GET_DPB_ADDRESS      =    31
                ...
                ...
                ...
                LD   C,FN_GET_DPB_ADDRESS
                CALL BDOS

                ...          ;DPB address is in register pair HL
                ...
```

7 GET I/O BYTE

Registers on entry	*Registers on exit*
C: 07H	A: I/O byte value

Effect

This function returns the current value of IOBYTE in register A.

Example

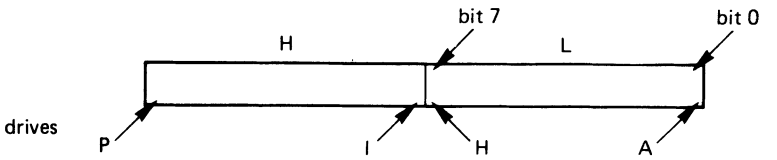
```
BDOS            =   0005H
FN_GET_IO_BYTE  =   7
                ...
                ...
                LD  C,FN_GET_IO_BYTE
                CALL BDOS
                ...         ;I/O byte value is in register A.
                ...
```

GET READ-ONLY INDICATORS 29

Registers on entry	*Registers on exit*
C: 1DH	**HL:** Status info.

Effect

This function returns a value in register pair HL where each bit indicates the status of a specific drive using the arrangement



If a bit is set, the related drive has been write-protected using Write Protect Disc (function 28).

Example

```
BDOS                        =    0005H
FN_GET_READ_ONLY_INDICATORS =    25
                ...
                ...
                LD   C,FN_GET_READ_ONLY_INDICATORS
                CALL BDOS

                LD  A,L
                AND 1
                JR  Z,PROCESS   ;Jump if drive A is not
                                ;write protected

UNPROTECT:      ...              ;Remove write protection
                ...

PROCESS:        ...
```

1DH

5 LIST OUTPUT

Registers on entry	*Registers on exit*
C: 05H **E:** ASCII character	

Effect

The ASCII character in register E is sent to the logical listing device. Tabs are expanded and checks are made for other CTRL codes.

Example

```
BDOS            =   0005H
FN_LIST_OUT     =   5
                ...
                ...
                LD  E,'C'
                LD  C,FN_LIST_OUT
                CALL BDOS
                ...
                ...
```

OPEN FILE 15

Registers on entry	*Registers on exit*
C: 0FH **DE:** FCB address	**A:** Directory code

Effect

This function opens a file whose entry exists in the disc directory with the currently active user number. You should not try to access a file unless it has first been successfully opened.

CP/M scans the referenced disc directory for a match in fields 1 to 14 of the FCB referenced by DE. If you put a wildcard (?) character in any position of the filename, any directory character will be accepted. Normally, you should not do this.

If an entry is found in the directory, the relevant directory information will be copied into bytes 16 to 31 of the FCB. You can then access the file with read and write operations.

If the file cannot be found, Open File returns the value FFH (255 decimal) in register A. If the file is found, a directory code in the range 0 to 3 is returned. You can use the directory code to find the directory entry in the current DMA buffer of your program. If you multiply the contents of the A register by 32 (shift the A register left by 5 bits) this will give you the start address of the directory entry in the buffer.

Note that the current record pointer in the FCB (byte 32) must be zeroed by your program if the file is to be accessed sequentially from the first record.

Example

```
BDOS                =   0005H
FN_OPEN_FILE        =   15
DATAFILE_FCB:   ...

                ...
                ...
                LD   DE,DATAFILE_FCB
                LD   C,FN_OPEN_FILE
                CALL BDOS

                INC  A
                JN   Z,EXIT
FAILED_OPEN:    ...             ;Identify failed file
                ...             ;with message
EXIT:           ...
```

0FH

9 PRINT STRING

Registers on entry	*Registers on exit*
C: 09H **DE:** String address	

Effect

Sends to the screen a character string which is stored in memory at the address held in the register pair DE. The end of the string should be marked by a $ (dollar) character as shown below.

Tab characters (CTRL/I) are expanded so as to move the cursor to the next tab position. A check is also made for start/stop scrolling characters (CTRL/S) and printer echo (CTRL/P) on keyboard input.

This function will not normally print '$' characters (except under certain versions of CP/M and under certain conditions – see chapter 7). To print '$' characters, you must use Console Out (function 2).

Example

```
BDOS            =   0005H
FN_PRINT_STRING =   9
END_OF_STRING   =   '$'

MESSAGE:        DEFM 'This is a string of text'
                DEFB END_OF_STRING
                ...
                LD   DE,MESSAGE   ;Pick up message
                                  ;start address.

