CHAPTER 18 Interfacing with BASIC

18.1 Interfacing with Sequential Access Devices

18.1.1 DCB (Device Control Blocks)
To perform I/O operations with sequential access devices such as cassette tapes, etc., a DCB is necessary to specify the conditions for interfacing. DCBs are required for each type of sequential access device ("CAS0:", "COM0:", etc.). The contents of the DCBs are shown below.

Item	Data No. (Size)	Description
1	0 - 3 (4 bytes)	Device name (ASCII code). The four-character device name specified in the file descriptor is entered here.
2	4 (1 byte)	I/O mode. Specified as one of the following values. 10_{16}: Sequential input 20_{16}: Sequential output 30_{16}: Sequential input/output
3	5 - 6 (2 bytes)	Entry point for the OPEN routine. The mode of the file (10_{16}: input, 20_{16}: output) is stored in variable FILMOD (address 068A). The OPEN routine references the mode data and opens the file for input or output.
4	7 - 8 (2 bytes)	Entry point for the CLOSE routine. The CLOSE routine also references variable FILMOD and performs close for input or output.
5	9 - 10 (2 bytes)	Entry point for the input routine for one byte. The input routine inputs one byte is then stored in accumulator A. When the end of the file is detected, FF is entered in variable EOFFLG (address 00F8).
6	11 - 12 (2 bytes)	Entry point for the output routine for one byte. This routine outputs the contents of accumulator A.
7	13 - 14 (2 bytes)	Entry point for EOF routine. This routine sets data FF in accumulator B if the EOF is detected during input. Otherwise, 00 is entered in accumulator B.
8	15 - 16 (2 bytes)	Entry point for LOF routine. This routine enters the number of characters in the buffer or the remaining characters in the file in register D (accumulators A, B).
9	17 - 20	Reserved for data unique to each device.
10	21 (1 byte)	Specifies the column position of the next character to be output (leftmost column is taken to be column '0'). This value is returned when the POS function is called. Normally, this value is initialized to 0 and incremented by 1 each time one byte is output by the output routine. Reset to 0 by CR (code 0D) or LF (code 0A). When this value exceeds the range for the length of one line, and the next character is not CR or LF, the output routine for one byte automatically generates CR or LF and resets the column position to 0.

Item	Data No. (Size)	Description
11	22 (1 byte)	Maximum value of characters per line. May be specified in the range 00 to FF. 00 indicates that the number of characters per line is infinite. As a result, BASIC does not automatically output CR/LF. 00 is set by executing WIDTH (device name), 255.
12	23 (1 byte)	Specifies the size of the print zone when items in a PRINT statement are delimited by "," (comma). The default value is 14.
13	24 (1 byte)	Column position of last print zone. This value is according to the maximum number of characters in the line and the size of the print zone. For example, when the maximum number of characters in the line is 80 and the size of the print zone is 14, this value will be 56.
14	25 (1 byte)	If the number of characters per line can be changed by the WIDTH statement, 00 is entered as the value of this item. Otherwise, 80_{16} is entered.

18.1.2 DCB table

This is a 32-byte table which stores the addresses of the DCBs for each device. Addresses are specified in two bytes and up to 16 DCB addresses can be stored in this table. In the current version, seven addresses are stored in the DCB table and space for nine more addresses is reserved. Device numbers ($0-15_{10}$) are assigned to the DCBs in sequence. The variable name for the DCB table is DCBTAB (address 0657).
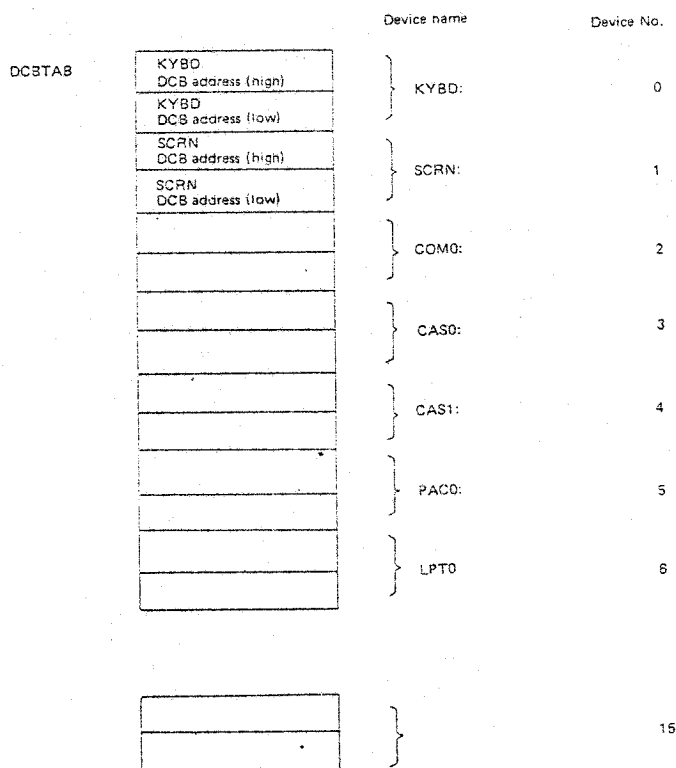

Fig. 18-1 DCB Table

18.1.3 Error processing

When an I/O error occurs during the execution of a routine or when
the required device is busy, the corresponding error code is set in
accumulator B and the following procedure is executed.

```
ERROR         EQU         $8433
              LDAB        #XX       ; SET THE ERROR CODE.
              JMP         ERROR     ; JUMP TO THE ERROR HANDLER.
```

The following error codes are commonly used.

