P800M Programmer's Guide 2
Volume I: DOM

A publication of

Philips Data Systems B.V.

Apeldoorn, the Netherlands

Publication number 5122 991 27373

June, 1977

Printed in the Netherlands.

This is the first of a five-volume set dealing with the Disc Operating System (non-real time and real time) for the P800M series. It describes the Disc Operating Monitor.

The other volumes of this set are:

II: Instruction Set

III: Software Processors

IV: Disc Real Time Monitor

V: Multi-Application Monitor

Other books pertaining to the P800M series are:

P852M System Handbook

P856M/P857M Handbook

P800M Operator's Guide

P800M Interface and Installation Manual

P800M Software Reference Data

P800M Data Communication User Manual

Great care has been taken to ensure that the information contained in this manual is accurate and complete. However, should any errors or omissions be discovered, or should any user wish to make a suggestion for improving this manual, he is invited to send his comments, written on the sheet provided at the end of this book, to:

SSS-DOC.

at the address on the opposite page.

Table of contents

The Disc Operating Monitor (DOM) is used in a non-real time disc system and is intended mainly as a tool for program development. Communication with the system normally takes place via the operator's typewriter by means of a comprehensive set of control commands, more or less as in time sharing, but not only disc storage but also the other peripherals in the configuration may be used without restriction.

This monitor allows the user to process and maintain on disc all kinds of user data, such as programs in source, object and load format as well as various files. This can be done in interactive as well as in batch processing mode.

All system components, including processors, are disc resident. The scheduled label feature allows some form of multiprogramming.

1 Principles of Operation

With the Disc Operating Monitor, there are two modes of operation:
- conversational, working in sessions
- batch processing, working in jobs.

A session is opened when the user types in his user identification, in reply to the message USERID:, which is output after the system has been loaded or after a previous session or job has been closed. The user may then decide on batch processing or conversational mode.

The user identification is an individual code name for each user, through which he gets access to the system and to his own library files. The user identification must have been declared previously and is then stored by the system in a catalogue on the disc. Thus, only those users whose identification is known to the system can access it. The system itself is also considered as a user, with user identification SYSTEM, with an entry in the disc catalogue (the first entry) and with its own library, i.e. the system software components.

The user communicates with the system through control commands which are normally typed in on the operator's typewriter in conversational mode, a card reader or punched tape reader in case of batch processing.
By means of these commands, the user may create files, call processors, handle his library, delete files, etc.

Per user there is one library on disc, pointed to by the user identification entry in the disc catalogue. All his permanent files are stored in this library, which is located on one disc only, i.e. the disc which contains the directory for that user. This implies that one user cannot have his files stored on several discs, unless, of course, he makes use of several user identifications. However, a user can have access to the system library and

to other user libraries by specifying the user identification of the user of those libraries, but he can only read the data in these files, not write in them.

The files contained in a Library may be of several types: source program files, object modules, load files (programs ready for execution) and files not belonging to any of these types, e.g. files created by user programs.

During a session the user will, outside his library, also need temporary storage space on the disc, either for the processors which he activates through his control commands or for his own programs. This storage space is always allocated to him on the same disc on which his library is located.
Files created during a session are always considered temporary. If the user wants to make them permanent, he must give a specific control command (KPF: Keep File) to have the file stored in his library. Temporary files which are not kept in a library with this control command are automatically deleted at the end of a session or by specific other commands, such as SCR (Scratch).

The allocation of disc storage space takes place dynamically, for library files as well as temporary files. A disc is divided into granules, areas of 8 sectors (one sector = 200 words) and the monitor handles the allocation of these granules to the user's files, when he is creating temporary files. For this purpose, the monitor keeps a table of the granules, to which files they belong and which ones are still available for allocation. On each disc, it also maintains a granule table for that disc.
These tables are updated each time a user gives a command to keep or delete a file. By means of the control commands the user can handle his files and library, call processors to update, assemble, compile, link-edit and debug his programs and he may execute them. In his programs the user may ask the monitor to perform certain functions by means of monitor request, which may be coupled to so-called scheduled labels to achieve a form of multi-tasking.

At the end of a session, the user must type in BYE to close his session. After this the system types out USERID: again and waits for the next session or job with the same or another user. In batch processing, the commands are processed sequentially and automatically until a program itself asks for operator intervention or until an error occurs, in which case the monitor looks for a new job or until a command is given to indicate the end of batch processing (BYE).
It is possible to switch from batch to conversational mode or vice versa during processing, the first choice of mode being made at the beginning of a session. If batch processing is chosen, the first command must be JOB.

Control Command Interpreter

An interesting feature of the system are the catalogued procedures: it is possible to store a sequence of CCI Control Commands on disc under a procedure name. When it is desired to execute this command sequence or part of it, the procedure is called from the input stream and it is possible at that point to fill in certain parameters or indicate which commands are to be executed and which ones are to be skipped in the catalogued procedure.

CONVENTIONS

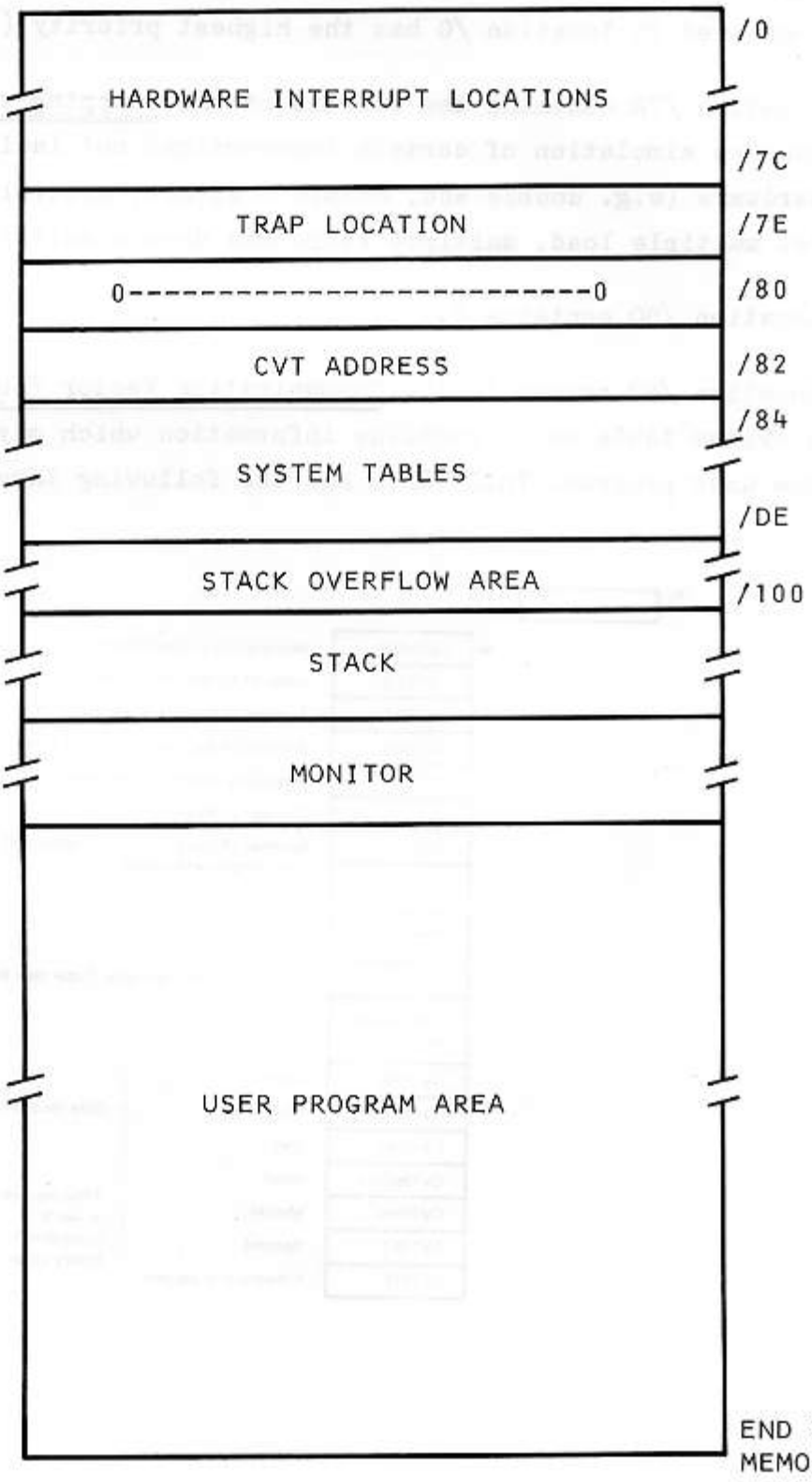
Hexadecimal numbers are written preceded by a slash (/), except when used in an operator control message. So:
LDKL A7,/44FE (instruction) but:
DM 44FE 4600 (operator command)
A granule on a disc is defined as an area of 8 consecutive sectors.
Control commands and processor calls are given to the Control Command Interpreter, which types out S: on the typewriter.
Operator control messages are given to the monitor, which types out M: on the typewriter after the INT button on the control panel has been pressed.
Clusters are object code records, the format of which is described in the System Software Manual.

0	0	0000	00
1	1	0001	01
2	2	0010	02
3	3	0011	03
4	4	0100	10
5	5	0101	11
6	6	0110	12
7	7	0111	13
8	8	1000	20
9	9	1001	21
A	10	1010	22
B	11	1011	23
C	12	1100	30
D	13	1101	31
E	14	1110	32
F	15	1111	33

↑
4-tallig

10	16
100	256
1000	4096

The Memory Layout is as follows:

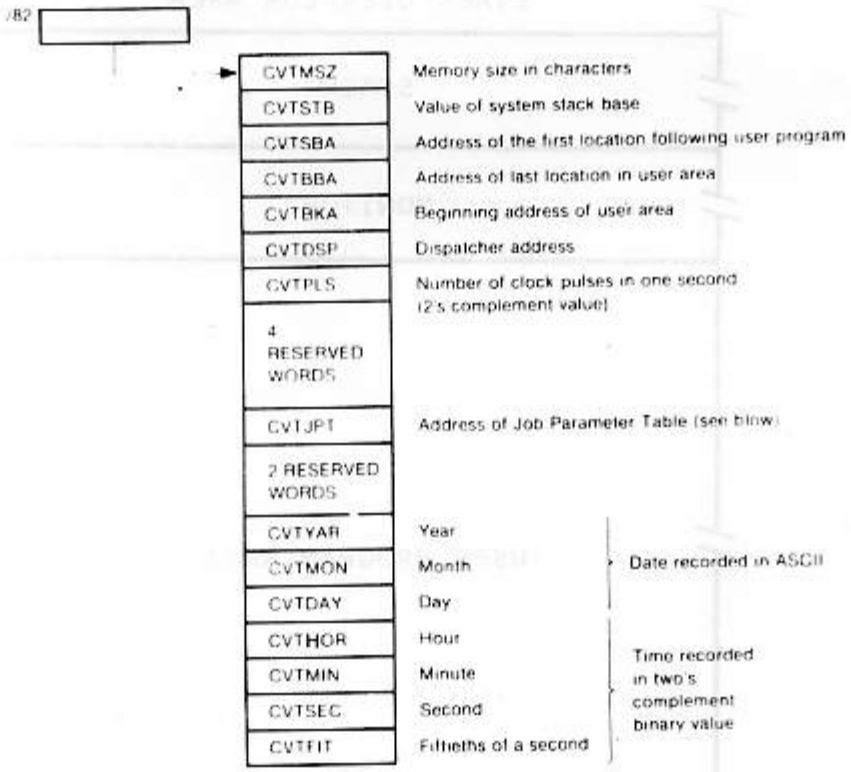


- Location /0 to /7C are hardware interrupt locations. They are hard-wired to internal and external interrupt lines. Each location contains the address of the interrupt routine required to service the interrupt connected to that location. The interrupt connected to location /0 has the highest priority (level 0).

- Location /7E contains the address of the trapping routine which handles simulation of certain instructions not included in the hardware (e.g. double add, double subtract, multiply, divide, multiple load, multiple store and double shift).

- Location /80 contains 0.

- Location /82 points to the Communication Vector Table. This is a system table which contains information which may be of use to the user program. This table has the following layout:



- Locations /84 to /DE are used for other system tables.

- The area occupied by the stack is defined at system generation time. When an interrupt occurs, P- register and PSW are stored here by hardware and a number of registers by software. The number of registers stored depends on whether the interrupt routine servicing the interrupt runs in inhibit mode (anywhere from 0 to 15 registers) or in enable mode or branches to the dispatcher (always 8 registers). The A15 register always points to the next free location in the stack (where all information is stored towards the lower memory addresses). When A15 reaches the value /100 or becomes lower a stack overflow interrupt is given.

