P800M Programmer's Guide 2
Volume IV: Disc Real Time Monitor

A publication of

Philips Data Systems B.V.

Apeldoorn, The Netherlands

Publication number 5122 991 27404

November, 1978

Printed in The Netherlands.

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

The other volumes of this set, to be used in conjunction with this one, contain the following:

- Volume I: Disc Operating Monitor
- Volume II: Instruction Set
- Volume III: Software Processors

- Volume VI: Extended Disc File Management

Other books pertaining to the P800M series are:

P852M System Handbook
P856M/P857M System Handbook
P800M Operator's Guide
P800M Interface and Installation Manual
P800M Software Reference Data

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 the book, to:

SSS-DOC
at the address on the opposite page.

PART 1 MONITOR USE

The Disc Real Time Monitor (DRTM) is a disc-oriented system which is intended to supervise the execution of user application programs in a real time environment. It is assumed that these programs have been developed and pretested under the DOM.
The system is based on a structure of priorities, incorporating hardware interrupt levels and software priority levels, where up to 48 hardware interrupt signals and 15 software user levels can be handled. This system is enhanced by the socalled scheduled label feature.
The monitor is of modular structure to allow the user to easily adapt it to his particular application.

The P800M Disc Real Time Monitor has been designed to supervise the execution of pretested programs in a real time environment, on the basis of a priority system consisting of up to 48 hardware interrupt levels and 14 software priority levels. The DRTM is compatible with the Disc Operating Monitor (DOM), i.e. programs can first be developed and partly tested under the DOM, then used under the DRTM.

There is no memory protection, so all programs run in system mode. It is assumed that they have all been debugged and pretested. There is hardly any distinction between system and user programs; not all system programs are confined to any particular level, nor are user-written programs. It is therefore better to speak of standard system programs and application programs. In Part 2 of this manual some guidelines are given as to the interrupt levels to which some standard system programs can best be connected.
The priority system incorporates hardware interrupt lines and software priority levels:

- 0 to 47 are the levels for the hardware interrupt lines
- 48 is the level for interruptable monitor service routines
- 49 - 62 are software priority levels for system and application programs
- 63 is the idle task level.

The highest priority level is 0, so hardware interrupts will always overrule software level programs.
The interrupt routines which service the internal and external hardware interrupts are connected to levels 0 to 47.

Some of the interrupt routines are standard; the user can easily include interrupt routines written by himself, however. Incoming interrupts are handled by hardware and receive control on the basis of the

priority level to which they have been assigned. The dispatcher, a monitor module at level 48 which divides central processor time between competing priorities, also allots processor time to the user programs. The highest level active program always gets control until it is interrupted. Registers are saved by hardware in a system stack (addressed by register A15) and by software in the same stack or in a save area, which the system reserves in front of the program or in a special save area.
There are several types of programs, each related to a particular memory area: memory resident, read only, swappable and background programs.

Read only programs are loaded upon activation, one at a time; when a higher priority read only program is activated, the current one is erased. For this reason these programs must be read only.

Background programs are programs connected to the lowest priority level. Several of them may be in memory at the same time, allowing multiprogramming between them.

Swappable programs are executed on a time slice basis, according to priority. One such program at a time is loaded into a special partition and its execution started. If, at the end of a time slice a program of the same or higher priority has become active, the first program (in fact the whole swap partition) is swapped out, back onto disc, and the new one is loaded into memory. Otherwise the first program continues. Programs must be declared swappable before activation.

All these programs, in particular the read only ones, can make use of the dynamic allocation area to obtain temporary memory space dynamically. Not only programs, but also re-entrant subroutines will utilize this area. The dynamic allocation area is located behind the monitor area in memory. Programs obtain and release memory space in this area by issuing certain monitor requests.

User written programs and certain standard system programs have to be kept on disc and declared as memory resident or read only programs. Disc access can be done via a Data Management package in sequential or in random access mode. For physical I/O operations, drivers are used.
Under DRTM the EDFM package can be used for accessing extended disc files. This is described in Programmer's Guide 2, Volume VI: EDFM

Files are stored on discs or DADs (Direct Allocation Devices). A DAD is logically considered as a whole disc, although physically it may be only part of a disc. For each user there is one library, pointed to by the user identification in the disc or DAD catalogue. All his permanent files are stored in this library, which is located on one disc or DAD only, i.e. the one containing the directory for that user. This implies that one user cannot have his files stored on several discs or DADs unless of course, he makes use of several user identifications. However, a user can have access to the system library and other libraries by specifying the user identification of the user of those libraries, but he can only read the data in those files and 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.

All I/O operations are initialized by monitor requests. Monitor requests can perform various functions. They consist of an LKM instruction followed by a DATA directive with a number as operand, to define the function. Necessary parameters must first be loaded into certain registers.