Error code	Message	Description
53_{10}	IO	Error in communication with a peripheral device.
59_{10}	IU	Specified device is in use (busy).
60_{10}	DU	Device is unavailable.

18.1.4 BREAK key processing

The following two procedures are available when BREAK signal
is detected during execution of an I/O operation with a
peripheral device.

(1) Processing BREAK as an error

In this case, processing is identical to that for an I/O error.
Error code 53 (I/O error) is set in accumulator B and control is
transferred to the error handler subroutine (label name ERROR).

```
LDAB      #53    ; ERROR CODE FOR I/O ERROR.
JMP       ERROR
```

This procedure does not effect the other open devices or
variables. When an ON ERROR GOTO statement has not been executed
in the program mode, or when the I/O error occurs in the direct
mode, the following error message will be displayed.

```
I/O ERROR (IN XXXX)
```

If an ON ERROR GOTO statement has been executed in the program
mode, control is transferred to the specified error trap routine.

(2) Abort processing

Control jumps to label name ABTDO (address $A908_{16}$). The BASIC
interpreter clears all variables, closes all files and initializes
all I/O devices. Then, the following message is displayed.

```
ABORT (IN XXXX)
```

18.2 Loading from Expansion Devices

The BASIC interpreter inhibits load from any device other than
"CAS0:, "CAS1:", "PAC0:" and "COM0". Loading from any device other
than these will result in an FC error. However, load from expansion
devices can be enabled by rewriting the hook on the RAM (normally set
to jump to the FC error routine). The RAM hook is 3 bytes long and has
a format: JMP XXX.

Write the entry point address of the program enabling loading from the
expanded device into the address portion of the hook. For load
processing, when control is returned from the OPEN routine, variable
ASCFLG (one byte, address 068C) is checked, and if ASCFLG is 00,
binary format load is performed.

The following two routes are used by the OPEN routine to set the value
of variable ASCFLG.

(1) FF is set in variable ASCFLG when the A option is specified in the
    SAVE statement and 00 is set when the A option is not specified.
    This data is written to the file header during program save and
    set in variable ASCFLG by the OPEN routine during load processing.
(2) If the A option is specified in the SAVE statement, a value other
    than FF is written as the first character of the file. If the A
    option is not specified, FF is written as the above character.
    Therefore, the value of ASCFLG can be set by reading of the first
    character of the file using the OPEN routine.

Hook name
HKLOAD

Address
05E2

Parameters
(A): Device number

Processing sequence

```
        HKLOAD    EQU       $05E2
        FCERR     EQU       $8C70
        LODCNT    EQU       $A6D0
                    :
                  LDD       #LOADC
                  STD       HKLOAD+1
                    :
        LOADCK    CMPA      XX          * check the device number
                  BEQ       LOADOK
                  JMP       FCERR       * GIVE 'FC Error'
        LOADOK    JMP       LODCNT      * CONTINUE LOADING
                    :
```

18.3 ABORT Processing

If an I/O operation is aborted by pressing the BREAK key, the BASIC interpreter initializes all devices and closes all files (communications channels). When one of the devices in the DCB table has been expanded, these devices will also have to be initizalized if I/O to another device is aborted. This initialization is also performed using a hook.

Hook name
HKABTD

Address
063C

Note:
Normally, 39_{16} (RTS command) is stored at address 063C.

18.4 RAM Management

18.4.1 Application files
· Application programs (BASIC interpreter, word processor, etc.) can use
the RAM to store the data required by their systems as application
files.