- The area after the monitor area is the user area.
  In the user area, one program can be run at a time.
  The area remaining after the program area is reserved for dynamic memory allocation. From this area, blocks of memory space can be requested by the system or by the user. For this purpose the user must send a 'Get Buffer' monitor request. When he does no longer need the buffer, he must send a 'Release Buffer' request.

Programs and routines under DOM run on the basis of an interrupt and priority system consisting of up to 64 levels. These levels are subdivided as follows:

0-47: levels for interrupt routines connected to the hardware interrupt lines	hardware interrupt levels
48 : interruptable monitor service routines	software priority levels
49 : disc file management	
55 : operator routines	
61 : abort module	
62 : user program	
63 : idle task	

Level 0 has the highest priority, 63 the lowest, so all hardware interrupts always have priority over the software levels.

HARDWARE INTERRUPT LINES

The interrupt lines are connected to memory location /0 to /7C. These locations contain the addresses of the interrupt routines which service the internal and external interrupts.
For the interrupts the user can define the priority levels at system generation time.

The following priorities are recommended for the various interrupt lines:

```
/0 :  power failure                - (interrupt location /00)
/1 :  LKM/stack overflow           - (interrupt location /02)
/2 :  real time clock              - (interrupt location /04)
/3 :  not used                                  etc.
/4 :  punched tape reader
/5 :  tape punch
/6 :  operator's typewriter
/7 :  control panel
/8 - /F:  free
/10:  X1215 disc
/11:  disc
/12:  disc
/13:  magnetic tape
/14:  cassette tape
/15:  card reader
/16:  -
/17:  line printer
/18 - /1F: free
```

Software Priority Levels

Only the user program and some of the monitor modules operate on software priority levels: 62, 49, 55, 61 and 63.
The user program is activated by the CCI command RUN.

Level 63 is reserved for the idle task, an instuction sequence to use up idle central processor time. See also Chapter 4: Programs.

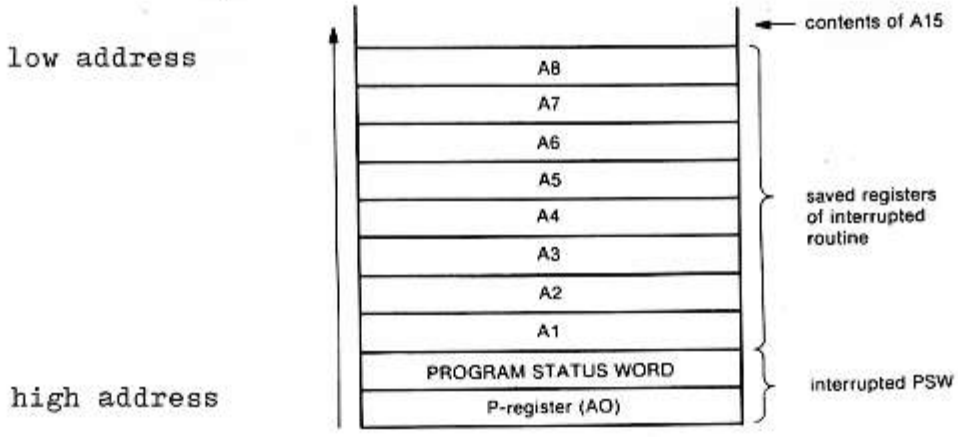
Dispatcher

The dispatcher is a monitor module (M:DISP) running on level 48, which divides central processor time by starting programs according to their priority.
The dispatcher can be entered only from an interrupt routine, i.e. from a level below 48, such as the I/O interrupt and monitor request handlers.

System Interrupt Stack

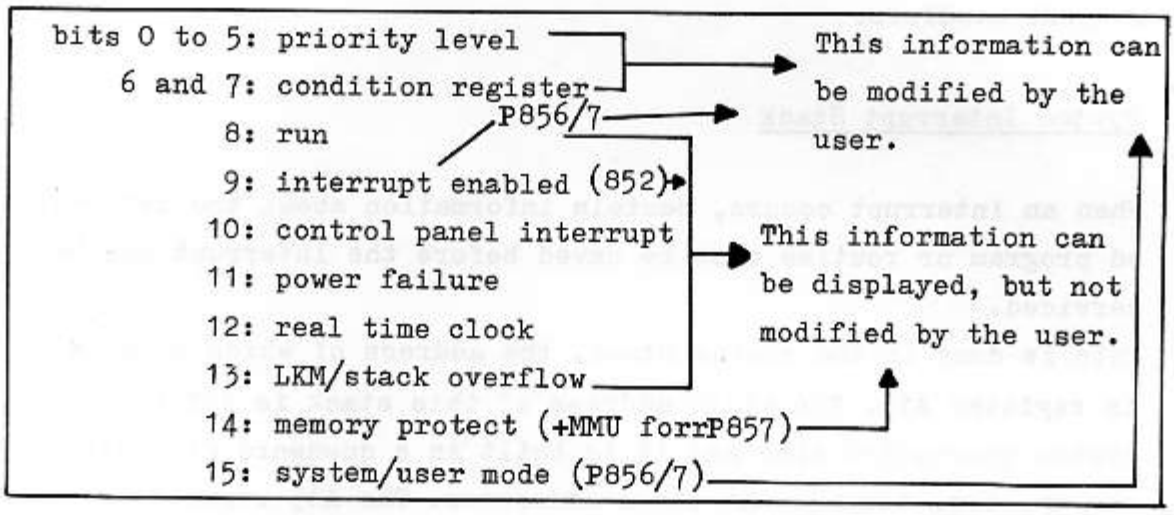
When an interrupt occurs, certain information about the interrupted program or routine must be saved before the interrupt can be serviced.
This is done in the system stack, the address of which is held in register A15. The start address of this stack is defined at system generation time and it is built in a downward direction in memory, i.e. towards the lower addresses. The A15 register always points to the first free location in the stack.
Upon interrupt, PSW and P-register are always saved in this stack and also a number of registers:

- any number if the interrupt routine runs in inhibit mode and takes care of restoring the registers itself;
- registers A1 to A8 if the interrupt routine runs in enable mode or ends with a branch to the dispatcher, because the dispatcher always handles the stack on this basis:



When the stack pointer (A15) reaches or becomes lower than the value /100, the last location before the stack overflow area, an interrupt occurs.

The Program Status Word, stored in the stack upon interrupt, contains the following information:



INTERRUPT ROUTINES

User interrupt routines must have been connected to one of the interrupt locations in memory (locations /0 to /7C). That is, the address of the interrupt routine must be placed in one of these locations, which at the same time determines the priority level of the interrupt routine. i.e. the interrupt routine whose address is loaded in location /0 has the highest priority (0).
The interrupt routine address may be loaded into this location in several ways. One of them is to link-edit and include the routine with the rest of the system modules at system generation time.

When an interrupt occurs, the P-register and PSW of the interrupted program are stored by hardware in the system stack pointed to by the A15 register (see Ch.3) and the system is put in inhibit mode.
Then the interrupt routine receives control and from within the routine the user may store any other registers by software, if he wishes. The interrupt routine may now continue in inhibit mode, or if the user decides that other interrupts must be able to overrule the current one, he may set the system to enable mode by giving an ENB instruction. (Note: If an INH instruction immediately follows the ENB instruction, a dummy instruction must be inserted, because external interrupts are scanned every two instructions. This dummy instruction may, for example, be another ENB, so the correct sequence becomes: ENB-ENB-INH). This, however, entails a substantial difference in the handling of the system stack. If the routine runs in inhibit mode from beginning to end, any of the registers A1 and A14 can be used, provided the user first takes care of storing old contents in the A15 stack and restoring them at the end of the routine. This may, for example, be done as follows:

```
STR     A1,A15          (On P856/7 and when the simulation rou-
STR     A2,A15          tine for multiple load/store has been se-
 |       |  |           lected at sysgen for P852, the sequence is:
STR     A8,A15          MSR 8,A15
    |                    '     '
  coding                  coding
    |                    '     '
    |                   MLR 8,A15
LDR     A8,A15          RTN A15     )
LDR     A7,A15
 |       |  |
LDR     A1,A15
RTN     A15
```

This is the case of an interrupt routine in inhibit mode with a normal return via the A15 stack. The RTN via A15 results in an automatic enable.

However, if other interrupts are to be enabled during a routine or the user makes an absolute branch from the interrupt routine to the dispatcher (ABL M:DISP; for dispatcher address: see CVT), he must take care that before the ENB or ABL instruction is given, the A15 stack contains only P, PSW and registers A1 to A8 inclusive, because on this basis the A15 stack is handled.

Conventions

- Interrupt routines must start by saving the old contents of the registers to be used in the routine.
- Before returning via A15, the old register contents must be restored.

  If a branch is made to the dispatcher, the stack must contain P, PSW and register A1 to A8, so any other registers used, must have been restored before making the branch.
- In case of an interrupt routine for internal interrupts (LKM/ stack overflow, real time clock, power failure, control panel), an RIT instruction must be given at the beginning of the routine, to reset the interrupt. See Volume II.

SOFTWARE LEVEL PROGRAMS

User programs run on one level: software level 62.

Level 63: Idle Task

Level 63 is reserved for the idle task, an instruction sequence to use up idle CPU time. It is memory resident and consists of 2 instructions:

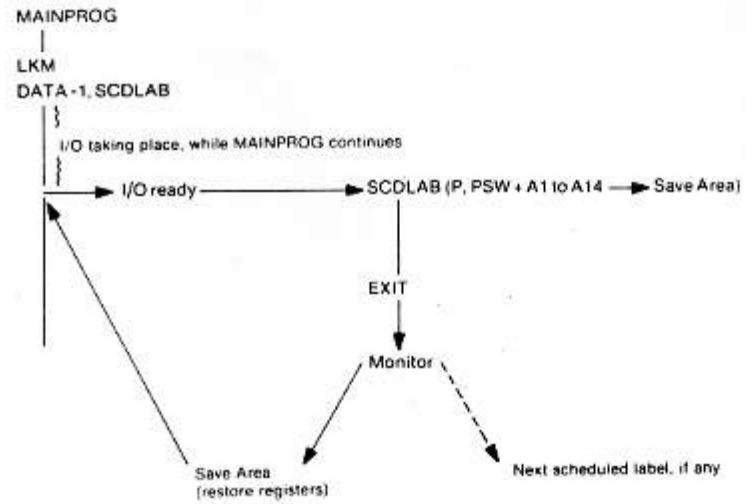
```
RF   *+2
RB   *-2
```

SCHEDULED LABELS

The scheduled label is a feature which allows the user to do a sort of multi-tasking by attaching a routine to a monitor request. To this end, the user specifies the monitor request as having the two's complement of the DATA number indicating the monitor function which is to be executed, followed by the label of the routine which must be executed upon completion of the request. For example:

```
normal I/O request:      with scheduled label:
  LDK   A7,CODE            LDK   A7,CODE
  LDKL  A8,ECBADR          LDKL  A8,ECBADR
  LKM                      LKM
  DATA 1                   DATA -1, SCDLAB
```

In this case, the routine SCDLAB is to obtain control on completion of the I/O monitor request specified. When a scheduled label is attached to an I/O request, one should not set the wait bit, so that the program can continue concurrently with the I/O operation requested. When that operation is finished, control is passed to the SCDLAB routine, and the P-register, PSW and registers A1 to A14 are stored in the 16-word save area (SAVADR) in front of the user program. (When entering a scheduled label routine, only the value of A8 (ECBADR) is significant.) When the routine is finished and exits, control is returned to the monitor, which passes control to a possible following scheduled label routine, or back to the main program by restoring the registers and PSW from the save area. This can be illustrated as follows:



Any number of scheduled labels may be given in a program. However, it is possible that one scheduled label is blocking execution of another one because it is active, i.e. using central processor time. In such cases the address(es) of the queued scheduled label routines are temporarily stored in a table (FILLAB). The maximum number of scheduled label addresses which may be stored in this table at any one time is the number defined at sysgen. This is the only restriction. Queued scheduled labels are treated on a first-in-first-out basis.

Note:

Although it is possible to give a Wait monitor request within a scheduled label routine, this is not normally recommended, for it blocks the whole program.

Example:

This example illustrates how scheduled labels are queued in the FILLAB table, when their execution is blocked in case of an I/O operation on a slow device.

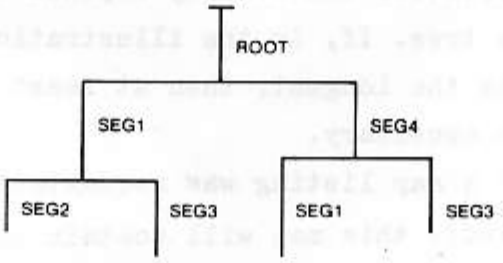
In a program there are three monitor requests for output: onto tape punch, typewriter and line printer. To each of these requests a scheduled label is attached. Each of these scheduled labels requests output on the typewriter. In such a case, it will be evident that the line printer output will be finished first and thus that the scheduled label attached to that request will receive control first. Now, when the other two output operations in the main program are finished, the scheduled labels attached to them will be queued in FILLAB, for they are also requesting output on the typewriter which is still busy with the last scheduled label. When that one is finished control is passed on the last one entered in the FILLAB table.