For I/O operations, several functions can be performed, such as binary, ASCII and object I/O, with conversion and control facilities, random read and write operations, etc.
Peripheral devices or disc files are referenced through file codes, logical numbers of 2 hexadecimal digits.

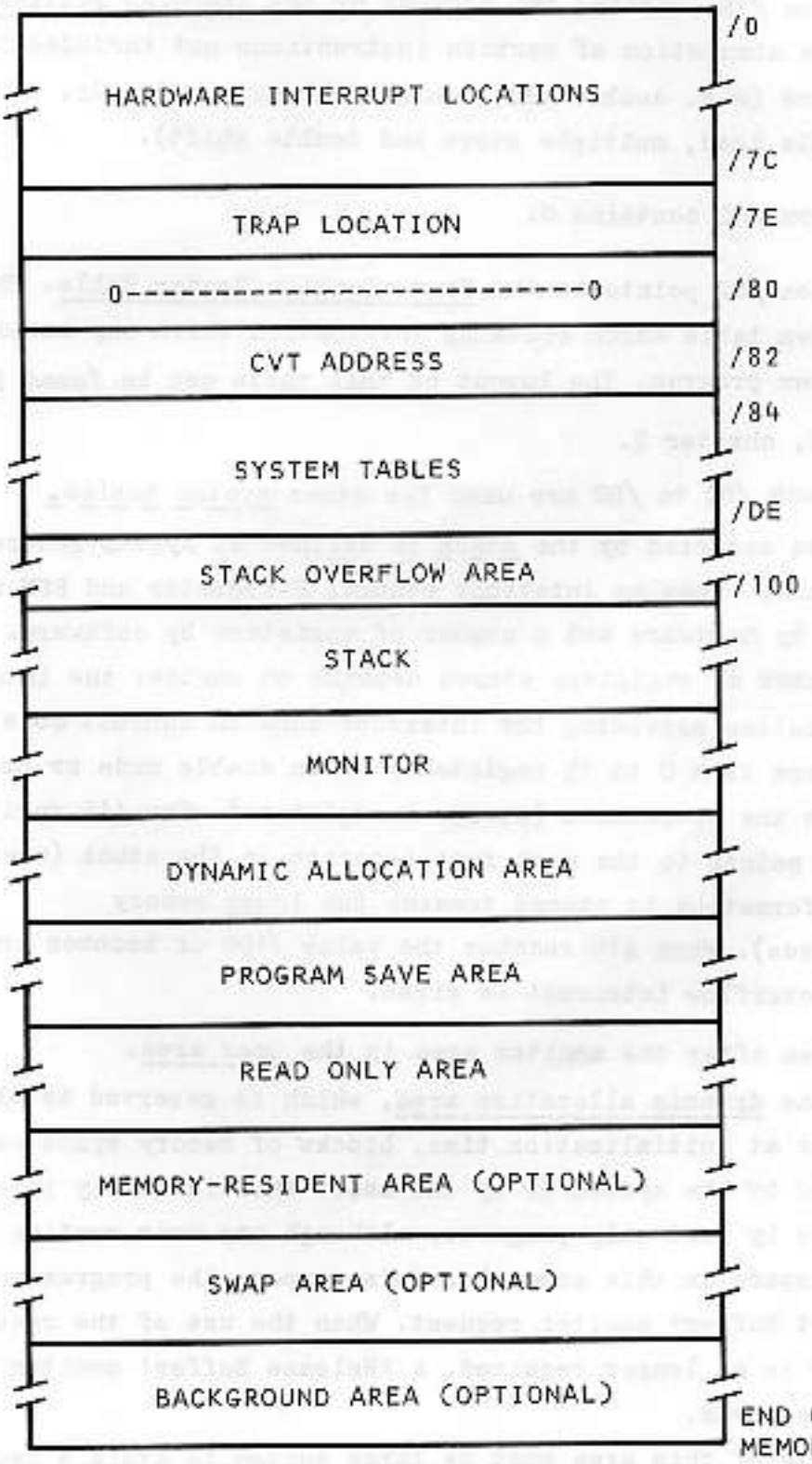
Monitor requests are also used to ask the monitor to perform the following functions:

- program activation
- waiting for the occurrence of an event
- program exit
- obtaining and releasing buffer space in the dynamic allocation area
- connecting programs to a priority level and disconnecting them
- switching from one program to another one at the same level to allow time slicing, i.e. allotting processor time to several programs on one level by turns, on the basis of timer interrupts
- getting the time and date
- informing the monitor of the occurrence of an event
- directing the monitor to wait with activation of a program until a certain, specified time
- reserving a peripheral device for exclusive use by one particular program; detaching that device from the program
- assigning and deleting file codes
- providing for the creating and use of unsollicited operator messages designed by the user himself.