                LD   C,FN_PRINT_STRING
                CALL BDOS         ;Print message.
                ...
```

PUNCH OUTPUT 4

Registers on entry	*Registers on exit*
C: 04H **E:** ASCII character	**A:** ASCII character

Effect

This function is a throwback to the days when input/output to CP/M was on paper tape. It sends the ASCII character in register E to the logical punch device.

Some programs such as PIP treat the device PUN as being special and precede/follow any file transfers to PUN: with a series of NULL (0H) characters.

Example

```
BDOS            =    0005H
FN_PUNCH_OUT    =    4
                ...
                ...
                LD   E,'C'
                LD   C,FN_PUNCH_OUT
                CALL BDOS
                ...
                ...
```

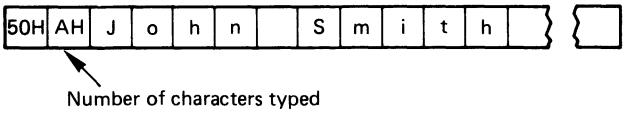
10 READ CONSOLE BUFFER

Registers on entry	*Registers on exit*
C: 0AH **DE:** Buffer address	

Effect

Reads a line input from the keyboard into a buffer whose address is in register pair DE. Keyboard input ends either when the buffer overflows or a carriage return or linefeed is typed.

The input buffer has the following form.



If you have not reserved storage space for an input buffer, the BDOS will use whatever memory the DE registers point to. You should therefore ensure that sufficient space is allocated.

The size of the buffer should be in the range 1 to 255 bytes and you need a region of memory equal to the buffer length plus 2 bytes. If the buffer length is marked as zero, the BDOS will treat it as having a size of 1 byte and will transfer 1 character.

A number of line-editing facilities are available during input. Their description and key combinations are given in the following table.

READ CONSOLE BUFFER 10

Key combination	*Effect*
RUB/DEL	Removes the last character and backspaces one character position
CTRL/C	Reboots the operating system when it is at the beginning of the line
CRTL/E	Forces end of line
CTRL/H	Removes the last character and backspaces one character position
CTRL/J	Terminates line input (LINEFEED)
CTRL/M	Terminates line input (RETURN)
CTRL/P	Printer echo on/off
CTRL/R	Retypes current line on next line, showing all the changes made
CTRL/U	Deletes current line
CTRL/X	Same as CTRL/U

Certain operations return the cursor to the leftmost position (for example, CTRL/U). In these cases, the cursor will be placed in the column position where the prompt ended, thus making operator data input and line correction more legible. In earlier releases of CP/M, the cursor returned to the extreme left margin.

Example

```
BDOS            =   0005H
FN_READ_BUFFER  =   10
                ...
BUFFER_LENGTH   =   80
BUFFER:         DEFS BUFFER_LENGTH
                DEFB BUFFER_LENGTH+1
                ...
                LD   DE,BUFFER      ;Set up buffer
                                    ;start address.
                LD   C,FN_READ_BUFFER
                CALL BDOS
                ...
```

33 READ RANDOM

Registers on entry	*Registers on exit*
C: 21H **DE:** FCB address	**A:** Return code

Effect

This function reads a 128-byte sector of data from a file into the current data buffer. The file must have been opened using Open File (function 15).

The sector number in the file is specified by the contents of bytes 33 to 35 of the FCB. Byte 33 is the least-significant byte and byte 35 the most-significant byte. Bytes 33 and 34 together form a 16-bit sector number in the range 0 to 65535. Byte 35 is used by CP/M to calculate the size of the file and you should set it to zero before using the function.

To use Read Random, you first set the sector number in bytes 33 and 34, set byte 35 to zero and then call the BDOS. Upon return, register A contains one of the error codes shown in table D.3 (on page 132) or zero, if the operation was successful. In the latter case, the current DMA address will hold the block required. The sector number in bytes 33 and 34 will not have been incremented, so a subsequent read will read the same record.

CP/M automatically sets the logical extent and current sector values in the FCB. Thus, you can read or write the file sequentially from the current randomly accessed position. If you do this, you should note that the last block that was randomly read will be reread when you switch to sequential reading, and the last record will be rewritten when you switch to sequential writing.

READ RANDOM 33

Example

```
BDOS                =   0005H
FN_READ_RANDOM      =   33
DATAFILE_FCB        ...

                    ...
                    ...
                    LD   (DATAFILE_FCB.RANDOM_BLOCK),HL ;Load random
                                                        ;block no
                                                        ;from HL
                    LD   DE,DATAFILE_FCB
                    LD   C,FN_READ_RANDOM
                    CALL BDOS
                    JP   Z,PROCESS              ;Read successful

ERROR:              ...
                    ...
PROCESS:            ...
```

Table D.3 File handling error codes

Error code	*Meaning when caused by read operation*	*Meaning when caused by write operation*
1	Reading unwritten data	–
2	–	Disc full
3	Cannot close current extent*	Cannot close current extent*
4	Seek to unwritten extent	–
5	–	New extent cannot be created (directory overflow)
6	File size overflow (byte 35 of FCB too big)	File size overflow

*Does not normally occur under proper system operation. If it does, the error can be cleared by re-reading extent zero (as long as the disc is not physically write-protected).

20 READ SEQUENTIAL

Registers on entry	*Registers on exit*
C: 14H **DE:** FCB address	**A:** Return code

Effect

Reads the next 128-byte sector of data from the file into memory at the current DMA address. The file is described by the FCB whose address is held in DE and this FCB must have been activated by an Open File or Create File function.

If the read is successful, a zero value will be returned in register A, otherwise a non-zero value will be returned. This could happen, for example, if end-of-file is detected.

The data sector is read from a position in the disc logical extent given by byte 32 of the FCB. This byte is then incremented by CP/M to point to the next sector. If byte 32 overflows, the next logical extent is opened and byte 32 reset to zero ready for the next read.

Example

```
BDOS                    =    0005H
FN_READ_SEQUENTIAL      =    20
DATAFILE_FCB:     ...