Note, that in this case the third scheduled label is the first to receive control, because the I/O to which it was attached was for line printer and terminated before the other two.

```
        IDENT SLAB
BUF1    DATA '1234567890'      SPECIFYING THE CHARACTERS TO BE
BUF2    DATA 'ABCDEFGHIJ'      OUTPUT
BUF3    DATA 'ZYXWVUTSRQ'
BUF4    DATA '1A2B3C4D5E'
BUF5    DATA 'BUF5'
BUF6    DATA 'BUF6'
DCB1    DATA 5,BUF1,5,0,0,0    DECLARING THE EVENT CONTROL
DCB2    DATA 5,BUF2,40,0,0,0   BLOCKS FOR THE OUTPUT OPERATIONS.
DCB3    DATA 3,BUF3,5,0,0,0    5 IS THE FILE CODE FOR TYPEWRITER,
DCB4    DATA 2,BUF4,12,0,0,0   3 FOR PUNCH, 2 LINE PRINTER.
DCB5    DATA 5,BUF5,6,0,0,0
DCB6    DATA 5,BUF6,6,0,0,0

START   LKD   A7,6             MAIN PROGRAM STARTS AND REQUESTS
        LDKL  A8,DCB1          STANDARD OUTPUT OF BUF1 ON THE
        LKM                    TYPEWRITER. SCHEDULED LABEL SLAB1
        DATA -1,SLAB1          ATTACHED THIS REQUEST.
        LDK   A7,6             MAIN PROGRAM REQUESTS STANDARD
        LDKL  A8,DCB3          OUTPUT OF BUF3 ON THE TAPE PUNCH.
        LKM                    SCHEDULED LABEL SLAB2 ATTACHED
        DATA -1,SLAB2          TO THIS REQUEST.
        LDK   A7,6             MAIN PROGRAM REQUESTS STANDARD
        LDKL  A8,DCB4          OUTPUT OF BUF4 ON THE LINE PRINTER.
        LKM                    SCHEDULED LABEL SLAB3 ATTACHED
        DATA -1,SLAB3          TO THIS REQUEST.
        LKM
        DATA 3                 MAIN PROGRAM EXIT

SLAB1   LDK   A7,6             SLAB1 REQUESTS STANDARD OUTPUT
        LDKL  A8,DCB2          OF BUF2 ON THE TYPEWRITER
        LKM
        DATA 1
        LKM
        DATA 3                 AND EXITS
SLAB2   LDK   A7,6             SLAB2 REQUESTS STANDARD OUTPUT
        LDKL  A8,DCB5          OF BUF5 ON THE TYPEWRITER
        LKM
        DATA 1
        LKM
        DATA 3                 AND EXITS
SLAB3   LDK   A7,6             SLAB3 REQUESTS STANDARD OUTPUT
        LDKL  A8,DCB6          OF BUF6 ON THE TYPEWRITER
        LKM
        DATA 1
        LKM
        DATA 3                 AND EXITS

        END   START
```

PROGRAM SEGMENTS AND OVERLAY STRUCTURE

If a program becomes very large, it may be possible to divide it into segments and execute it in an overlay structure. Each segment, in such a case, must be considered as a self-contained program. These segments are treated as programs: they are given names and they are catalogued with KPF (Keep File) commands, under the same user identification. The user must first decide on the overlay structure in which he will have his programs executed. To this end he sets up a so-called 'tree' of the segments which form his program. This must be done in such a way that those segments which are related to each other, through references, form a path i.e. a sequence of contiguous segments which can be in memory at the same time and start with the first segment (root). This is shown in the example below:



In this illustration we can discern the following paths:

ROOT - SEG1 - SEG2
ROOT - SEG1 - SEG3
ROOT - SEG4 - SEG1
ROOT - SEG4 - SEG3

These segments have been previously catalogued under the names SEG1, SEG2, SEG3 and SEG4 and they are declared as follows:

```
SEG SEG1,SEG2,SEG3,SEG4
RUN ROOT
```

The first control command defines these programs as segments of an overlay structure and the second command starts the execution of the program ROOT, which is the first segment of the overlay structure. ROOT takes care of loading SEG1 or SEG4, which, in turn, load the other segments.

The design of such an overlay structure depends on several factors: the available memory space, the frequency of use of each segment, the relations between the different segments. The root segment is the program that gets control at the start of execution. It contains those program parts which have to be in memory throughout program execution. When a reference is made to a place in another segment, that segment must first be loaded. If another path is required by the root, it overlays the previously loaded segments or part of them.
The same segment may be implemented in different paths in a tree, if it is required by other segments in that path, as shown in the illustration above. Any overlay structure can thus be used, since the user himself is responsible for calling the loader and transferring control to other segments.

The memory area required for the overlay depends on the length of the longest path in a tree. If, in the illustration above, the path ROOT-SEG1-SEG3 is the longest, then at least the area required for those segments is necessary.
At link-edit time, if a map listing was requested (M parameter in the LKE control command), this map will contain the heading L= XXXX, indicating the program length. If the user wants to use Get and Release Buffer requests, he must reserve an a area, in the root, equal to the combined Lengths of all the segments in the longest path of his overlay structure. This also goes for the blank common. Buffers will be allocated after the last word loaded into memory when the root is started through a RUN command.

When a segment is loaded, it is always loaded at its loading address + 8, because the Linkage Editor generates four words preceding it and fills them with:
- Start address
- Number of sectors in the program
- Length of the program (in characters)

- Beginning address of the symbol table (for the debugging package).

This enables the calling program to branch indirectly to the address of the called program.

Below, an example is given of how a reference is made from one segment to another:

```
ROOT        |
            LDK    A7,n          Load segment number n
            LDKL   A8,LDAD       Load address
            LKM                  Monitor request to load segment n
            DATA   9
            ADK    A7,0          Test if request accepted:
            RF(4)  ERROR         No: branch to error routine
            CFI    A14,8+i,A8    Yes: Call entry point number i in
            |                    segment n
SEGMENT N   |
                1st data word    DATA Entry point 0
                                 DATA Entry point 1
                                 DATA Entry point 2
                                 |
                Entry point i    EQU   *
                                 |
                                 RTN   A14   return to calling program via
                                             register A14
```

CATALOGUED PROCEDURES

A catalogued procedure is a sequence of CCI control commands stored on disc under a procedure name. These commands are executed when the procedure is called from the input stream. It is possible to use or not use certain parameters of the commands in the procedure.

Catalogued procedures are kept in a special file named M:PROC, with file type UF. For each <userid> , one M:PROC file can be used. The M:PROC file may contain several procedures, each of which must begin with the procedure name (the first character of which must be $) and be terminated with an END command:

```
$PROC1
<commands of procedure 1>
END
$PROC2
<commands of procedure 2>
END
:EOF
```

As shown, the last procedure in the M:PROC file must be followed by an :EOF.

The first line of a procedure is the name used to identify the procedure call. The end of the procedure is indicated by the END command, so input commands will be read from the normal input stream.

Each user can update his M:PROC file without any difficulty. Adding new procedures is also done by updating the M:PROC file.

A call for a catalogued procedure is made by specifying the procedure name in the input command stream on the device with file code /EO. The Control Command Interpreter (CCI) first checks if it is the name of a standard CCI control command. If it is not, it assigns file code /DO to the file M:PROC of the current <userid> and starts looking for the name used to call the procedure. If it is found, the catalogued procedure is called. If it is not found, the CCI will assign file code /DO to the system M:PROC file and look for the procedure name there. Having found the procedure, the CCI will create a disc file with code /EE containing all the commands to be executed for the procedure.

Parameter Use

The use of parameters is allowed in the procedure body, but not in the first line (procedure name) nor in the END record. Comments are not allowed in these lines either.

In the catalogued procedure body, the parameter may appear in the first field (command name field) or in the second field (parameter field) of a line. The following forms of parameter specification are allowed:

- @⟨param⟩

  where ⟨param⟩ is a character string of up to 4 characters, identifying the parameter name. If this name is not specified after the name of the procedure called at execution time, no parameter will be used.

  For example, if procedure $PR1 is catalogued as:

```
$PR1
RDS
FRT /S,@NLST
LKE
RUN
END
```

  and is called as: $PR1
  it will cause compilation as: FRT /S
  but if called as: $PR1 NLST=NL
  compilation will be done as: FRT /S,NL

- @⟨param⟩ = default value

  where the default value will be taken if the parameter is not specified.

- @⟨param⟩ =

  where the whole line of the procedure will be ignored if the parameter is not specified.

  For example:

  A procedure $PR2 is defined as:

```
$PR2
@CM1=
LED @PNAM=
@CM2=FRT /S
INC @MOD=
LKE
RUN
END
```

  If this procedure is called as: $PR2 CM1=RDS
  the following commands will be executed:

```
                                    RDS
                                    FRT /S
                                    LKE
                                    RUN
```

If it is called as: `$PR2 CM1=RDS, CM2=ASM`

the procedure will be executed as:

```
RDS
ASM /S
LKE
RUN
```

If it is called as: `$PR2 PNAM=PRGRAM,MOD=MAIN`

The procedure will be executed as:

```
LED PRGRM
FRT /S
INC MAIN
LKE
RUN
```

Note:

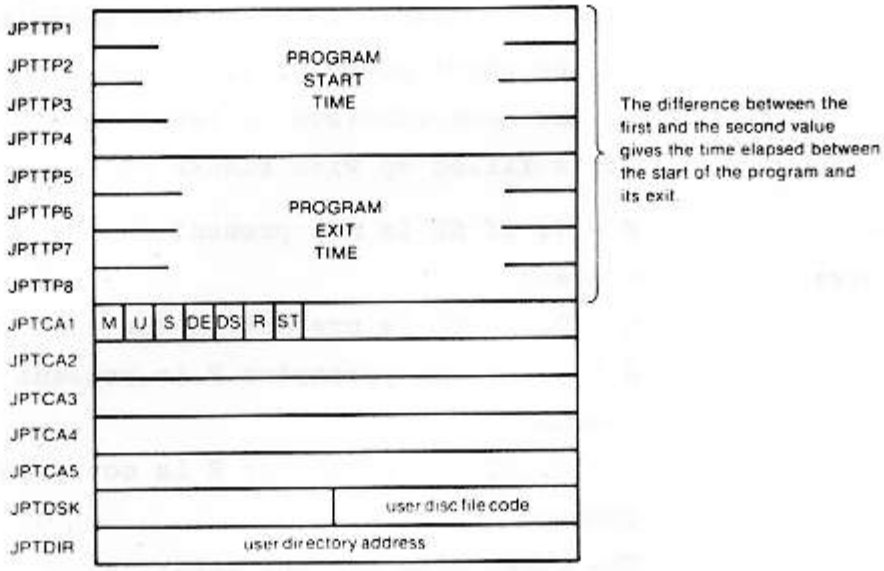
- all the parameters in the procedure call are keyword parameters, separated by commas.
- in the procedure body, parameters may be separated by spaces or commas: spaces are used to separate the command field from the parameter field; between parameters, only commas are allowed.
- in PSE and MES commands used in a catalogued procedure, no blanks are allowed within the message.
- Input commands for processors, e.g. LED are not allowed within a catalogued procedure but must be done on another file (e.g. /E0).
- when, from within a catalogued procedure, another catalogued procedure is called, no return is made to the first (calling) procedure.
  If the second (called) procedure is not found, an immediate return is made to the first procedure.
- the user can not use a catalogued procedure to open a new session, for the commands JOB and BYE implicitly cause the END of the procedure.
- the SCR command may not be used inside a catalogued procedure, except as SCR /S or SCR /0.

Error Messages:

ERROR IN PROCEDURE DEFINITION
ERROR IN PROCEDURE GENERATION (procedure is not correct)
PROCEDURE NOT CATALOGUED (procedure unknown in M:PROC file)
M:PROC NOT CATALOGUED
I/O ERROR

JOB PARAMETER TABLE

The Job Parameter Table consists of a 5-word communication area, followed by a certain number of words used by the system. The layout is as follows:



JPTCA1 to JPTCA5 are used by the system processors as follows and refer to the CCI commands ASM, LKE, FOR, FRT and their options:

Assembler: M = 0, if NL is present in the control command
M = 1, if NL is not present in the control command
The other bits are not meaningful.