Application files are protected against use and accidental destruction
by other application programs. Required data can be stored in these
files in the same manner as data for BASIC programs can be stored in
RAM files.
(1) Before execution of an application program (Fig. 18-2)
    All application files are stored in the upper addresses of the RAM.
(2) During execution of an application program (Fig. 18-3)
    The application program reserves a work area for itself by moving
    the application files stored at addresses lower than its own to
    addresses lower down in the free area. However, the location of
    this work area varies according to the status of the other
    application files. Therefore, if a fixed work area is required,
    the area immediately following the system area is reserved for
    this purpose. To secure work areas for execution, each application
    program expands its application files into the fixed and variable
    work areas.
(3) Upon termination of application program execution
    Upon termination of execution of an application program (power
    switch is turned OFF, RESET switch is pressed or normal
    completion), control returns to the Menu leaving the RAM
    allocation as it was during the execution of the application
    program.
    Then, when the same or another application program is selected
    from the Menu, the menu program calls the file reform routine for
    the files of the previously executed application program.
    The file reform routine selects only the required data from the
    fixed and variable work areas to create an application file and
    returns the RAM to the status in (1) above. Control is then
    transferred to the application program selected from the menu.
    For application programs which do not require application files,
    the free area is used as work area as shown in Fig. 18-2.
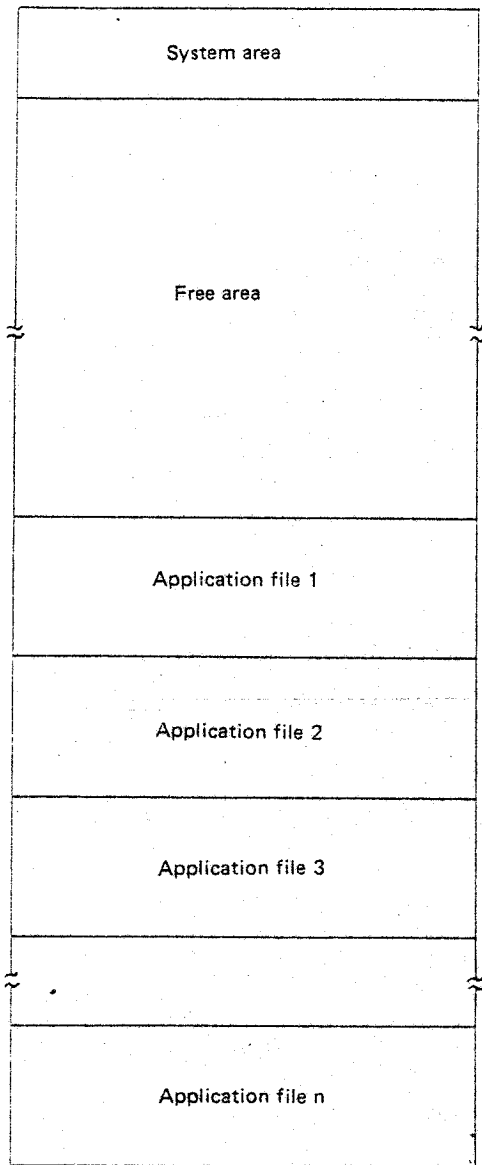   In this case, the file reform routine is not called.

18.4.2 RAM map

System area
Free area
Application file 1
Application file 2
Application file 3
Application file n

High-order

System area
Fixed work area
Application file 1
Variable work area
Application file 3
Application file n
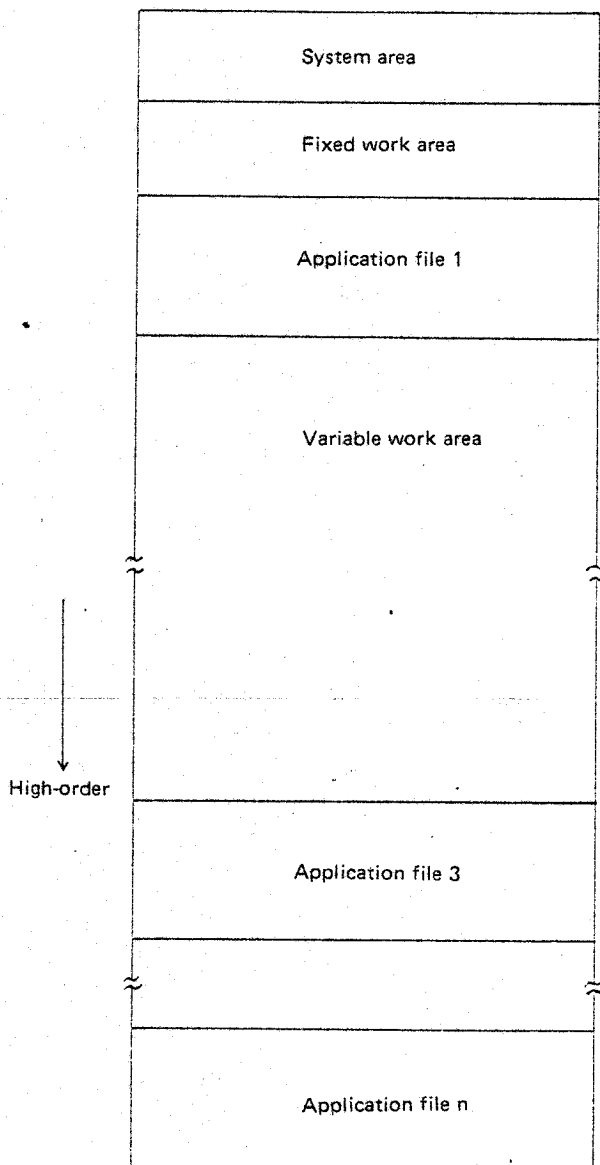
When application file 2 is used

Fig. 18-2 RAM map (1)

Fig. 18-3 RAM map (2)

18-7

18.4.3 Data configuration

BASTAB

Indicates the beginning of the application file. When the system is
initialized, the address set here is the same as that indicated by
RMLTAD.

RMLTAD

Indicates the last address in the RAM +1. The value of RMLTAD is set
when the RAM is checked during system initialization.

CNDADR

Indicates the entry point of the file reform routine. The address of
the file reform routine for the application program is set in this
variable when the application program is executed and the application
files are expanded.