                  ...
                  ...
                  LD   DE,DATAFILE_FCB
                  LD   C,FN_READ_SEQUENTIAL
                  CALL BDOS

                  JP   Z,PROCESS
FAILED_READ:      ...                    ;End of file reached
                  ...
PROCESS:          ...
```

READER INPUT 3

Registers on entry	*Registers on exit*
C: 03H	A: ASCII character

Effect

This function is a throwback to the days when input/output was on paper tape. It reads the next ASCII character from the logical reader into register A.

Example

```
BDOS            =   0005H
FN_READER_IN    =   3
                ...
                ...
                LD  C,FN_READER_IN
                CALL BDOS
                                ;ASCII character
                                ;is in register A.
                ...
                ...
```

23 RENAME FILE

Registers on entry	*Registers on exit*
C: 17H **DE:** FCB address	**A:** Return code

Effect

Before calling this function, the FCB addressed by register pair DE is set up by you to hold two file descriptions. The first 16 bytes of the FCB contain a description of the old file, the next 16 bytes a description of the new file.

The drive code at byte 0 is used to select the drive; the drive code at byte 16 of the FCB is assumed to be zero.

After a successful operation, a value in the range 0 to 3 is returned in A. If the old filename cannot be found in the directory, FFH (255 decimal) is returned.

Note that this function will not accept the wildcard character (?) in the FCB. You must also ensure that the new file name does not already exist.

Example

```
BDOS                =    0005H
FN_RENAME_FILE      =    23
DATAFILE_FCB:    ...

                 ...
                 ...
                 LD   DE,DATAFILE_FCB
                 LD   C,FN_RENAME_FILE
                 CALL BDOS

                 INC  A
                 JN   Z,PROCESS
FAILED_RENAME:   ...              ;Old filename not in directory
                 ...
PROCESS:         ...
```

RESET DISC SYSTEM 13

Registers on entry	*Registers on exit*
C: 0DH	

Effect

Write Protect Disc (function 28) lets you protect a disc against writing from within your program. Reset Disc System restores the entire disc system to a reset state where all discs are set to read/write. Drive A will be the only currently selected drive and the default data buffer address will be set to 0080H.

Example

```
BDOS                    =    0005H
FN_RESET DISC_SYSTEM    =    13
                  ...
                  ...
                  LD   C,FN_RESET_DISC_SYSTEM
                  CALL BDOS

                  ...
                  ...
```

0DH

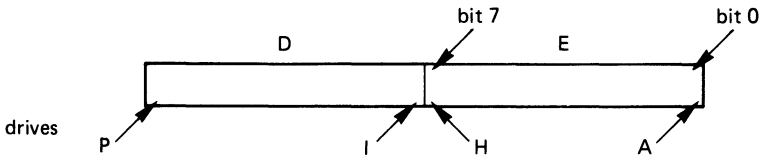
37 RESET DRIVE

Registers on entry	*Registers on exit*
C: 25H **DE:** Drive info	**A:** 00H

Effect

Resets one or more drives so that any write protection caused by a call to Write Protect Disc (function 28) is cancelled and the drives are returned to the Read/Write status.

Drives are specified by setting bits in the register pair DE using the following arrangement.



To maintain compatibility with MP/M, CP/M returns a zero value in register A.

Example

The following code resets drives A, B, C and D.

```
BDOS                =   0005H
FN_RESET_DRIVE      =   37
                ...

                ...
                ...
                LD   D,0                ;Specify drives in
                                        ;registers DE
                LD   E,15
                LD   C,FN_RESET_DRIVE
                CALL BDOS
```

RETURN CURRENT DISC 25

Registers on entry	*Registers on exit*
C: 19H	A: Current disc

Effect

This function returns the drive number of the currently selected drive. The drive numbers are in the range 0 to 15 and correspond to drives A to P.

Example

The following example checks if drive A is the currently selected drive and warns the operator if it is not.

```
BDOS                    =   0005H
FN_RETURN_CURRENT_DRIVE =   25
DRIVE_A                 =   0
                ...
                ...
                LD   C,FN_RETURN_CURRENT_DRIVE
                CALL BDOS

                CP   DRIVE_A
                JR   Z,PROCESS  ;Jump if drive A is currently
                                ;selected drive

PROMPT:         ...             ;Prompt operator to make disc
                ...             ;available and select drive by
                ...             ;program

PROCESS:        ...
                ...
```

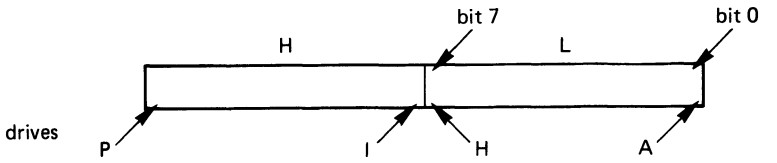
19H

24 RETURN LOGGED-IN DRIVES

Registers on entry	*Registers on exit*
C: 18H	**HL:** Drive status

Effect

This function returns a 16-bit value in the register pair HL, where each bit position corresponds to a drive using the following scheme.



If the appropriate bit is set, the drive is logged-in as a result of either of the following

- An explicit drive selection
- An implicit drive selection caused by a file operation that specified the drive field

Example

The following example checks if drive B is logged-in and warns the operator if it is not.