Linkage Editor: M = 1, if the parameter M is present in the command
M = 0, if the parameter M is not present in the command.U and S are set as follows:

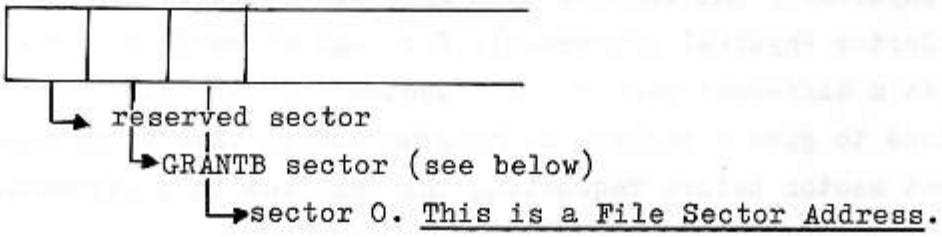
if N is present in the command, U = S = 0
if S is present in the command, U = 0 and S = 1
if U is present in the command, U = 1 and S = 0
if neither U nor S is present, U = S = 1
DE = 1, if the parameter DE is present in the command
DS = 1, if the parameter DS is present in the command
ST = 1, if a start address has been specified in the command
JPTCA2 contains the binary value of the common displacement.
If there is no common, this word is reset to zero.
JPTCA3 to JPTCA5 contains the entry point name of the start address, if it has been specified. If the name consists of fewer than 6 characters, it is filled up with blanks and left justified.

FORTRAN Compilers:

M = 1, if NL is not present in the control command
M = 0, if NL is present in the command
R = 1, if the parameter R is present in the command
R = 0, if the parameter R is not present in the command
The other bits are not used.

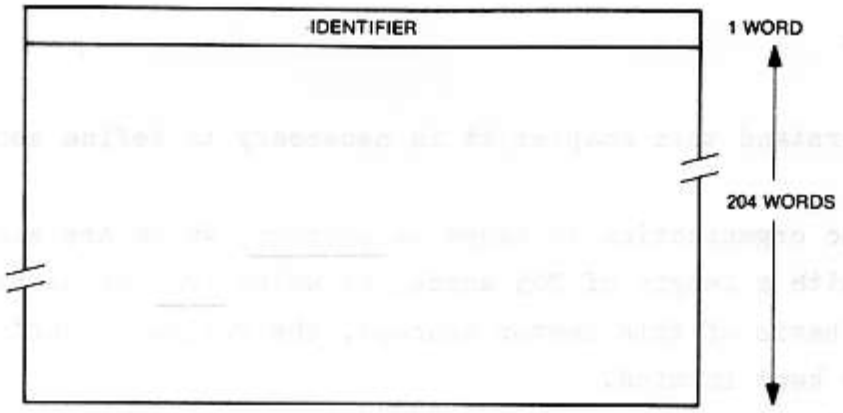
To understand this chapter it is necessary to define some concepts first.
The disc organization is based on sectors, which are sections of a track with a length of 205 words, of which 200 are usable.
On the basis of this sector concept, the following definitions must be kept in mind.

- Disc Sector Physical Address (DSPA):
  the physical address of a sector on the disc, i.e. the address of a sector in a consecutive arrangement.
- Disc Sector Logical Address (DSLA):
  the address of a sector on the disc as calculated by interlacing, which is ordering the sectors in such a way as to make access as efficient as possible (see below).
- File Sector Address (FSA):
  the address of a sector inside a logical file as managed by Disc File Management (DFM).
  Inside a file, the sector numbering starts at the third sector:

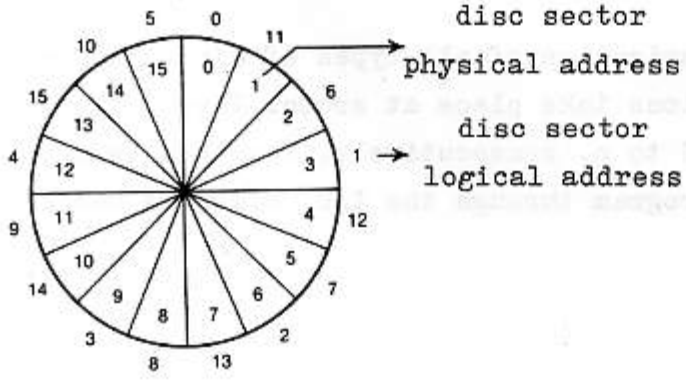


SECTORS

The basic unit in the organization of all types of disc used is a sector. All access operations take place at sector level. The sectors are numbered for 0 to n, consecutively. They can be accessed directly from any program through the I/O driver. A sector has the following format:



The identifier is written in the first word of every sector by the DISC PREMARK program. For moving head discs it contains the number of the cylinder in which this sector is located. For fixed head discs its value is not significant. The sector identifier is used by the system to check if a seek operation has been successful or not. The first word is set by the driver when a sector is written onto the disc. This word must not be modified during the I/O transfer, or a 'seek error' status may be returned later, when the same sector or another one in the same cylinder is accessed.
Although physically the sectors on a disc are numbered consecutively (Disc Sector Physical Addresses), for logical handling they are numbered in a different manner (disc sector logical addresses). This is done to give a requesting program enough time to process the current sector before requesting the next one in a sequential process.
For these logical sector addresses the sectors are interlaced, on a factor -3 basis for discs with 8 or 16 sectors per track.



16 sectors

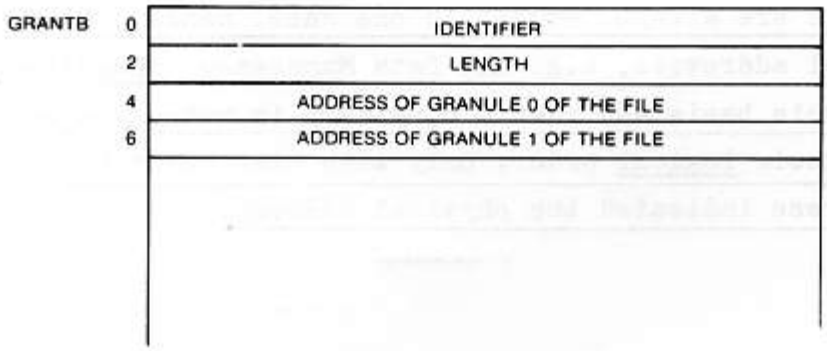
The disc sectors are always, except in one case, handled according to their logical addresses, e.g. all Data Management operations take place on this basis and when a disc dump is made, the sectors are dumped in their logical order. Only with disc error messages is the sector address indicated the physical address.

FILES

We have seen that the term file sector address takes into account that the first two sectors of a file are reserved: one for file header and one for a table called GRANTB, so that sector addressing starts with the third sector, which gets file sector address 0. Space allocation on the disc for a file is done on the basis of granules, where each granule is an area of 8 consecutive disc sector logical addresses, i.e. 8 logically consecutive sectors. A file is always stored on an integer number of granules, so one granule cannot be shared by two or more files. The system allocates as many granules as necessary to a file which is being written. These granules are chained and attached to the file code assigned to that file.
The addresses of the granules allocated to a file are kept in the table GRANTB in the second sector of the first granule of that file.
The sector GRANTB is initialized when a file code is assigned.
GRANTB is filled with the addresses of the allocated granules, any remaining words being set to zero.

For sequential access, when an attempt is made to write onto a granule which has not yet been allocated, a new one is allocated and GRANTB is updated. The system manages a table called BITAB which contains one bit for each granule on the disc. This table is copied into memory when a disc pack is loaded and updated in core and written back onto disc when a Keep File command is given. A bit is set to 1 when the corresponding granule has been allocated and it is 0 when the granule is still free. Allocation takes place via this table, after which the granule address is stored in GRANTB.

? If every 2nd word contains address
isn't the total 100 * 200 * 8 = 160K words ?

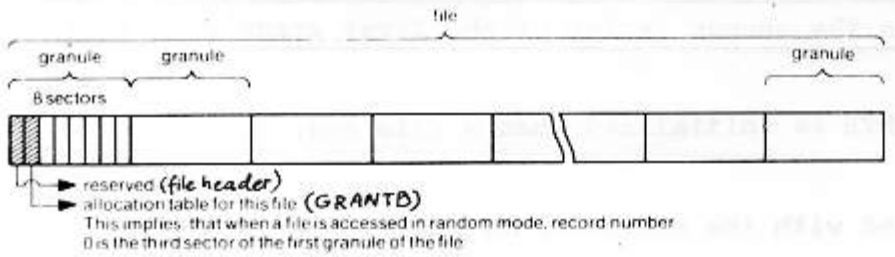


In this table, location 2 (i+2) is the address of the ith granule. If the granule has not yet been allocated, the location contains the value zero.

The granule address is a binary value from 0 to n, representing the relative sector address, from the beginning of the disc, of the first sector of the granule. GRANTB is 200 words long, so the file length is limited to 320k words (200x8 sectors).

Note: Random access files of more than 200 granules are possible, but when such a file is deleted under DOM, only the first 200 granules are deleted.

Finally, a file is organized as follows:



Inside the file, the useful area in a sector is 200 words (see Record Format).

At the start of a session with the DOM, allocation begins from the first granule available and new granules, if any, are added in ascending order (higher sector addresses). No backward search is done to take into account any granules which may have been deallocated again, e.g. after deleting a file. On the other hand, a specific command provides the possibility to start allocation at the first granule (SCR: Scratch control command). The alloca-

tion table BITAB stored on the disc is updated only when a file is made permanent (KPF control command) and when a file is deleted (DEL control command).
Allocation is done for temporary files only when they are written. So, a file which has been made permanent (KPF control command) cannot be extended directly. Updating a sequential file is done in the normal manner by copying it through the Update processor (Line Editor). Updating a permanent file in random access can only be done directly if no extra granule is required (see Data Management).

The first two granules on a disc are always reserved and used by the system for catalogues and libraries.

ACCESS MODES

Files can be accessed in two ways: random and sequential, each access mode requiring a different organization of the file.

Random

This type of organization has the following characteristics:

- records are of fixed length: one logical record=one physical record=one disc sector. = 200 words usable
- the records in the file may be organized in any manner. Access takes place according to the file sector address, i.e. by the relative position of the record in the file (not counting the first two sectors, which are the file header and GRANTB). The logical sequence of the records is not identical to the physical sequence. Such a file is a keyed file, the relative number of the record (sector) being the key.
- when a random file is created under DOM, one granule is allocated at a time, so extension of the file is possible as long as the file has not been made permanent (KPF control command).
- records are accessed directly by specifying the file sector address.
- records may be retrieved, updated and restored individually.
- random access files should be Assigned, Kept and Deleted under DRTM, not under DOM. See Note on page 32.

Sequential

This type of organization has the following characteristics:

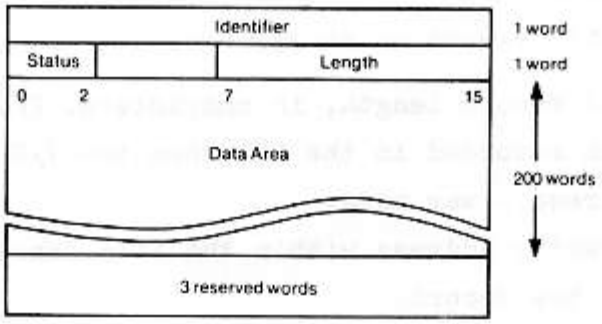
- records may be of any length, up to 16k words. The sector format of such records is described below.
- the only relation between the records in a sequential file is their sequence. The records of such a file must be presented in the order in which they must be written onto the disc, i.e. the

logical sequence of the records is identical to the physical sequence.
- to retrieve a record, the whole file must be scanned up to the desired record.
- to update a record, the file must be processed as a whole.

Additional details on Access Mode are given in chapter 7, Data Management.

Record Formats

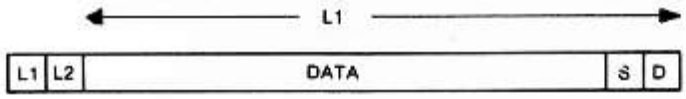
The format of physical record (sector) has already been described (see SECTORS). This is the record format for random files.
In sequential files a sector may contain a logical record or it may be part of a logical record. Sectors have the following format in these files:



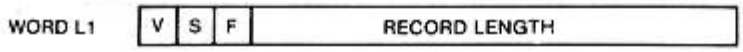
- status consists of 3 bits:
  bit 0: if 1, this sector has been deleted
  bit 1: if 1, an EOS (End-of-Segment) record has been written in this sector. This record is the last one in the sector, for the first record following an EOS always starts at a new sector.
  bit 2: if 1, EOF (End-of-File) record has been written in this sector. An EOF record requires a full sector without any other records in it. So when an EOF is written for a file, first the current sector buffer is written, if necessary, and then an additional sector for the EOF record.
- Length: specifies the length, in characters, of the used area of this sector.

- the data area contains data written in either sequential or random access mode.
- the last 3 words are not used.

The logical records in sequential access files are always compressed and blocked by the system to save disc space (trailing blanks are removed). The format of these logical records is as follows:



L1 is the length of the record, including the 2 words S and D, but excluding L1 and L2:



V = 1: this record has been deleted from the file
S = 1: this record is an EOS record
F = 1: this record is an EOF record

L2 is the initial record length, in characters. This is the requested length recorded in the ECB when the I/O request for writing this record was made.