INITAB

INITAB bit 6 is set (logic '1') to indicate that the files must be
reformed before the next application program can be executed. This
flag is set when the value of CNDADR is set.
When this flag is set, the Menu program calls the subroutine whose
address is stored in CNDADR (file reform routine for the previously
executed application program) before transferring control to the
application program selected from the menu. This flag is reset within
the subroutine after the application files are reformed. When
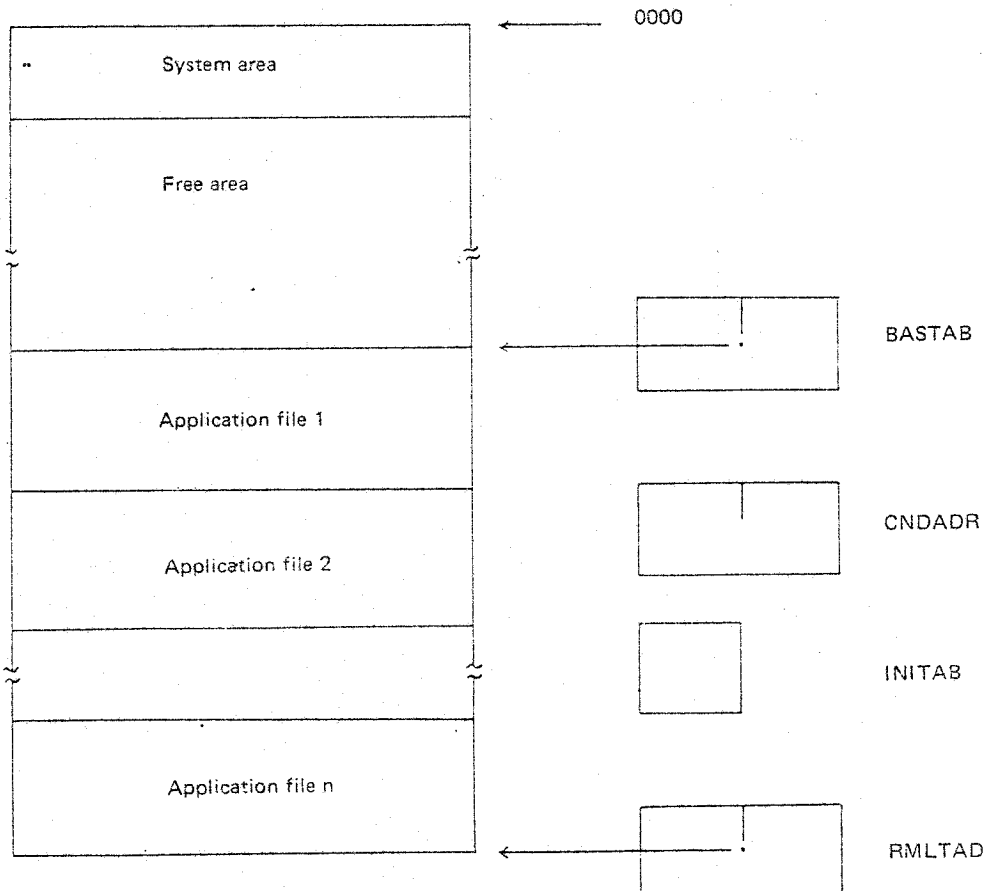application files are not used, this flag must not be set.

Fig. 18-4 Pointers Used for Application Files

18-8

Fig. 18-5 shows an example of when two application files exist simultaneously. The beginning of application file 1 is indicated by BASTAB while the end of application file 2 is indicated by RMLTAD.

(1) File size

The file size is shown by the first two bytes of the file in higher- and lower-order byte sequence. The starting address of the next application file can be obtained by adding the file size to the beginning address of the current file.

(2) Application ID

Application programs are assigned unique one-byte values which are used as IDs. These application IDs are used by application programs when searching for their files.

(3) Data

The data length is the file size -3 bytes. Data format is optional. Unique formats may be used for individual application programs.
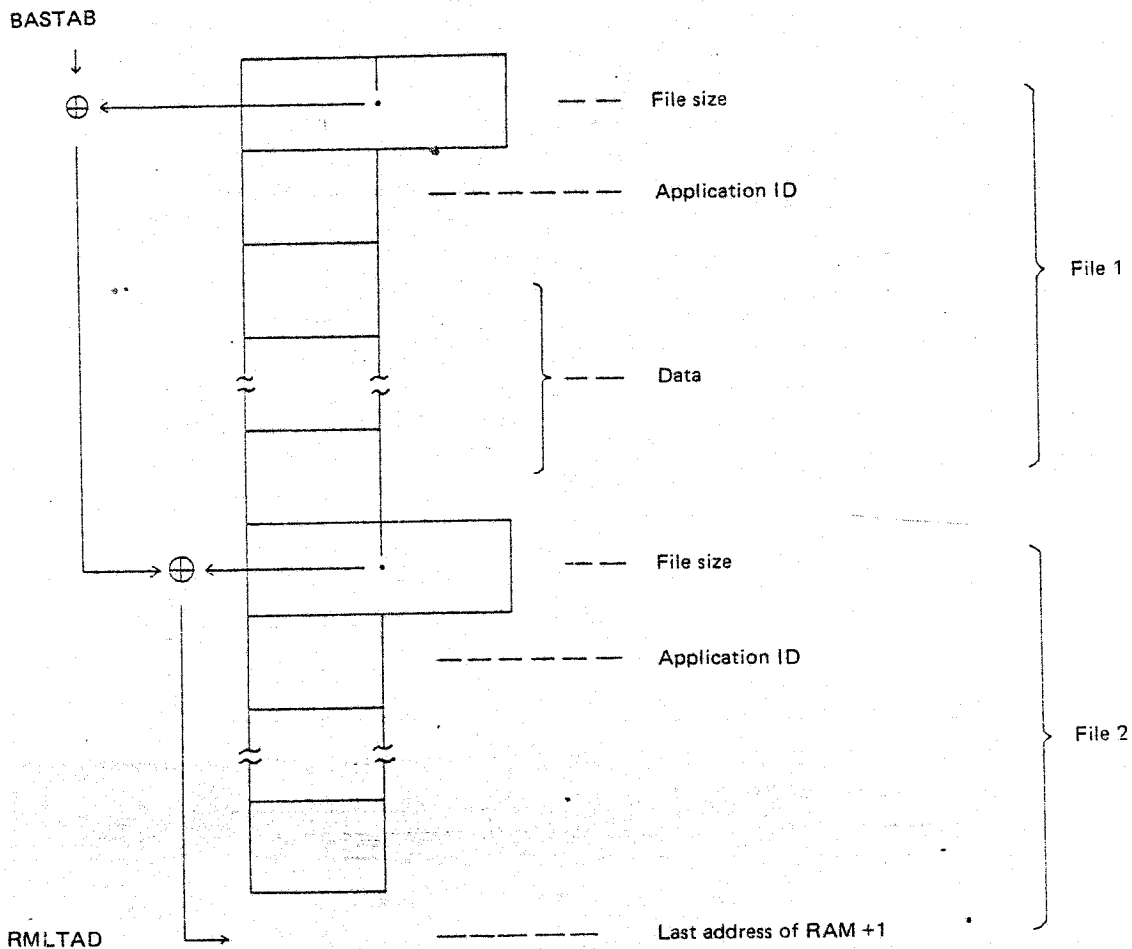
BASTAB