```
BDOS                    =   0005H
FN_RETURN_LOGGED_IN_DRIVES  =   24
DRIVE_B_BIT_PATTERN     =   2
                ...
                ...
                LD   C,FN_RETURN_LOGGED_IN_DRIVES
                CALL BDOS

                LD   A,L                ;Check logged-in drives
                AND  DRIVE_B_BIT_PATTERN
                JP   Z,PROCESS          ;Jump if drive B
                                        ;logged-in

PROMPT:         ...                     ;Prompt operator to
                ...                     ;make drive available
                ...                     ;and log in drives by
                ...                     ;program
PROCESS:        ...
```

RETURN VERSION NUMBER 12

Registers on entry	*Registers on exit*
C: 0CH	**H:** System type **L:** Version number

Effect

Provides information that helps you write programs which can be used under different versions of CP/M.

When control passes back to your program, CP/M will have inserted in registers H and L two numbers that describe which type and version of the operating system your program is running under. The number in H describes the system type as shown in the left-hand table below. The number in L describes the version number as shown in the right-hand table below.

H	*System type*
0	CP/M
1	MP/M
2	CP/NET

L	*Version*
20	2.0
21	2.1
...	
2F	2.15
30	3.0
31	3.1
...	
3F	3.15

Example

The following example checks the system and version number and exits if the system is not CP/M.

```
BDOS            =   0005H
FN_VERSION_NO   =   12
                ...
                ...
                LD   C,FN_VERSION_NO
                CALL BDOS

                LD   A,H        ;Check system
                OR   A          ;type & version.
                JR   Z,PROCESS  ;Jump if CP/M
EXIT:           ...

PROCESS:        ...
```

17 SEARCH FOR FIRST ENTRY

Registers on entry	*Registers on exit*
C: 11H **DE:** FCB address	**A:** Return code

Effect

This function searches the directory for a match with the file given in the FCB addressed by DE. If the file is not found, the value FFH (255 decimal) is returned, otherwise a number in the range 0 to 3 is returned.

A wildcard facility is available: if you insert a ? character in any position in the filename, filetype or extent field, these positions will be ignored during the search. If the drive code field contains a ? character, the default disc will be searched. In this case, the function will return any matched entry belonging to any user number.

Note that you should not attempt any disc operation between Search for First Entry and Search for Next Entry.

Example

```
BDOS                    =    0005H
FN_SEARCH_FOR_FIRST     =    17
                  ...
                  ...
                  LD   C,FN_SEARCH_FOR_FIRST
                  CALL BDOS

                  INC  A
                  JN   Z,ENTRY_NOT_FOUND
PROCESS:          ...                    ;Process directory entry
                  ...
ENTRY_NOT_FOUND:  ...
```

SEARCH FOR NEXT ENTRY 18

Registers on entry	*Registers on exit*
C: 12H	A: Return code

Effect

This function is similar to Search for First Entry (function 17). However, in this case, the directory is scanned from the last matched entry. When no more directory items match, the value FFH (decimal 255) is returned. Return codes are the same as for Search for First Entry.

You should not attempt any disc operations between calls to Search for First and Search for Next.

Example

```
BDOS                    =    0005H
FN_SEARCH_FOR_NEXT      =    18
                  ...
                  ...
                  LD   C,FN_SEARCH_FOR_NEXT
                  CALL BDOS

                  INC  A
                  JN   Z,NO_MORE_ENTRIES
PROCESS:          ...                    ;Process this directory
                  ...                    ;entry
                  ...
NO_MORE_ENTRIES:  ...
```

14 SELECT DISC

Registers on entry	*Registers on exit*
C: 0EH E: Selected disc	

Effect

This function allows you to change the currently selected drive by program. You put the appropriate drive number in register E and then call the BDOS entry point. Drive numbers are in the range 0 to 15, where 0 refers to drive A and 15 to drive P.

The drive is logged-in until the next cold start, warm start (pressing CTRL/C) or call to Disc System Reset (function 13). If the disc in the drive is changed, the drive will go to a read-only status on the next disc operation to that drive.

FCBs that specify drive code zero refer to the currently selected drive; those that specify codes 1 to 16 refer to a specific drive (A to P).

Example

The following code selects drive B.

```
BDOS                =   0005H
FN_SELECT_DISC      =   14
                ...
                ...
                LD   E,1              ;Select drive B
                LD   C,FN_SELECT_DISC
                CALL BDOS

                ...
                ...
```

SET DMA ADDRESS 26

Registers on entry	*Registers on exit*
C: 1AH **DE:** New DMA address	

Effect

This function defines the start address of the disc data buffer for file operations (the DMA address). The disc data buffer itself will be 128 bytes in size.

The DMA address will be reset to 0080H when one of the following occurs.

- A cold start
- A warm start (pressing CTRL/C)
- Reset Disc System (function 13) is called

Example

```
BDOS                    =    0005H
FN_SET_DMA_ADDRESS      =    26
DATA_BUFFER:      ...