S is the file sector address within the file containing the first word of the record.

D is the displacement in characters in the sector S, of the first word belonging to the record.
Data is up to 1600 words long (one granule), trailing blanks removed.

Special Records

There are some special records in sequential files, which have the following format (cf. Record Format) (values in hexadecimal):

:EOS: 4004 - 0 - S - D
(4004 = L1)

:EOF: 2004 - 0 - S - 4
(2004 = L1, 4 = D)

Blank card: 0004 - 0050 - S - D
(0004 = L1, 0050 = L2) (hexadecimal 50=decimal 80=card length)

Note: Records on disc always consist of an even number of characters. So, whatever the value of L2 given by the user at creation time, L1 always represents an even number of characters, because when the requested length is an odd value, a dummy character is added to the record. An EOS is always stored in one sector and must be the last record in the sector.
An EOF is always written in a separate sector.

FILE TYPES

Under the DOM, the following four file types can be distinguished:

- Source file (SC)

  Source files are sequential files input in source language or after an update of the source language. These files are used as input to one of the language processors.

- Object Files (OB)

  An object file is a sequential file with one record per object cluster. Each object module is followed by an EOS record. A new object module starts at a new sector. The final object module in an object file is followed by both an EOS and an EOF record.
  As an object module is not a file, it need not necessarily be stored on an integer number of granules. Therefore deletion of an object module need not result in deallocation of a granule. However, in such cases a flag is set in the second word of every sector of the deleted module (See Sector Format: Status). When an object file is read sequentially, the Data Management routines will automatically skip any deleted sectors.

- Load Files (LM)

  A load file is accessed in random mode. Each full sector of such a file contains 188 code words and 12 control words for relocation bits (see Linkage Editor).
  The first four words of a load file contain the following information:

  - start address of the load module
  - number of sectors in the load module
  - memory length required
  - address of entry points table (for debugging functions).

- Undefined Files (UF)

  This type comprises all other files, such as data files.

Load Module Size

All system or user programs must be kept on disc in the load format generated by the Linkage Editor or at system generation.
The number of granules required for keeping a program is
((S-1)/188+2)/8+1
S being the program size in words. Thus, the following table gives the maximum program size for a number of granules:

Number of Granules	Maximum Program Size (Words)
1	1128
2	2632
3	4136
4	5640
5	7144
6	8648
7	10152
8	11656
9	13160
10	14664
11	16168
12	17672

DISC STRUCTURE

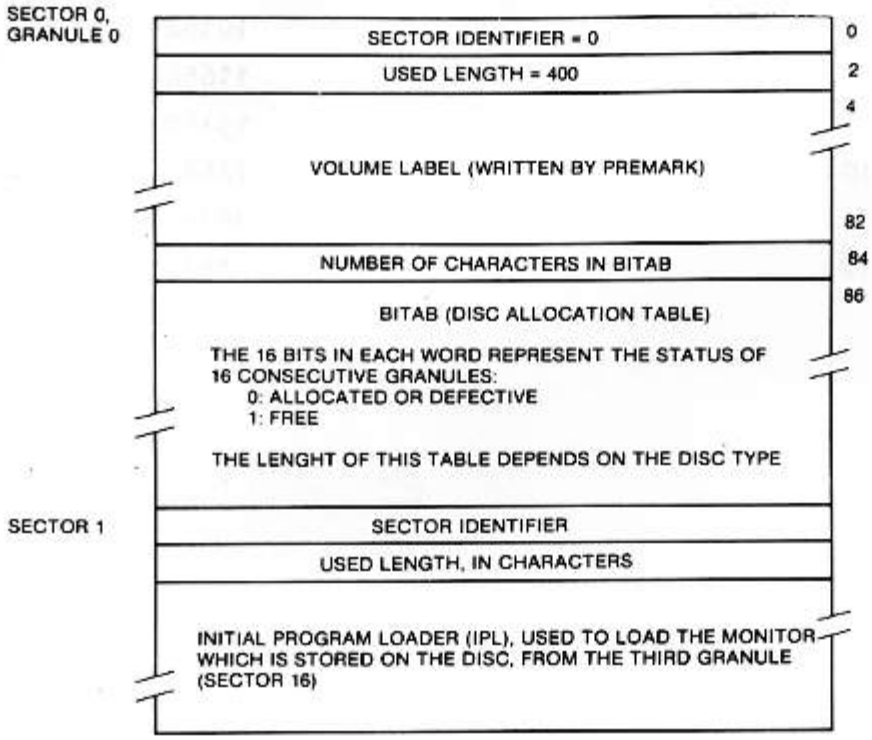
Catalogue and Library Structure

On a disc, the Catalogue contains one entry for each user identification declared on that disc. Each of these entries contains a pointer to a library, one for each user in the Catalogue. Each user library consists of a directory and the files in the library.

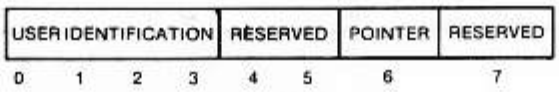
Catalogue Structure

The first granule on a disc contains a catalogue of the users of this disc. Its layout is as follows:

- Sector 0: Volume label and disc allocation table (see Premark: Appendix B).
- Sector 1: IPL (Initial Program Loader).
- Sectors 2 to 7: Catalogue.

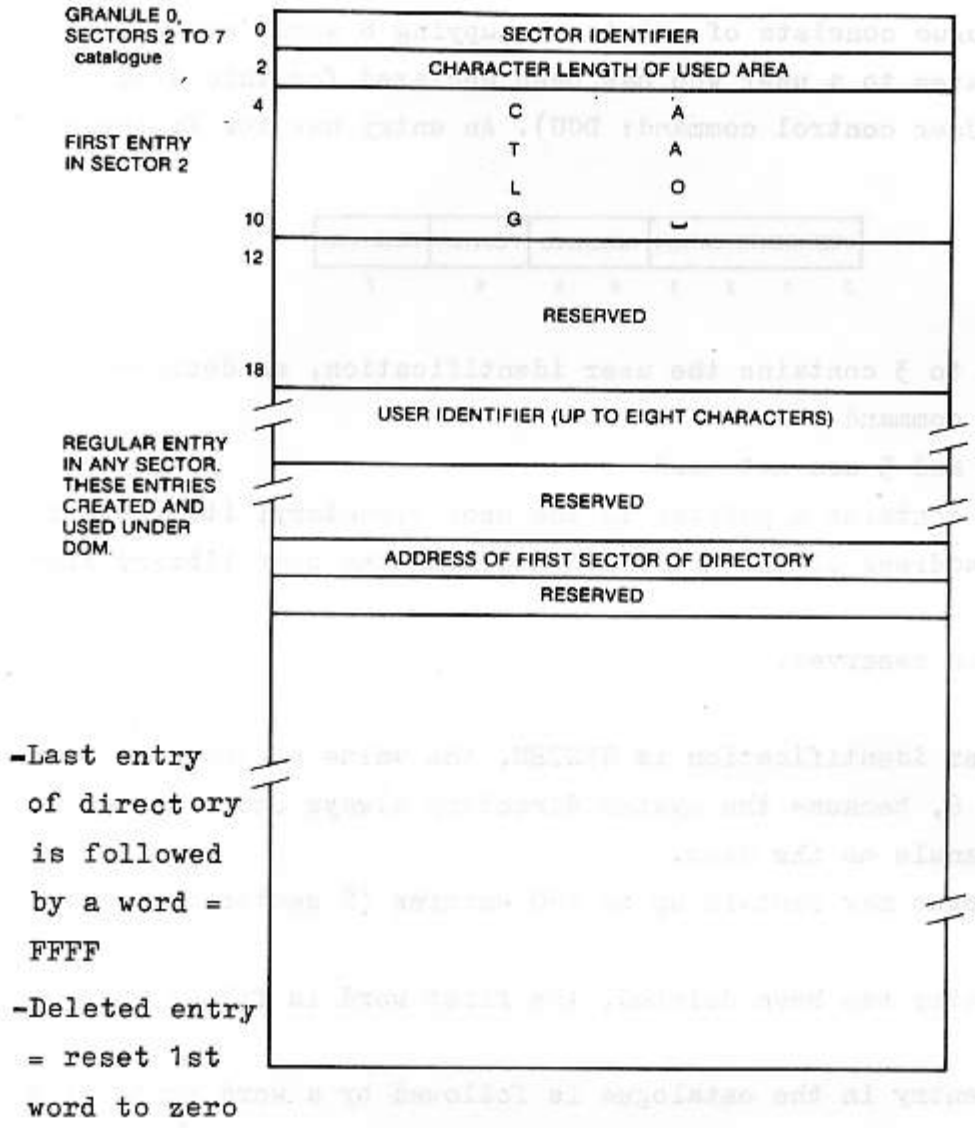


The Catalogue consists of entries occupying 8 words each; each entry relates to a user who has been declared for this disc (Declare User control command: DCU). An entry has the following format:



- words 0 to 3 contains the user identification, as declared in the DCU command
- words 4 and 5 are not used
- words 6 contains a pointer to the user directory; it is the disc sector address of the granule containing the user library directory.
- word 7 is reserved.

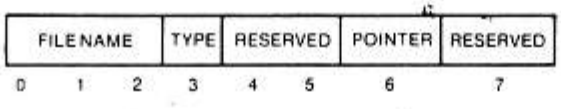
If the user identification is SYSTEM, the value of the pointer in word 6 is 8, because the system directory always occupies the second granule on the disc.
The Catalogue may contain up to 150 entries (6 sectors, 25 entries each).
When an entry has been deleted, the first word is filled with /0000.
The last entry in the catalogue is followed by a word containing /FFFF.



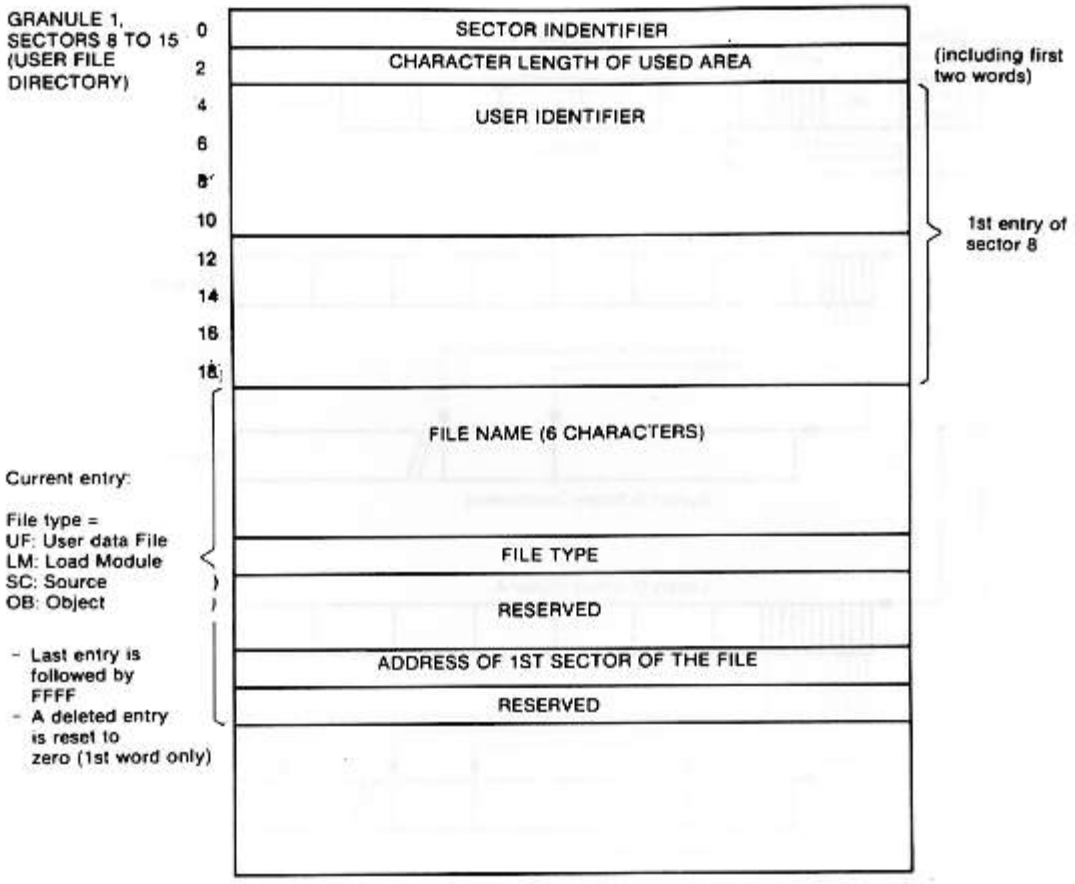
Format of user catalogue (sectors 2 to 7) -

Directory Structure

Each user is provided with his own directory and library. The directory occupies one granule and contains the names of and pointers to the user's files. The granule containing the user directory may be located anywhere on the disc, except when the user is SYSTEM. In this case it is the second granule on the disc.
Each entry in a directory consists of 8 words:



- words 0, 1 and 2 contain the file name
- word 3 contains the file type, which may be one of the following:
  . for source files: SC
  . for object files: OB
  . for load files: LM
  . for undefined files: UF
- word 6 is the file pointer; it contains the disc sector address of the first granule of the file
- words 4, 5 and 7 are reserved.