Fig. 18-5 Use of Pointers for Two Application Files

18.4.4 Configuration of BASIC application files

BASIC application files must be stored at the end of the application file area.

(1) Application ID

BASIC : 80_{16}

(2) Warm start hook

The one- to three-byte machine language command stored in this hook is executed to execute BASIC warm start. 39_{16} (RTS command) is set here when the system is initialized.

When the expanded BASIC code is stored in the RAM, a JMP command (C3XXXX) is set in this hook to transfer control to the initialize routine for expanded BASIC.

(3) Lowest address used by BASIC

The address specified in the MEMSET statement is set.


```
                    ———————  File size

   80   :          ———————————  Application ID

                    —  Hot Warm start hook

                    ————  BASIC text area size (n1)

                    ————  RAM file area size (n2)

                    ————  Lowest address in BASIC

                    ————  Programmable function key
                          table (16 bytes x 10 keys = 160 bytes)

                    ————  BASIC program text (n1 bytes)

                    ————  RAM file (n2 bytes)
```
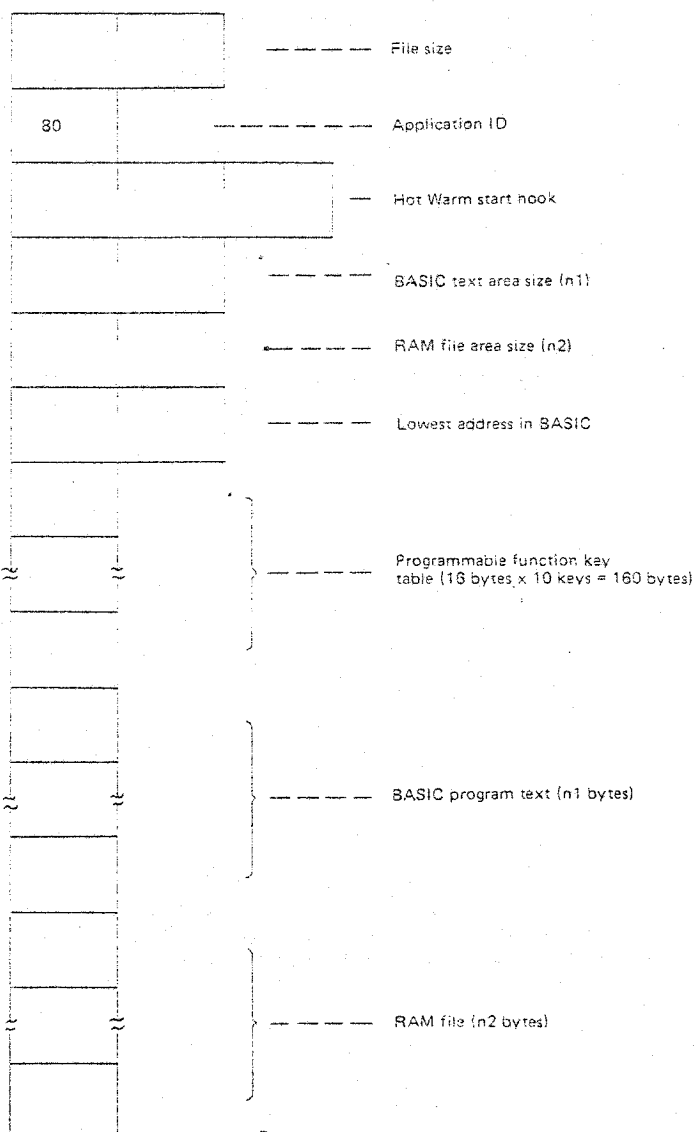
Fig. 18-6 Application File

18.5 Initializing Extended BASIC

18.5.1 Expansion method

When executing warm start, the BASIC interpreter copies the DCBs and the DCB tables from the ROM and initialize the hooks and pointers. To expand BASIC, these hooks and DCBs must be changed after warm start has been executed. Three methods of expanding BASIC (ROM base, RAM base and Disk base) are available.