                  ...
                  ...
                  LD   DE,DATA_BUFFER
                  LD   C,FN_SET_DMA_ADDRESS
                  CALL BDOS
```

1AH

30 SET FILE ATTRIBUTES

Registers on entry	*Registers on exit*
C: 1EH **DE:** FCB address	**A:** Directory code

Effect

The *file attributes* are the most significant bits in bytes 1 to 11 of the FCB. They have the following significance.

M/S bit of byte	*Use*
1 to 4	Reserved for use by your programs
5 to 8	Reserved for future system use
9	Read-only attribute
10	System attribute
11	Reserved for future system use

If the Read-only attribute is set, the file will be protected against writing; if it is zero, you can write to the file.

If the System attribute is set, the file will not be listed by a DIR command.

To set an attribute, you first set the appropriate bit in the FCB and then use the Set Attribute function. The DE register pair should address an FCB with an unambiguous filename and the appropriate attributes should be set or unset.

If you want to check whether a file has any attributes set, you should use Search for First (function 17). This will return the file's directory entry in the form of an FCB and you can then look at the appropriate bit to see if the attribute is set or not.

You should not set attributes in any way other than by using Set Attributes.

Example

```
BDOS                    =   0005H
FN_SET_FILE_ATTRIBUTES  =   30
DATAFILE_FCB:   ...

                ...
                ...
                LD   DE,DATAFILE_FCB
                LD   C,FN_SET_FILE_ATTRIBUTES
                CALL BDOS
```

SET/GET USER CODE 32

Registers on entry	*Registers on exit*
C: 20H E: 0FFH (Get) User code (Set)	A: User code or no value

Effect

This function either changes the current user number or tells you what it is.

If register E contains FFH, the value of the current user number is returned in register A; it is in the range 0 to 15.

If register E contains any other value, the current user number is changed to the modulus 16 of this value (the remainder when the value is divided by 16).

Example

The following example sets the user code to 3.

```
BDOS                =   0005H
FN_SET_USER_CODE    =   32
USER_CODE           =   3
                    ...
                    ...
                    ...
                    LD   E,USER_CODE
                    LD   C,FN_SET_USER_CODE
                    CALL BDOS
```

The next example returns the user code and stops the program if it is not 1.

```
BDOS                =   0005H
FN_GET_USER_CODE    =   32
                    ...
                    ...
                    LD   E,255              ;Put FFH in E
                    LD   C,FN_GET_USER_CODE
                    CALL BDOS
                    DEC  A
                    JP   Z,PROCESS          ;User code = 1

EXIT:               ...
                    ...
PROCESS:            ...
```

8 SET I/O BYTE

Registers on entry	*Registers on exit*
C: 08H **E:** I/O byte value	

Effect

This function changes the IOBYTE value to that given in register E.

Example

```
BDOS            =   0005H
FN_SET_IO_BYTE  =   8
                ...
                ...
                LD  C,FN_SET_IO_BYTE
                LD  E,IO

                CALL BDOS
                ...
                ...
```

SET RANDOM SECTOR 36

Registers on entry	*On exit*
C: 24H **DE:** FCB address	Bytes 33 to 35 of FCB set to sector after end of file

Effect

When you are reading or writing a file sequentially, this function returns the random sector address of the current data sector in bytes 33 to 35 of the FCB addressed by register pair DE.

The information returned can be useful in two ways.

- You may scan a file sequentially to extract the positions of various fields. As each field is found, you can then call Set Random Sector to find the random sector position for the current sector. If the size of each data unit is 128 bytes, you can then put the sector position of each field in a table, together with the key of the field, for later retrieval. Once you have scanned the entire file and built up this table, you can move instantly to a specific sector by performing a random read.

  You can generalise the procedure to handle sectors containing variable-length records. Your program need store only the byte position relative to the start of the buffer, along with the field key and sector number, to find the exact start position of the field at a later time.
- If you want to process a file from an 'offset' position along the file, you can use Set Random Sector to help you. All you do is access the file sequentially until the offset is reached, use Set Random Sector to mark the position of the offset and then access the file with random read and write operations from that point

Example

```
BDOS                    =    0005H
FN_SET_RANDOM_SECTOR    =    36
DATAFILE_FCB       ...

                   ...
                   ...
                   LD   DE,DATAFILE_FCB
                   LD   C,FN_SET_RANDOM_SECTOR
                   CALL BDOS

                   ...       ;Random sector position is in
                             ;bytes 33 to 35 of DATAFILE_FCB.
```

24H

0 SYSTEM RESET

Registers on entry	*Registers on exit*
C: 00H	

Effect

Reloads the BDOS and CCP and then passes control to the CCP. The CCP re-initialises the disc system by selecting and logging in disc drive A. It will then select the drive previously selected by the CCP as the default drive.

This function has exactly the same effect as a jump to location BOOT.

Example