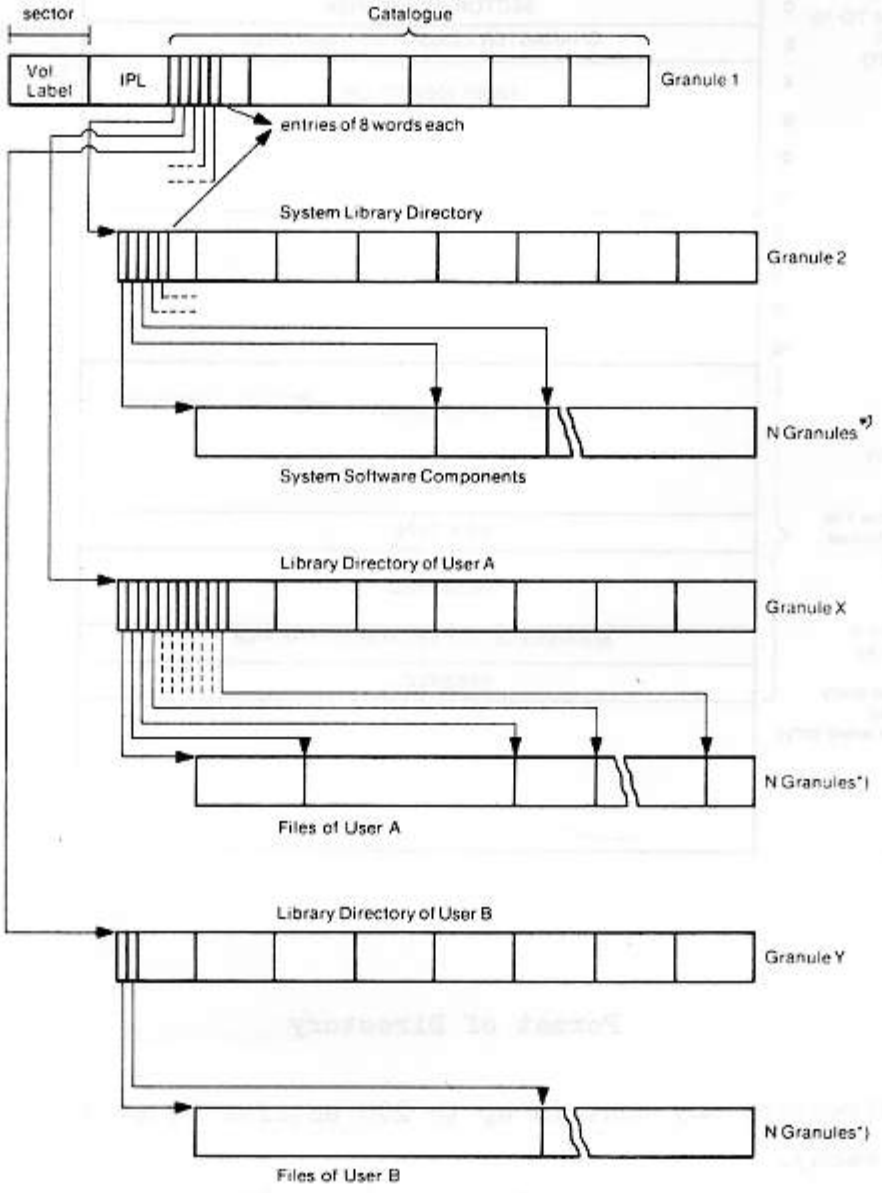
Format of Directory

A user directory may contain up to 200 entries (8 sectors of 25 entries each).
Each entry points to a granule table (GRANTB), containing the successive addresses of the granules allocated to the file represented by the entry in the directory. GRANTB may contain up to 200 granule addresses.

When an entry has been deleted, the first word is filled with the value /0000.
The last entry in a directory is followed by a word containing the value /FFFF.

The granule for the user directory is allocated at the time when a new user is declared to the system (DCU command) and entered in the Catalogue.

Below a drawing is shown of the DOM library structure.



*) These granules are not necessarily adjacent.

Example of the Structure of Catalogue and Libraries

DOM System Disc Structure

The System Disc contains all the system software, i.e. the monitor, the system processors and the system object library. These software components are on the disc in the same manner as the files of the different users are, i.e. in a user library coupled to a user identification and a directory. For the system software the user identification SYSTEM is used. It is necessary that the first file in the library of the 'user' SYSTEM is the monitor and that all the granules it occupies are adjacent. The format of the monitor is the same as that of load modules and it may be catalogued under name and type.

The other SYSTEM software components must be present in the directory of this first user identified by SYSTEM, but the granules they occupy need not necessarily be adjacent. Thus the first granules of the System Disc are occupied as follows:

Granule 0: Sector 0: Volume Label
Sector 1: IPL=Initial Program Loader
Sectors 2 to 7: Volume Catalogue

Granule 1: Directory of the first user, i.e. SYSTEM

Granules 2 to n consecutively contain the Monitor.

The other software components use granules randomly.

Any granules remaining on the System Disc can be used by other users for their files.

All I/O operations are initiated by an I/O monitor request. At system generation time, the necessary tables for fulfilling this request must have been filled and the necessary modules loaded. When the request is given, with an LKM instruction, register A7 must have been loaded with parameters about the particular type of I/O function, while register A8 must contain the address of an Event Control Block which holds the necessary information about the data to be transferred.

There are several types of I/O request (as specified in A7):

- Random I/O requests: for random access I/O operations on disc devices.
- Basic I/O requests: for these requests the monitor will not do any character checking or data conversion, so they are used in case of binary I/O. The monitor handles only the control command initialization and signals the end of the I/O operation.
- Standard ASCII I/O requests: these requests provide more monitor facilities, such as error control characters, data conversion from external code to internal ASCII code and vice versa, character checking for end of data. Characters are stored 8 by 8 bits, two to a word.

Moreover, a number of control functions can be performed through a monitor request, such as writing EOS or EOF records, skipping forward or backwards, rewinding, etc.

In the Event Control Block (ECB), pointed to by register A8, the user specifies the file code (see below) of the device or file concerned with the I/O operation, and additional parameters such as buffer address and buffer length. At the end of the I/O operation, the monitor places information about the result of the I/O operation in this ECB, so that it may be verified by the user program.

For non-disc devices, I/O operations are done at record level, by I/O drivers running at level 48. No blocking- deblocking is performed. For disc devices, the user can use an I/O driver or he can access the disc through the Data Management package. If he uses the driver, he must specify an absolute sector address for the I/O operation. With Data Management, he specifies the relative sector address for direct access and Data Management will find the correct sector, Moreover, Data Management automatically provides additional functions, such as blocking-deblocking.
In a following chapter detailed information will be given about the I/O monitor requests.

DOM LOGICAL FILES AND UNITS

To facilitate use of the system, the control command language allows for implicit addressing of some system logical units and for calling some system temporary files by a predefined name.

Temporary Source File: /S

At creation time, a source file is always temporary. The monitor allocates the necessary granules to this file on the disc which contains the library of the current user.
A source file is sequential.
The disc temporary source file /S can receive a file read from the source input unit or a file which is the result of an updating process done for a library source file. The /S file may then be used as input to one of the language processors.
It is possible to have this file listed, printed or punched (LST, PRT and PCH control commands) or the file may be made permanent by keeping it in a library (KPF control command) for later use.

Temporary Object File: /O

An object file, i.e. an output file of one of the language processors, is always temporary at creation time. Disc space is allocated to it when it is being produced, in the same way as for tem-

porary source files, i.e. by the monitor. The temporary object file /O receives the object modules read from the object input unit, or object modules produced by one of the language processors or selected from the object files of user libraries.
The /O file is sequential.
The /O temporary object file is used as the main input file for the Linkage Editor.
By means of the control command POB it is possible to have this file punched on tape and by means of the control command KPF all or part of its object modules may be placed in a library.

Temporary Load File: /L

The output of a link-edit operation is the temporary load file /L. This file is random file. Disc space is allocated to it by the monitor as for the other temporary files. This file may be executed by means of the control command RUN or it may be stored in a library (KPF control command). It is also possible to have the /L file punched out (PLD command), but only for LKM output, not for OLE output. In this case the format is converted to standard object cluster format. Such a module cannot be directly introduced from the object input unit to the /L file. It first has to be put on the /O file and be link-edited.

Source Input Unit

As defined at system generation time, the source input unit may be:

- card reader
- punched tape reader
- ASR punched tape reader
- cassette tape unit
- magnetic tape unit.or disc, i.e. any input device.

By means of a specific control command (RDS) source programs can be read from this unit and be copied onto disc as a temporary file ready for assembly or compilation, or they may be copied into a library and made permanent (RDS, followed by KPF control command). There is no restriction in addressing the unit from the user program.

Object Input Unit

This unit is also defined at system generation time, and may be one of the following:

- disc
- punched tape reader
- ASR punched tape reader
- cassette tape unit
- magnetic tape unit.

By means of the command RDO, an object module file on punched tape can be read from this unit, to be copied onto disc as a temporary file ready to be link-edited or it may be copied into a library and made permanent for later use (RDO, followed by KPF control command).
Because no object code is punched on cards, this unit may not be a card reader.

Command Input Unit

This is the unit on which the control commands are entered. As defined at system generation time it normally is the operator's typewriter, but it may be any input device.

Print Unit

If a line printer is included in the configuration, this may be defined as the print unit at system generation time, otherwise it will be the print equipment of the operator's typewriter.

Punch Unit

One of the following two may be defined as the punch unit at system generation time:

- tape punch
- ASR tape punch
- cassette tape unit
- magnetic tape unit
- disc, i.e. any output device.

Note:
During a session, new assignments may be defined for different units, but when a system is loaded the assignments are the ones defined at system generation time.

FILE CODES

The following file codes are standard assignments for the logical files and units specified, as incorporated in the Disc Operating Monitor.
Depending on the configuration, different logical units may have the same physical assignment:

01:	user typewriter (answers from CCI S:); used by user program)
02:	print unit
03:	punch unit (output)
04-09:	reserved for peripheral devices
D0:	catalogued procedure input
D4:	/S file or library source file (Line Editor output)
D5:	/O file (ASM output, LKE input)
D6:	/L file (LKE output)
D7:	system object file (library)
D8:	user object file (library)

D3 and D9 to DF are reserved for system use.

E0:	control command input
E1:	source input
E2:	object input
EE:	catalogued procedure output
EF:	system operator's typewriter (system output in response to INT button: M:)

F0 to FF are logical addresses of disc units. These are reserved for system use, and are not to be used by the user.

The file codes 01,02,03,E0,E1,E2 and EF can be used without having been assigned in a previous ASG control command.
The file codes 04 to CF can be used by the user to address his own files and any additional peripheral devices in his configuration, but only after he has assigned these file codes through ASG commands, or, for 04 to 09, at SYSGEN time.
File codes 0A to DF are scratched when an SCR or BYE command (see chapter 9) is given.

ACCESS MODES

Two access modes are allowed for disc files: random and sequential.

- Random access is possible only for fixed length records of 200 words (one sector).
  A record is accessed by means of the record number (file sector address).
- Sequential access is possible for records of variable length of up to 3200 characters (1600 words = 8 sectors = 1 granule).
  The interface for sequential access in read or write mode is the same as for punched tape, cassette tape or magnetic tape, the blocking-deblocking function and blocking buffer allocation being handled by the system.

A file written in sequential mode can be accessed in random mode.
For further details see Data Management in Chapter 7.

Data Management consists of a set of routines to help the user transfer his records between the memory and the peripheral devices, to help him create files of a particular type and retrieve records from these files. The routines are selected and included in the monitor at system generation time.
Data Management is memory resident and runs at level 49. This implies, that a request coming from a program at level 49 can be processed only when this program, or any others connected to level 49, have given a Wait monitor request.
All operations on the peripheral devices take place through file codes, so the user need not know the type and physical address of each device. The system will find this out by translating these file codes with the aid of monitor tables.
There are two types of Data Management, i.e. two ways of writing or reading files:
Sequential Access Method and Direct Access Method.

The user creates a file by assigning a file code to a temporary disc file (ASG control command) and writing onto it by running a program.
Under the Disc Operating Monitor, if the user wants to make the file permanent for later use, he must do so by specifying a KPF (Keep File) control command at the exit of the program, otherwise this file will be scratched at the end of the session or when a scratch command is used. The file is then catalogued and may be consulted any time, after a file code has been assigned to it. Once the file has been made permanent, however, the number of sectors it occupies cannot be changed, because granule allocation is done only when the temporary file is created.
All Data Management operations, i.e. reading and writing a record, writing EOS or EOF, etc., are done by I/O monitor requests in the program for each record, in which the user can specify the access method and the data management function. It is not necessary to call special routines. The mode of access is determined by the first request for a whole run.