After initialization has been completed, the BASIC interpreter executes BASIC expansion in the following sequence. The DCBs and hooks are rewritten by the initialize routines in ROM or RAM or by the DISK boot program.

(1) Check executed for whether the expansion ROM has been set in the memory bank in which the BASIC interpreter is currently located. Control is transferred to (3) below, if the expansion ROM is not stored in this memory bank.
(2) The initialize routine for the expansion ROM is executed.
(3) Check executed for whether the floppy disk unit is available for serial communications. If the disk unit is not connected, control is transferred to (5) below.
(4) The boot program is loaded from the floppy disk unit and then executed.
(5) Warm start hook is executed. (If RAM-base expansion is to be executed, a JMP command to transfer control to the initialize routine is set in this hook.)

18.5.2 Expanded ROM format
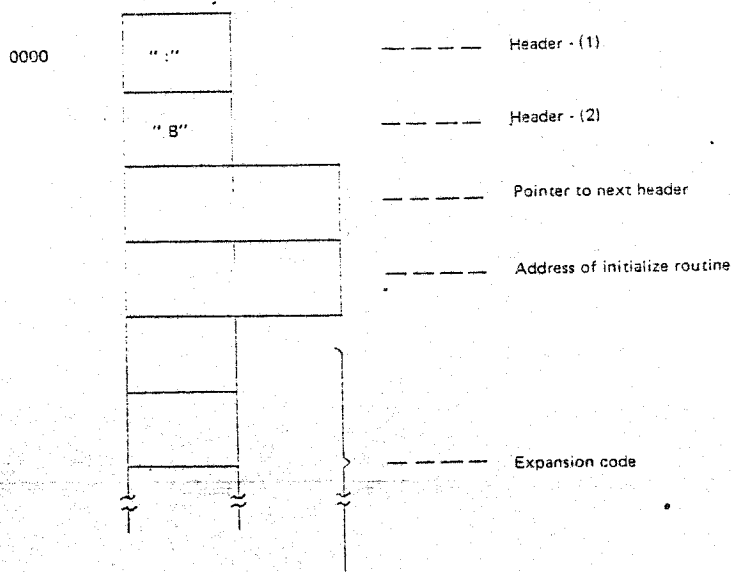
Format for expanding BASIC on a ROM base is shown below.

Fig. 18-7 Expanded ROM Format

Notes:
(1) The expanded ROM for extended BASIC must be located in address 6000_{16}.
(2) Other application programs may be stored in the same ROM with extended BASIC. However, the header of extended BASIC must be located at the starting address of the ROM.

18.5.3 Expansion on RAM base
18.5.3.1 Loading extended BASIC
The memory area for extended BASIC is reserved by creating a special
application file at the end of the other application files. The
procedure for loading extended BASIC is described below.
(1) The BASIC interpreter is executed after initialization (CTRL/@).
(2) Load extended BASIC and the program to reserve the necessary
    memory area into the machine language area (LOADM command).
(3) Execute the program for reserving the memory area.
    This program renews BASTAB and RMLTAD and reserves a RAM area
    sufficient to store extended BASIC. It then moves extended BASIC
    from the machine language area to these files. Also, the warm
    start hooks, etc., in the BASIC application file are rewritten and
    the initialize routine for extended BASIC is attached at the end
    of the initialize routine chain which starts from the warm start
    hook.
(4) Transfer control to the BASIC interpreter warm start routine.
The above sequence makes extended BASIC resident in the RAM. Thereafter,
when warm start is executed, the initialize routine in extended BASIC
rewrites the DCBs and hooks to expand BASIC.
As the area reserved for extended BASIC is at the end of the
application files area, it remains unaffected even if the application
files are used by other application programs.
The extended BASIC codes must be assembled to enable their use at the
destination addresses. However, these addresses of course vary with
the current RAM capacity. In order to enable use of the codes
irrespective of the RAM capacity, extended BASIC must be relocated
after it is moved to the RAM.

18.5.3.2 Program for reserving extended BASIC area
The procedure for reserving the necessary memory area for extended
BASIC is described below.
(1) When control is transferred to the program for reserving memory
    area, the BASIC interpreter is already running and the BASIC
    application files are already extended. The file reform routine is
    therefore called to store only the necessary data in the
    application files. (Fig. 18-8)

```
        LDX       CNDADR
        JSR       , X
        AIM       #$BF, INITAB
```

(2) Next, the BASIC application files are moved forward (BASTAB
    → RMLTAD-1) to reserve the area for extended BASIC. (BASTAB is
    also updated).
    Simultaneously, (RMLTAD) is also updated and set at the head of
    the extended BASIC area to protect extended BASIC. (Fig. 18-9)
(3) Extended BASIC, loaded simultaneously with the memory reserve
    program, is then moved to the newly reserved application files.

(4) A jump command to transfer control to the initialize routine for extended BASIC is set in the warm start hook in the BASIC application file (currently, RTS command) or in the initialize hook for extended BASIC already existing in the RAM.