```
BDOS            =   0005H
FN_SYSTEM_RESET =   0
                ...
                ...
EXIT:           LD  C,FN_SYSTEM_RESET
                CALL BDOS
```

00H

WRITE PROTECT DISC 28

Registers on entry	*Registers on exit*
C: 1CH	

Effect

This function allows you to protect the currently selected drive until one of the following occurs.

- A cold or warm start (pressing CTRL/C)
- Reset Drive (function 37) is called
- Reset Disc System (function 13) is called

Any attempt to write to the disc will produce the message BDOS ERR on d: R/O.

Note that discs protected by Write Protect Disc are protected only against programs using CP/M BDOS functions. Programs that use BIOS or firmware routines can bypass this method of protection.

Example

```
BDOS                    =    0005H
FN_WRITE_PROTECT_DISC   =    28
                ...
                ...
                LD   C,FN_WRITE_PROTECT_DISC
                CALL BDOS
```

1CH

34 WRITE RANDOM

Registers on entry	*Registers on exit*
C: 22H **DE:** FCB address	**A:** Return code

Effect

This function writes a 128-byte sector of data to a file from the current data buffer. The file must have been opened using Open File (function 15).

The sector number in the file is specified by the contents of bytes 33 to 35 of the FCB. Byte 33 is the least-significant byte and byte 35 the most-significant byte. Bytes 33 and 34 together form a 16-bit sector number in the range 0 to 65535. Byte 35 is used by CP/M; you should set it to zero before using the function.

To use Write Random, you first set the sector number in bytes 33 and 34, set byte 35 to zero and then call the BDOS. Upon return, register A contains one of the error codes shown in Table D.3 (on page 132) or zero, if the operation was successful. The sector number in bytes 33 and 34 will not have been incremented, so a subsequent write will rewrite the same record.

CP/M automatically sets the logical extent and current sector values in the FCB. Thus, you can read or write the file sequentially from the current randomly accessed position. If you do this, you should note that the last sector that was randomly read will be reread when you switch to sequential reading, and the last sector will be rewritten when you switch to sequential writing.

Example

```
BDOS                =   0005H
FN_WRITE_RANDOM     =    34
DATAFILE_FCB    ...

                ...
                ...
                LD   (DATAFILE_FCB.RANDOM_BLOCK),HL ;Load random
                                                    ;block number
                                                    ;from HL
                LD   DE,DATAFILE_FCB
                LD   C,FN_WRITE_RANDOM
                CALL BDOS
                JP   Z,PROCESS              ;Write successful

ERROR:          ...                                    22H
                ...
PROCESS:        ...
```

WRITE RANDOM WITH ZERO FILL 40

Registers on entry	*Registers on exit*
C: 28H **DE:** FCB address	**A:** Return code

Effect

This function is similar to Write Random (function 34) with the exception that a previously unused sector is filled with zeros before the data is written.

Example

```
BDOS                        =    0005H
FN_WRITE_RANDOM_ZERO_FILL   =    40
DATAFILE_FCB:     ...

                  ...
                  ...
                  LD   DE,DATAFILE_FCB
                  LD   C,FN_WRITE_RANDOM_ZERO_FILL
                  CALL BDOS

                  JP   Z,PROCESS        ;No error

ERROR_IN_WRITE:   ...
                  ...

PROCESS:          ...
```

28H

21 WRITE SEQUENTIAL

Registers on entry	*Registers on exit*
C: 15H **DE:** FCB address	**A:** Return code

Effect

Writes a 128-byte sector of data from the current DMA address to the file. The file is described by the FCB whose address is held in DE and this FCB must have been activated by an Open File or Create File function.

If the write is successful, a zero value will be returned in register A, otherwise a non-zero value will be returned. This could happen, for example, if the disc is full.

The data sector is written to a position in the disc logical extent given by byte 32 of the FCB. This byte is then incremented by CP/M to point to the next sector. If byte 32 overflows, the next logical extent is opened and byte 32 reset to zero ready for the next write.

Write operations can take place on an existing file. In this case, new sectors will be appended to the file.

Example

```
BDOS                    =    0005H
FN_WRITE_SEQUENTIAL     =    21
DATAFILE_FCB:     ...