At system generation time the user has to define the number of buffers and their lengths for use by the Data Management package. In general 2, or at most 3, buffers of one sector length will suffice. These will then be included in the system to be allocated automatically.

The first sector on each disc contains a disc allocation table (BITAB) in which the status of each granule, free or allocated, is recorded. See page 31
The second sector of each file contains a granule table (GRANTB) with the addresses of all the granules of this file. See page 31
Four types of file exist:
- source (SC): only sequential access possible
- object (OB): sequential and direct access possible
- load module (LM): only direct access possible
- undefined (UF): sequential and direct access possible.

SEQUENTIAL ACCESS METHOD

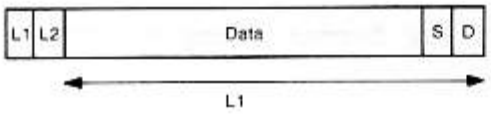
A file is sequential when the only relation between the different records is their sequence. When such a file is created, the records must be presented in the same order in which they must be written onto the disc. To access the file, it must be scanned sequentially until the desired record is found.
The logical sequence in a sequential file is identical to the physical sequence of the records in the file.

Record Structure

User records may contain up to 3200 characters. This implies that a record may be part of a sector or that it may occupy a number of sectors. When these records are written onto disc they are first blocked into a buffer.

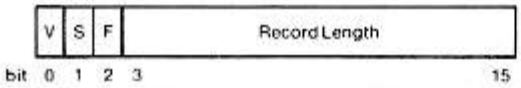
Logical Record Format

In order to save disc space, the system compresses and blocks the logical records used in sequential access; trailing blanks are removed. The format of a record is as follows then:



where:

L1 is the record length, including the words S and D, but not including L1 and L2:

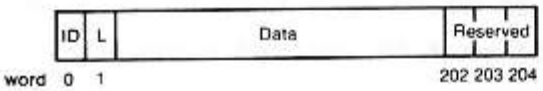


V = not used
S = 1, if the current record is a segment mark (EOS)
F = 1, if the current record is a file mark (EOF)

L2 = the initial record length in characters, as specified in the user's ECB (word 2) in Write mode
S = file sector address of the first word of the record.
D = the displacement in sector S of the first word belonging to the record (number of characters).

Sector Format

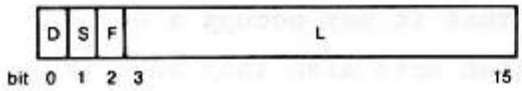
The format of a sector is as follows:



where:

ID is the cylinder identification: a number from 0 to 202, used to check seek operations on moving head discs.

L is the length of area used for data in the sector (0 to 400 characters)



where:

D = 1, if the sector has been deleted from the file.

S = 1, if the current sector contains a segment mark (EOS).

F = 1, if the current sector contains a file mark (EOF).

Special records

Some special records have the following format (compare with - Logical Record Format):

:EOS: 4004 - 0 - S - D
(4004 = L1)

:EOF: 2004 - 0 -S - 4
(2004 = L1; 4 = D)

: Blank card: 0004 - 0050 - S - D
(0004 = L1; 0050 = L2 (hexadecimal 50 = decimal 80 = card length).

Note:

Records on disc always consist of an even number of characters. So, whatever the value of L2 given by the user at creation time, L1 always represents an even number of characters, because when the requested length is an odd value, a dummy character is added to the record.

An EOS is always stored in one sector and must be the last record in the sector.
An EOF is always written in a separate sector.

File Creation and Processing

A sequential disc file is created by the program delivering the logical records with the aid of the Data Management Package. Each record is written by an I/O request up to 'Write EOF'. When this request is encountered the contents of the last blocking buffer are output to the disc and an EOF record is written in the next sector.
When a request is given to write an EOS, the current sector is terminated with an EOS record and the following record will be written at the beginning of the next sector. Before creating a sequential file the user must assign a file code to a temporary disc file. If the user wants to have the file catalogued, he must give a KPF control command before closing the session. Before reading such a catalogued file or writing onto it, an assign command has to be given for it.

Updating

If a user wants to update a sequential file he must first read it, then update it and finally write the updated file on another temporary disc file. Such an updated file can be made permanent in the same way as the original file and under the same name, by means of the KPF control command. If the user gives a new name to the updated file, the original file is not destroyed, which enables him to have several versions of one program belonging to the same Library.

Read a Record

To get a logical record from an input sequential disc file, the user may give a Read monitor request (LKM 1), as for any other device. The system automatically provides the disc buffer, fills it, deblocks the records and recovers any errors. Only the sign-

ificant part of the record will be stored in the record area specified by the user in his ECB, i.e. control words will be removed by the system.
When an EOS or EOF mark is encountered, this is indicated to the user in the Status word of the ECB (word 4). An attempt to read records beyond an EOF mark will cause an EOF status to be returned to the user.

Write a Record

To put a record on an output file may be done by means of a Write monitor request (LKM 1), as for any other device. When the record is moved to the disc buffer it will be formatted with control words, as it consists only of data words in the user area. The system will also take care of buffer allocation, record blocking and error recovery. For temporary sequential files, records can be written onto a file until the maximum number of granules has been allocated (200). After this, an 'End of Medium' Status will be returned in the user's ECB if an attempt is made to write over the number of granules already allocated.

Opening a File

Opening a file need not be requested by the user as this is implicit in the first read or write request.
When the ASG control command is given an entry is made in a file code table and a Logical File Table is built by the system to record information about the file used. This, however, does not imply that the file has been opened yet.

Closing a File

Closing a file is done in write mode, after the last record of a file has been written onto the disc, by giving a Write EOF monitor request. If the user wants to write the contents of the last buffer onto a disc without closing the file, he should give a Write EOS request. This is the case with the common object file created by the language processors, which do not have to close the /O file

as this is done by the system when the Linkage Editor is called:

Positioning a File

It is possible for the user only to position the file at the first logical record. This is done by giving an I/O request with the order to rewind the file.

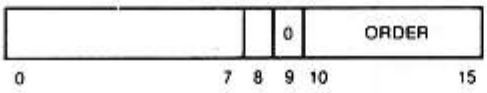
Data Management Requests

The Data Management package is activated by an I/O monitor request (LKM 1). The function which has to be performed is loaded into the A7 register, while the A8 register must be loaded with the address of an Event Control Block, containing the necessary parameters. The calling sequence is as follows:

```
LDK    A7,CODE
LDKL   A8,ECB
LKM
DATA 1
```

where:

the word CODE is made up as follows:



bit 8 = 1: wait for completion of the event is implicit in the request.

= 0: control will be returned to the calling program after initialization of the operation. To check for completion the program must give a Wait monitor request (LKM 2).

bit 9 is not used, and must be reset to zero. (If it is 1, the status /C010 will be returned).

ORDER contains the function code:

/01,/02: Read a Record (Basic, Standard)

/05: Write a Record (Basic)

/06: Write a Record (Standard)
/07: Write a Record (Object, 4+4+4+4 tape format)
/08: Write a Record (Object, 8+8 tape format)
/22: Write EOF mark (Close a file)
/26: Write EOS mark
/30: Get information about a file code
/31: Rewind the file.

Notes:

When the order /30 is given, the user must specify the file in ECB word 0; the ASCII characters 'DK' (physical files F0 to FF) or 'DL' (logical files) will be returned in ECB word 1;
the other words will be reset to zero, because disc file codes are handled by the system.

Basic orders (01, 05) and Object Write orders (07, 08) are converted to Standard Read/Write for disc files.

The standard ECB, to which register A8 points, has the following layout:

	0 — 7	8 — 15	
ECB 0	Event Character	File Code	Y/X
ECB 1	Record Area Address		X
ECB 2	Requested Length		X
ECB 3	Effective Length		Y
ECB 4	Status		Y
ECB 5	Not Used		X

The words marked X must be filled by the user.
Those marked Y must be reserved by the user, but will be filled by the system.

ECB 0: Event Character: Bit 0 is set to 1 on completion of the I/O operation. The other bits are not used and reset to zero.
File Code: Defines the logical reference to the file.
ECB 1: Specifies the beginning address of the area where the record is stored in memory.

ECB 2: Length in characters of the record area. (Words for basic read on cards).

ECB 3: Number of characters which has actually been moved from or to the record area. (Words for basic read on cards).

ECB 4: This word contains the status returned to the user program by Data Management. See page 127 . In addition, status /10 is returned when an attempt is made to write beyond the last allocated granule of a file (End of Medium) and for temporary files, when over 200 granules are written.

ECB 5: is not used with sequential access method.

DIRECT ACCESS METHOD

When the direct access method is chosen, the records within a file may be organized in any manner and accessing a record may be done at random, by specifying a file sector address from 0 to 1597. This is possible because with this method a logical record is equivalent to a physical record on the disc: one sector. When a direct access file is created, the records may be delivered in any order. The system will create a granule table and allocate granules to the records (sectors) that are delivered. Each granule address is noted in the table, and when the user wants to read a particular record, he specifies the file sector address, upon which the system will be able to find it with the aid of this granule table. Thus, such a direct access file may be considered a keyed file, where the relative number of the record (=sector) is the key. Although the great advantage of a direct access file is that the user can read, write or update individual records without having to scan or copy a whole file sequentially, he may nonetheless want to be able to access such a file sequentially. If this is the case, the individual records must be formatted in the same way as for a sequential file, i.e. the sector format must be the same and the records must be written sequentially and terminated with a 'WRITE EOF' request.

Record Structure

With direct access, a logical record is the same as a physical record: a record is equal to a sector.
The first word contains the cylinder identification (used by the disc driver to check the position of the head after a seek operation). The remaining 204 words may be used for data storage.

File Creation and Processing

A file is created when an Assign command or monitor request is given. This reserves the required number of granules on the disc where the records can then be written. Such a file can be made

permanent by means of a Keep File control command.
When a request is made, each record is transferred directly from the user area to the disc.
Only one granule is allocated initially, and the file is extended granule by granule during its creation.
Random access files do not have to be closed by the user.

File Retrieval

Here lies the main advantage of a direct access file, because once the file has been assigned, any sector record can be retrieved, erased or updated and rewritten individually.
The size of a file is fixed at creation time, when a granule table is also written with an entry for each granule of the file. When the file is catalogued by means of a KPF command, creation is terminated and no more granules can be added. Therefore the user must know the maximum size of the file at creation time (no more than 1598 sectors)

Read a Sector

This is done in the same way as for sequential access, the only difference being that the user must specify in ECB5 the relative number of the sector within the file (i.e. the file sector address, a number from 0 to 1597), and specify / A for the read order in register A7. Moreover, he must supply the system with a 205-word buffer in which the physical record will be stored. As mentioned above, the first word contains the cylinder identification.

Write a Sector

This is done in the same way as for sequential files, the only difference being that the user must specify in ECB5 the relative number of the sector to be written into the file (a number from 0 to 1597) and specify / B for the write order in register A7.

Moreover, he must supply the disc buffer in which the information is stored: a 205-word buffer of which the first word will be replaced by the cylinder identification by the system (required by the physical disc I/O driver).

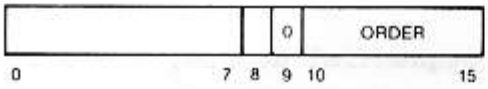
Data Management Requests

The Data Management package is activated by an I/O monitor request (LKM 1). The function which has to be performed is loaded into the A7 register, while the A8 register must be loaded with the address of an Event Control Block, containing the necessary parameters. The calling sequence is as follows:

```
LKD     A7,CODE
LDKL    A8,ECB
LKM
DATA 1
```

where:

the word CODE is made up as follows:



bit 8 = 1: wait for completion of the I/O operation is implicit in the request.

= 0: control will be returned to the calling program after initialization of the I/O operation. To check for completion, the program must give a Wait monitor request (LKM 2).

bit 9 is not used and must be reset to zero. (If it is 1, the status /C010 will be returned).

ORDER contains the function code:

/0A: Read (Random)

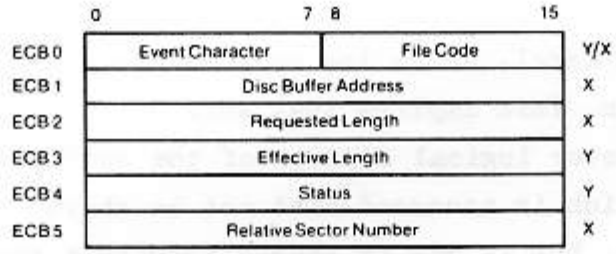
/0B: Write (Random)

/30: Get information about a file code.