(5) Control jumps to the BASIC interpreter warm start entry point.

```
LDX        $8004

JMP        0, X
```
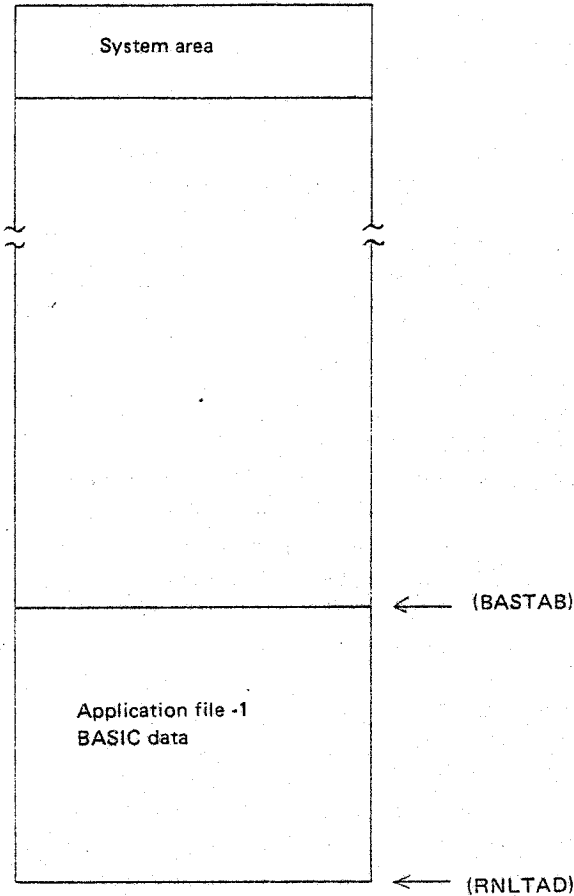

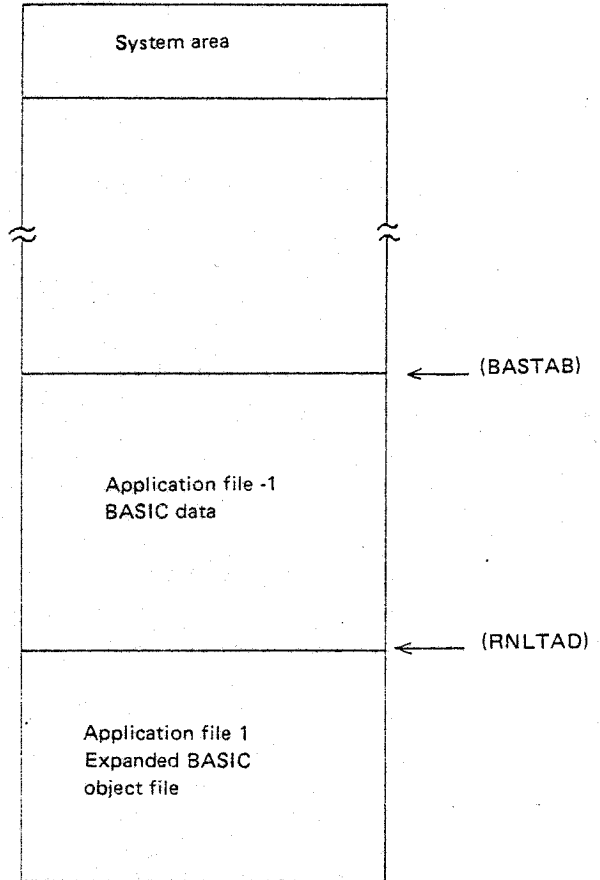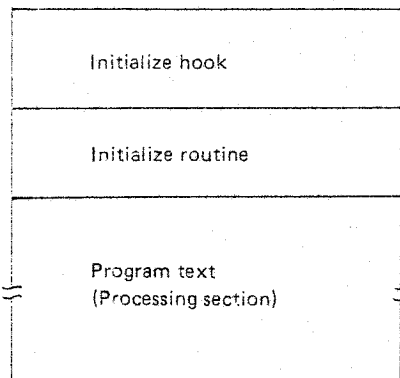Fig. 18-8 Status before Reserving Memory Area   Fig. 18-9 Status after Reserving Memory Area

18.5.3.3 Configuration of extended BASIC object file
The configuration of the extended BASIC object file is shown below.

```
+-----------------------+
|                       |
|   Initialize hook     |
|                       |
+-----------------------+
|                       |
|   Initialize routine  |
|                       |
+-----------------------+
|                       |
|   Program text        |
|   (Processing section)|
|                       |
+-----------------------+
```

(1) Initialize hooks
    The initialize hook consists of the 3 bytes shown below. When
    multiple extended BASICs reside in the RAM, this hook is used to
    link the different initialize routines.
    The initial value of the hook is RTS (39_{16})

```
+-------+-----------+
|       |           |
|  39   | 0 0   0 0 |
|       |           |
+-------+-----------+
```
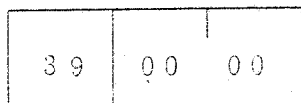
(2) Initialize routine
    The initialize routine starts from the next address following the
    initialize hook. Each time BASIC is warm started, this routine
    rewrites the hooks, adds DCBs, etc.
    When the initialize routine is entered, the pointer to the sign-on
    message is stored in register (X). This is either the current
    BASIC sign-on message or else the sign-on message set by the
    previous initialize routine for extended BASIC. The pointer to the
    sign-on message must be set in register (X) when the initialize
    routine is existed. To display a sign-on message for extended
    BASIC, set the pointer for the sign-on message in register (X) on
    exit from the initialize routine for extended BASIC. The sign-on
    message will then be output when control is returned to the BASIC
    interpreter or when control is transferred to the next initialize
    routine for extended BASIC. If the set message is to be output
    when the initialize routine is entered, STROUT should be called on
    entry.
    If above sign-on message is not to be output, the value of
    register (X) should be retained so that this register can be
    returned to its initial value on exit from the initialize routine.
    In this case, the normal message or the message set by the
    previous initialize routine will be output.
(3) Chaining initialize routines
    When multiple extended BASICs are to be expanded on the RAM, the
    initialize routines for all of these BASICs must be executed at
    warm start. First, as the warm start hook has been rewritten to
    transfer control to the first BASIC initialize routine, this
    routine is executed.