                  ...
                  ...
                  LD   DE,DATAFILE_FCB
                  LD   C,FN_WRITE_SEQUENTIAL
                  CALL BDOS
                  JP   Z,PROCESS
FAILED_WRITE:     ...                ;Disc full or hardware error
                  ...
PROCESS:          ...
```

Appendix E: BDOS Functions of CP/M 80 Family

FN	1.4	2.2	MP/M II	CP/NET 1.2	CP/M PLUS	PERSONAL CP/M	FN
0	SYSTEM RESET	SYSTEM RESET	SYSTEM RESET	SYSTEM RESET	SYSTEM RESET	SYSTEM RESET	0
1	READ CONSOLE	CONSOLE INPUT	CONSOLE INPUT	CONSOLE INPUT	CONSOLE INPUT	CONSOLE INPUT	1
2	WRITE CONSOLE	CONOUT	CONOUT	CONOUT	CONOUT	CONOUT	2
3	READER INPUT	READER INPUT	RAW CONSOLE INPUT	READER INPUT	AUXILIARY INPUT	AUXILIARY INPUT	3
4	PUNCH OUTPUT	PUNCH OUTPUT	RAW CONSOLE OUTPUT	PUNCH OUTPUT	AUXILIARY OUTPUT	AUXILIARY OUTPUT	4
5	LIST OUTPUT	LIST OUTPUT	LIST OUTPUT	LIST OUTPUT	LIST OUTPUT	LIST OUTPUT	5
6		DIRECT CONSOLE I/O	DIRECT CONSOLE I/O	DIRECT CONSOLE I/O	DIRECT CONSOLE I/O	DIRECT CONSOLE I/O	6
7	GET I/O STATUS	GET I/O BYTE	GET I/O BYTE	GET I/O BYTE	AUXILIARY STATUS	AUXILIARY I/P STATUS	7
8	SET I/O STATUS	SET I/O BYTE	SET I/O BYTE	SET I/O BYTE	AUXILIARY O/P STATUS	AUXILIARY O/P STATUS	8
9	PRINT CONTROL BUFFER	PRINT STRING	PRINT STRING	PRINT STRING	PRINT STRING	PRINT STRING	9
10	READ CONSOLE BUFFER	READ CONSOLE BUFFER	READ CONSOLE BUFFER	READ CONSOLE BUFFER	READ CONSOLE BUFFER	READ CONSOLE BUFFER	10
11	GET CONSOLE STATUS	GET CONSOLE STATUS	GET CONSOLE STATUS	GET CONSOLE STATUS	GET CONSOLE STATUS	GET CONSOLE STATUS	11
12	LIFT HEAD	RETURN VERSION NUMBER	RETURN VERSION NUMBER	RETURN VERSION NUMBER	RETURN VERSION NUMBER	RETURN VERSION NUMBER	12
13	INIT BDOS	RESET DISC SYSTEM	RESET DISC SYSTEM	RESET DISC SYSTEM	RESET DISC SYSTEM	RESET DISC SYSTEM	13
14	SELECT DISC	SELECT DISC	SELECT DISC	SELECT DISC	SELECT DISC	SELECT DISC	14
15	OPEN FILE	OPEN FILE	OPEN FILE	OPEN FILE	OPEN FILE	OPEN FILE	15
16	CLOSE FILE	CLOSE FILE	CLOSE FILE	CLOSE FILE	CLOSE FILE	CLOSE FILE	16
17	SEARCH FILE	SEARCH NEXT	SEARCH NEXT	SEARCH NEXT	SEARCH NEXT	SEARCH NEXT	17
18	SEARCH NEXT	SEARCH NEXT	SEARCH NEXT	SEARCH NEXT	SEARCH NEXT	SEARCH NEXT	18
19	DELETE FILE	DELETE FILE	DELETE FILE	DELETE FILE	DELETE FILE	DELETE FILE	19
20	READ SEQUENTIAL	READ SEQUENTIAL	READ SEQUENTIAL	READ SEQUENTIAL	READ SEQUENTIAL	READ SEQUENTIAL	20
21	WRITE SEQUENTIAL	WRITE SEQUENTIAL	WRITE SEQUENTIAL	WRITE SEQUENTIAL	WRITE SEQUENTIAL	WRITE SEQUENTIAL	21
22	CREATE FILE	CREATE FILE	CREATE FILE	CREATE FILE	CREATE FILE	CREATE FILE	22
23	RENAME FILE	RENAME FILE	RENAME FILE	RENAME FILE	RENAME FILE	RENAME FILE	23
24	RET LOGGED-IN DRIVES	RET LOGGED-IN DRIVES	RET LOGGED-IN DRIVES	RET LOGGED-IN DRIVES	RET LOGGED-IN DRIVES	RETURN CURRENT DISC	24
25	GET DRIVE NO	GET CURRENT DRIVE	GET CURRENT DRIVE	GET CURRENT DRIVE	RETURN CURRENT DISC	RETURN CURRENT DISC	25
26	SET DMA ADDRESS	SET DMA ADDRESS	SET DMA ADDRESS	SET DMA ADDRESS	SET DMA ADDRESS	SET DMA ADDRESS	26
27	GET ALLOC	GET ADD ALLOCN MAP	GET ADD ALLOCN MAP	GET ADD ALLOCN MAP	GET ADD ALLOCN MAP	GET ADD ALLOCN MAP	27
28		WRITE PROTECT DISC	WRITE PROTECT DISC	WRITE PROTECT DISC	WRITE PROTECT DISC	WRITE PROTECT DISC	28
29		GET R/O INDICATORS	GET R/O INDICATORS	GET R/O INDICATORS	GET R/O INDICATORS	GET R/O INDICATORS	29
30		SET FILE ATTRIBUTES	SET FILE ATTRIBUTES	SET FILE ATTRIBUTES	SET FILE ATTRIBUTES	SET FILE ATTRIBUTES	30
31		GET DISC PAR. ADD.	GET DISC PAR. ADD.	GET DISC PAR. ADD.	GET DISC PAR. ADD.	GET DISC PAR. ADD.	31
32		SET/GET USER CODE	SET/GET USER CODE	SET/GET USER CODE	SET/GET USER CODE	SET/GET USER CODE	32
33		READ RANDOM	READ RANDOM	READ RANDOM	READ RANDOM	READ RANDOM	33