Note: When the order /30 is given, the user must specify the file code in ECB word 0; the ASII characters 'DL' will be returned in ECB word 1.

The other words will be reset to zero, because disc file codes are handled by the system.

The standard ECB, pointed to by register A8, has the following layout:



The words marked X must be filled by the user.
Those marked Y must be reserved by the user, but will be filled by the system.

ECB0: Event Character: Bit 0 is set to 1 on completion of the I/O operation. The other bits are not used and reset to zero.
File Code: Defines the logical reference to the file.

ECB1: Specifies the beginning address of the 205-word disc buffer.

ECB2: Whatever the value of the requested length, 205 words will be written on the disc or read into memory.

ECB3: This word is not affected.

ECB4: This word contains the status returned to the user program by Data Management when the sector transfer is terminated. See page 127 . In addition, status /10 is returned when an attempt is made to write beyond the last allocated granule of a catalogued file and, for catalogued files, when over 200 granules are written.

ECB5: Specifies the relative position of the sector within the file, i.e. the file sector address.

Physical Disc Access

This is a direct access on a physical sector level. The Data Management module is not used for this type of access, but special orders are used in the monitor request (LKM1):

/11=physical read
/15=physical write

Moreover, the file code in the ECB must be one of the disc unit file codes /F0 to /FF.
Access is done at sector level, where the sectors are numbered consecutively from 0 to n. This implies that word 5 in the ECB must contain the disc sector logical address of the sector which is to be accessed. The disc which is accessed must not be shared by the user and Data Management, but it may be shared between a number of other programs doing direct physical access. An example of this type of access is 'Dump Disc'.

The initial loading procedure is very simple:
The bootstrap is loaded, either through the toggle switches or by pushing the IPL button on the control panel, then the Initial Program Loader (IPL) is loaded into memory, followed by the monitor.

LOADING BOOTSTRAP AND IPL

The bootstrap can be loaded in one of two ways, depending on whether the optional ROM bootstrap is included in the system or not:

If not, the procedure is as follows:

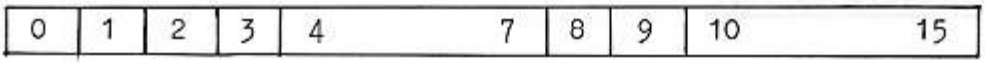
- switch on the CPU
- load the bootstrap into the first 64 memory locations manually, by means of the toggle switches and the Load Memory button on the control panel. The 64 bootstrap values can be found in Appendix E. Then check by reading these locations out.
- set up the device parameters on the toggle switches, as shown below, and load this value into the A15 register.
- put the disc containing IPL and monitor into the disc drive, push the START button on the drive and wait till the READY button lights
- push the MC button
- load 0 into the A0 register
- push the RUN button on the CPU control panel
- the IPL is now loaded into memory and it loads, in turn, the monitor, after which the monitor initialization phase is started with the typing out of the monitor identification.

If the ROM bootstrap is included in the CPU, which is highly recommended, the procedure is much simpler:

- switch on the CPU
- put the disc containing IPL and monitor into the disc drive, push the START button on the disc unit and wait for the READY button to light

- set up the device parameters on the toggle switches on the CPU, as shown below, and load this value into the A15 register
- push the IPL button on the control panel. This loads the bootstrap into memory, which immediately loads the IPL from disc. The IPL then loads the monitor and the monitor initialization phase is started with the typing out of the monitor identification.

The device parameters on the data switches must be set as follows:



where:

bit 0 = 0: tape format used is 8+8
= 1: tape format used is 4x4

bit 1 = 0: the device used is not a disc
= 1: the device used is a disc

bit 2 is used only if bit 1 = 1:
= 0: the disc used is a fixed-head disc
= 1: the disc used is a moving-head disc

bit 3 = 0: the device is connected to the I/O processor
= 1: the device is connected to the programmed channel

bits 4 to 7 are used to qualify the CIO Start command sent by the bootstrap and are transferred to the addressed control unit on lines BIO 12 to 15 when required

bit 8 = 0: the control unit involved is a single device control unit
= 1: the control unit involved is a multiple device control unit

bit 9 = 1: the disc used is an X1215 disc (P824)

bits 10 = 15 contain the device address.

Initial Program Loader (IPL)

The disc IPL program is written onto the disc when the disc is premarked. It is written in absolute binary, so, to enable it to run anywhere in memory it does not contain any memory direct reference.

When loaded, the IPL reads and loads into memory from the disc from which it has itself been loaded, starting from sector number /12 (the first two sectors of a file are reserved for the system). The first four words of sector /12 contain:

- start address of the load module
- number of sectors used by this module
  (This is the standard load format on disc; these words are generated by the disc linkage editor).

Note:
As long as it has been stored on the disc according to the IPL requirements, any stand-alone program, even if it does not use the disc, can be loaded into memory by the disc IPL.

Programs to be loaded by disc IPL
- must be in disc system load format (188 code words plus relocation per sector)
- must be catalogued on disc as a file starting at disc sector logical address /10
- must be built of consecutive granules.

STARTING A SESSION OR JOB

After the system has been loaded by IPL procedure, the operator must initialize the date and time. The system types out
DATE:
and the operator then answers by typing in the date as follows:

DD MM YY (LF)(CR) or YY MM DD (LF)(CR)

where DD,MM and YY are 2 characters giving day, month and year, separated by a delimiter, which may be any character on the keyboard. Then the system types

TIME:

and the operator answers by typing in the time as follows:

HH MM (LF)(CR)

where HH,MM are 2 decimal characters specifying hour and minute. The control panel key must be in CLOCK position, if the time must be updated automatically.

After the user has entered the time, the monitor types out:

BATCH PROCESSING? to which the user can reply with:

Y,<DNDA> if he wants to work in batch processing mode, where <DNDA> (device name + device address) is the new assignment for file code /E0; or

N (Y)

which will be followed by system output of

S:

after which the user can assign /E0:

AS␣<DNDA>, /E0

N if he wants to work in conversational mode.

If batch processing mode is chosen the system starts reading the control commands (on cards or punched tape) immediately. The first of these commands must be a JOB command.

If conversational mode is chosen, the system reads the user identification from the device with file code /E0. This will usually be the typewriter, so the system types out:

USERID:

and the user types in his user identification in one of the two following ways:

/<disc number>,<userid> or

<userid>

In the first case, the system will scan only the disc specified by /<disc number> to find the given user identification.

In the second case, the system will scan the catalogue of each online disc, starting with disc unit F0, until it finds the user identification specified. If SYSTEM is specified as user identification, the first user of disc unit /F0 will be taken, whatever name may be stored for this user on the disc. In such cases we have a SYSTEM session.

If the user is not yet present on a disc and has no entry in the Catalogue he must first declare himself to the system in order to be registered in the Catalogue and to obtain space for a directory for his own library. This must be done in a so-called system session, i.e. after the system message USERID: the user types in SYSTEM and gives a DCU control command with his own user identification. The system will then make an entry for him in the Catalogue and allocate a granule for his library directory. Then the user closes this session with the control command BYE, after which the system again types out USERID: Now the user may start with his own user identification and proceed as he wishes.

Note:
If the user reply to the message USERID: is
SYSTEM
a system session is opened; however, if the user types in
/FO, SYSTEM
the CCI will look for a user named SYSTEM on disc/FO and the session opened is not a system session.

In cases of errors, the following messages may be output:

- INPUT COMMAND I/O ERROR
An I/O error has been detected during the reading of the user identification. The user must type in a new userid on the typewriter.
- I/O ERROR
An I/O error has been detected during the loading of the disc allocation table from the disc into memory. The user must type his userid again on the typewriter.
- USERID UNKNOWN
The userid specified has not been found on any of the discs. The user must type in a new userid on the typewriter.

BATCH PROCESSING

Batch processing is started at system initialization time or when the user switches from conversational to batch processing mode by giving a BYE␣BYE control command.
The first command of a batch must always be JOB. The control command input stream, i.e. the sequence of CCI commands necessary to perform one or a series of jobs, is punched on cards or paper tape or output on disc and executed sequentially, without operator intervention.
The advantage of batch processing is, that when an error or abort occurs, the system simply looks for and starts the next job. When an error occurs, only the commands JOB, BYE and END are recognized and other commands up to one of these three are skipped. Because a job often requires many CCI commands, in more or less the same sequence, these commands can be stored on disc as catalogued procedures, in which case the user will only have to specify the procedure name and any necessary parameters in the input stream.
For this feature see page 23
The following restrictions must be taken into account:
- a program is not aborted when it reads the next job;
- there is no limit on the number of cards read, lines printed or records punched, nor on the execution time.

A batch is started as follows:
- put the batch command sequence on the input stream device (tape or card reader)
- first command must be a JOB command

Note: When a program exits, register A7 contains an exit code in its right character. If bit 8 of A7 is set to 1, in batch mode the whole batch will be aborted up to a following JOB or BYE␣BYE control command. The exit codes are specified under the monitor requests.

CHANGING A DISC PACK

The procedure is different depending on the type of disepending on changed:

- If it is the system disc, changing it must be followed by reloading the system by IPL procedure as described in the previous paragraph.
- If it is not the system disc, as soon as the disc becomes ready, an interrupt is sent to the monitor. Until the Control Command Interpreter is loaded again to re-initialize the system, reading or writing on the new disc is not possible and a non-operational status will be returned.

All file codes which had been assigned to the disc unit which becomes ready, are scratched, for catalogued as well as for temporary files. The codes assigned to the other disc units will not be affected.

Note:
If the user changes the disc pack containing the current session user, the system will close the session by simulating an automatic BYE control command and ask for the next user identification by typing USERID:

COPYING OR UPDATING THE SYSTEM DISC

Two examples are given to show how a copy can be made of the System Disc onto another disc and how a System Disc can be updated.

Example 1
After a new disc has been formatted by means of the PREMARK disc initialization program, it is mounted on the unit assigned to file code /F1. The system is assigned to the disc with file code /F0. /F0 must be copied onto /F1. It is assumed that the first user of

the System Disc (i.e. the system itself) is catalogued as OLDSYSTM. The following command sequence must be given:

```
USERID: /F1,NEWSYSTM   (NEWSYSTM must have been declared during Premark)
SVU OLDSYSTM,/FO
```

It is possible to replace the command SVU by

```
MOV
KPF
```

for each of the system components as they appear in the directory of OLDSYSTM. The monitor must be the first file and be stored on consecutive granules.

Note:

This procedure may be used only once and it must be done on a clean disc, i.e. one which has only been Premarked. If this procedure is done a second time for the same disc, the disc will be destroyed. To replace the supervisor a second time it is better to use the procedure described below, in example 2.

Example 2

It is assumed that disc number 1 contains the new system and that disc number 2 has to be updated. The system to be used is that of disc number 1. So, /FO is assigned to disc number 1, /F1 is assigned to disc number 2. The monitor of disc number 1 is catalogued as MON1 in the directory.

The following command sequence should be given:

```
USERID: SYSTEM
MOV MON1,/L
RSU /F1
BYE
```

It is also possible to use the system of disc number 2, in which case the command sequence is different:

/FO is assigned to disc number 2, /F1 is assigned to disc number 1. The first user (i.e. the system) of disc number 1 is catalogued as

NEWSYSTM and its first file is MON2. Then:

```
USERID: SYSTEM
MOV MON2,/L,NEWSYSTM
RSU /FO
```

SYSTEM MESSAGES

Apart from the messages which may result from an error in a control command the following messages may be output by the system:

Abort Messages

When a program is aborted for any reason, messages are output for the user on the devices with file code 01 and/or file code 02, specifying:

- the location where the abort occurred in the program:
  PROG ABORTED AT XXXX
- the reason for the abort:
  NOT WIRED INSTRUCTION
  OVERFLOW IN SIMULATION ROUTINE SAVE AREA
  BUFFER AREA DESTROYED
  TOO MANY SCHEDULED LABELS
  OPERATOR ABORT
  BUFFER ALLOCATION OVERFLOW
  DISK OVERFLOW
  DISK QUEUE OVERFLOW
  MEMORY OVERFLOW DURING LOADING PHASE
- the contents of the PSW and the registers at the moment of the abort.

Peripheral Unit Error Messages

- When an error occurs during an I/O operation, messages are sent to the operator, giving the status of the operation as follows:

  PU DNXX,ST,RY

  where:
  DN is the device name
  XX is the device address
  ST is the device status
  RY invites the operator to correct the error and retry the last operation. In this case the operator must press the interrupt (INT) button on the CPU control panel and type in

  RY XX

  after he has fixed the error or, if the error cannot be corrected,

he may release the operation on that device by pressing the INT button and typing

RD XX

- DKER␣<disc address>␣<cylinder/track/physical sector number>␣<status> is a message to signify a disc error. It provides the address of the disc as well as the cylinder, track and sector number (given in one number) and the hardware status (see Appendix D). If the status is /8000, the disc has become ready, but has not been reinitialized by the Control Command Interpreter.