Upon completion of execution of the initialize routine, control
jumps to the initialize hook. At this stage, if the initialize
hook is still set to its initial value, the RTS command will be
executed and control returned to the BASIC interpreter. If the
initialize hook has been rewritten to jump to another BASIC
initialize routine, that routine will be executed next. Initialize
routines can in this way be chained and executed in succession
until the RTS command is encountered.

18.5.3.4 Rewriting warm start and initialize hooks
The procedure for adding the initialize routine for an extended BASIC,
newly loaded in the RAM, to the end of the execution chain starting
from the warm start hook is described below.
(1) The warm start hook in the BASIC application file is checked. If
    the value of the warm start hook has not been rewritten (that is,
    if it is still RTS), it is rewritten to jump to the initialize
    routine for extended BASIC.
    If the warm start hook has already been rewritten (if a jump
    command has been set), operation proceeds to (2) below.
(2) The extended BASIC initialize hook at the jump destination of the
    warm start hook is checked. If it has not been rewritten, it is
    rewritten to jump to the initialize routine for the newly loaded
    extended BASIC.
    If the initialize hook has already been rewritten (if a jump
    command has been set), control is returned to (2). This operation
    is repeated until an initialize hook in which RTS has not been
    rewritten.is encountered.

18.5.4 Extended BASIC work area

The following RAM area is used as the work area for extended BASIC irrespective of whether BASIC has been expanded on the ROM or on the RAM.

    0A38 -- 0A3D   6 bytes

For RAM base expansion, if the work area is insufficient, a work area in the application files is reserved along with the area required for loading extended BASIC. For ROM base expansion, a RAM area is reserved with the application files as in RAM base. This area is then used as the work area. (Subroutines are set in the ROM and executed manually (EXEC command) after system initialize.)
The same procedure is followed to retain data in extended BASIC.

18.6 System Variables and Hook Table

18.6.1 System variables

(1) INITAB (address 0078_{16}, 1 byte)

Bits 0 to 5 and bit 7 are initialize request flags. One bit is assigned for each application. The flag is set (logic "1") to indicate that initialization has been executed. It is reset at system initialize.

, The bit of this variable corresponding to the application program to be executed is checked prior to execution if the program requires initializaiton for its files, etc. If the flag is reset (logic "0"), initialize processing is performed to reserve the necessary work areas, etc., for the files and execution of the program is performed only after the INITAB flag becomes "1". If the flag is set (logic "1"), this means that the application program has already been initialized. It can therefore be executed immediately. INITAB flags are not assigned to application programs which do not require initialization.

Bits currently used are as follows.

        Bit 0 ------ Menu program
        Bit 7 ------ BASIC interpreter

Bit 6 is a file reform request flag. For application programs which require their files be expanded, the pointer to the file reform routine must be set in variable CNDADR and bit 6 of INITAB must also be set after file expansion has completed.

The file reform routine is called by the menu program and resets bit 6 of INITAB after reforming the files.

(2) RMLTAD (address $012C_{16}$, 2 bytes)

This is the pointer for the last address in the RAM +1. This variable is set at system initialize. Also functions as the pointer for the last address of the application files +1.

(3) BASTAB (address 0134_{16}, 2 bytes)

Pointer to the starting address of the application files. Set to the same address as RMLTAD at system initialize.

(4) CNDADR (address 0136_{16}, 2 bytes)

Pointer to the file reform routine. Set by the application program. Valid only if INITAB bit 6 is also set.

(5) DCTAB (address 0657_{16})

DCB table

(6) DEVNUM (address $063E_{16}$)

Enables LOAD from expansion devices.

(7) ASCFLG (address 068C, 2 bytes)

Specifies mode (ASCII or binary) for load. Set by the device OPEN routine.

The BASIC interpreter interprets the flag status as follows:
FF: ASCII load
0: Binary load
(8) OPTBUF (address 068F)
The character string in the file descriptor used to specify
options is set in this buffer. The option routine uses this data.
The file descriptor option statement is set in this buffer in its
original form. It is not placed in brackets. (00) is used as the
end mark. If (00) is entered as the first character, option is
assumed not to have been specified.

18.6.2 Hook table
(1) HKLOAD (address $05E2_{16}$)
Enables LOAD from expansion devices.
(2) HKABTD (address $063C_{16}$)
Used to initialize expansion devices in case of ABORT.

18.6.3 Entry Point Table

	Label name	Address
(1)	ERROR	8433
(2)	ABTDO	A9D8
(3)	FCERR	8C70
(4)	LODCNT	A6D0