34		WRITE RANDOM	WRITE RANDOM	WRITE RANDOM	WRITE RANDOM	WRITE RANDOM	34
35		COMPUTE FILE SIZE	COMPUTE FILE SIZE	COMPUTE FILE SIZE	COMPUTE FILE SIZE	COMPUTE FILE SIZE	35
36		SET RANDOM SECTOR	SET RANDOM SECTOR	SET RANDOM SECTOR	SET RANDOM SECTOR	SET RANDOM SECTOR	36
37		RESET DRIVE	RESET DRIVE	RESET DRIVE	RESET DRIVE	RESET DRIVE	37
38			ACCESS DRIVE	ACCESS DRIVE	ACCESS DRIVE		38
39			FREE DRIVE	FREE DRIVE	FREE DRIVE		39
40		WRITE RANDOM ZERO FILL	WRITE RANDOM ZERO FILL	WRITE RANDOM ZERO FILL	WRITE RANDOM ZERO FILL	WRITE RANDOM ZERO FILL	40
41			TEST & WRITE RECORD		TEST & WRITE RECORD		41
42			LOCK RECORD	LOCK RECORD	LOCK RECORD		42
43			UNLOCK RECORD	UNLOCK RECORD	UNLOCK RECORD		43
44			SET MULTI-SECTOR CT.		SET MULTI-SECTOR CT.		44
45			SET BDOS ERROR MODE	SET BDOS ERROR MODE	SET BDOS ERROR MODE	SET BDOS ERROR MODE	45
46			GET DISC FREE SPACE		GET DISC FREE SPACE		46
47			CHAIN TO PROGRAM		CHAIN TO PROGRAM		47
48			FLUSH BUFFERS		FLUSH BUFFERS	FLUSH BUFFERS	48
49					GET/SET SCB		49
50					DIRECT BIOS CALL		50
59					LOAD OVERLAY		59
60					CALL RSX		60
64				LOGIN			64
65				LOGOFF			65
66				SEND MSG ON NETWORK			66
67				GET MSG FROM NETWORK			67
68				GET NETWORK STATUS			68
69				GET CONFIG TABLE ADD			69

70		SET COMPATIBILITY ATT			70
71		GET SERVER CONFIG			71
98			FREE BLOCKS		98
99			TRUNCATE FILE		99
100	SET DIR LABEL		SET DIR LABEL		100
101	RETURN DIR LABEL		RETURN DIR LABEL		101
102	READ FILE XFCB		READ FILE DATE STAMPS AND PASSWORD MODE		102
103	WRITE FILE XFCB		WRITE FILE XFCB		103
104	SET DATE & TIME		SET DATE & TIME		104
105	GET DATE & TIME		GET DATE & TIME		105
106	SET DEFAULT P/WORD	SET DEFAULT P/WORD	SET DEFAULT P/WORD		106
107	RETURN SERIAL NO		RETURN SERIAL NO		107
108			GET/SET PROGRAM RETURN CODE		108
109			GET/SET CON MODE	GET/SET CON MODE	109
110			GET/SET O/P DELIMITERS	GET/SET O/P DELIMITERS	110
111			PRINT BLOCK	PRINT BLOCK	111
112			LIST BLOCK	LIST BLOCK	112
113				DIRECT SCREEN FNS	113
124				BYTE BLOCK COPY	124
125				BYTE BLOCK ALTER	125
128	ABSOLUTE MEMORY REQ				128
129	RELOC MEMORY REQ				129
130	MEMORY FREE				130
131	POLL				131
132	FLAG WAIT				132
133	FLAG SET				133
134	CREATE ONE				134
135	OPEN ONE				135
136	DELETE ONE				136
137	READ ONE				137
138	CONDITIONAL READ ONE				138
139	WRITE ONE				139
140	CONDITIONAL WRITE ONE				140
141	DELAY				141
142	DISPATCH				142
143	TERMINATE PROCESS				143
144	CREATE PROCESS				144
145	SET PRIORITY				145
146	ATTACH CONSOLE				146
147	DETACH CONSOLE				147
148	SET CONSOLE				148
149	ASSIGN CONSOLE				149
150	SEND CLI COMMAND				150
151	CALL RSP				151
152	PARSE FILENAME		PARSE FILENAME		152
153	GET CONSOLE NO				153
154	SYSTEM DATA ADDRESS				154
155	GET DATE & TIME				155
156	RETURN PROCESS DESCRIPTOR ADDRESS				156
157	ABORT PROCESS				157
158	ATTACH LIST				158
159	DETACH LIST				159
160	SET LIST				160
161	COND ATTACH LIST				161
162	COND ATTACH CON				162
163	RETURN MP/M VERSION NO				163
164	GET LIST NO				164

Appendix F: The ZASM Macro Assembler

Examples in this book are written using Research Machines Ltd's Z80 macro assembler ZASM.

The main differences between ZASM and Microsoft's M80 macro assembler, for example, are

- ZASM allows you to have longer symbol names and your programs can therefore be more readable than those generated by M80, which allows a maximum of only 6 significant characters
- ZASM is a true Z80 assembler using Zilog mnemonic conventions
- ZASM will generate the following types of file directly

  .COM files
  .HEX files
  .REL files

  M80 generates only .REL files

Index