A special feature, enhancing the multiprogramming aspect of the system, is the scheduled label, which is used in conjunction with monitor requests.

A scheduled label is a subroutine which is started when the monitor request function has been completed, although it is specified at the same time as the monitor request, i.e. the main program can continue while the requested function is performed. For example, combined with an I/O request, the main program continues while the requested I/O function is performed and the branch to the scheduled label routine is not made until that operation has been terminated. This can be very useful, for example to analyze the results of a monitor request.

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. The layout of this table can be found in Part 2, chapter 2.

- 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 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.
  From the dynamic allocation area, which is reserved by the monitor at initialization time, blocks of memory space can be requested by the system or by the user. This is mainly intended for use by read only programs, although any user routine may request space in this area. For this purpose the program must give a 'Get Buffer' monitor request. When the use of the requested buffer is no longer required, a 'Release Buffer' monitor request must be given.
  The size of this area must be large enough to avoid a dead-lock in the system. Programs requesting space in this area might be

put in wait state, if such a request cannot be serviced immediately.

The Read Only Area is used by the disc resident read only programs. These programs are loaded dynamically into this area, one at a time. When a read only program of higher priority than the one currently in memory is requested, the latter is erased by the former. For this reason these programs are read only and must use the dynamic allocation area for buffers, work areas, etc. This partition must be at least /880 words long. It may be used by standard system programs, such as the operator communication package and certain monitor request modules or by application programs. These programs must previously have been declared as read only.

The Program Save Area is used to save the registers of a program when it is interrupted. Its size can be specified at system generation time and must be such that it can contain all save areas of all programs, including system programs.

This save area must also provide space for saving registers in case one or more scheduled labels are included in the program.

The Memory Resident Area is an optional partition. If used, it contains programs which are urgently required, i.e. interrupt routines or high priority real time programs. These programs are loaded on operator request, remain resident in memory and are scheduled according to their priority levels.

The Swap Area is used to load swappable programs into memory, one at a time, on the basis of priority level and time slicing. These programs, when swapped, are written back onto disc, so they may need a work area in the Dynamic Allocation Area.

The Background Area is optional as well and is used solely for programs connected to level 62. These are loaded dynamically and several of them may be in this area at the same time, to allow for multiprogramming between them. No program swapping takes place. When execution of a program is completed, its memory space will be released automatically. The size of the monitor area is calculated by the system at loading time and depends on the configuration.

The size of the other memory areas can be defined at system generation time, but they may be modified at system loading time, when the monitor is initialized.

Programs and routines under DRTM 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 lines
48 : interruptable monitor service routines	software priority levels
49 : high-priority system or user programs	
50-63: user and system programs:	
50-61: foreground programs	
62 : background programs	
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 :	floating point processor	etc.
/4 :	not used	

/5 : not used
/6 : operator's console
/7 : control panel
/8 - /E: Data Communication on Programmed Channel
/F : punched tape reader
/10: CDC disc (address /16)
/11: X1215 disc (address /02)
/12: flexible disc (address /03)
/13: magnetic tape
/14: cassette tape
/15: card reader
/16: tape punch
/17: line printer
/18: X1215 disc (address /01)
/19: CDC disc (address /17)
/1A - /1F: Data Communication on I/O Processor
/2F: page fault

Levels 48 and 49

Level 48 is handled by the system as a pseudo-hardware interrupt level, i.e. programs on this level also use the A15 stack and have the same interfaces as interrupt routines. Usually they are intended for processing a higher level interrupt, which does not require servicing at that level. For example, an I/O request is entered via an LKM interrupt. As soon as the request is recognized, the I/O program branches to level 48, so that other interrupts may be serviced, while the I/O request is handled.
Most of the I/O interrupts are received at high priority levels, but processed at level 48.

Level 49 is used especially by the system to process a number of monitor requests. However, any user program may use this level. System and user programs are not distinguished at this level, they are dispatched in the same way. This implies, that user programs at level 49 may influence the response time of system programs at level 49 in a real time system.

Software Priority Levels

The levels 50 to 62 are connected to user programs which may or may not be related to each other. The programs are activated by control command or monitor request, in which case it is possible for one program to activate another one.
The levels 50 to 61 are reserved for foreground user programs, while level 62 is reserved for background user programs, i.e. programs which have the lowest priority in the system. Actually these are programs which have not been connected to a software level. They are automatically loaded as background level programs.

Note: It is possible to connect a foreground program (core-resident, read-only or swappable) to level 62. However, when such a program exits, it is 'erased', as is done with background programs.

By means of the monitor request 'Switch inside a Level' it is possible to divide processor time among several programs on one level, on the basis of real time clock interrupts.
Requests for program activation are handled by the 'Activate' monitor module, then passed on to the dispatcher (see below).

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

Dispatcher

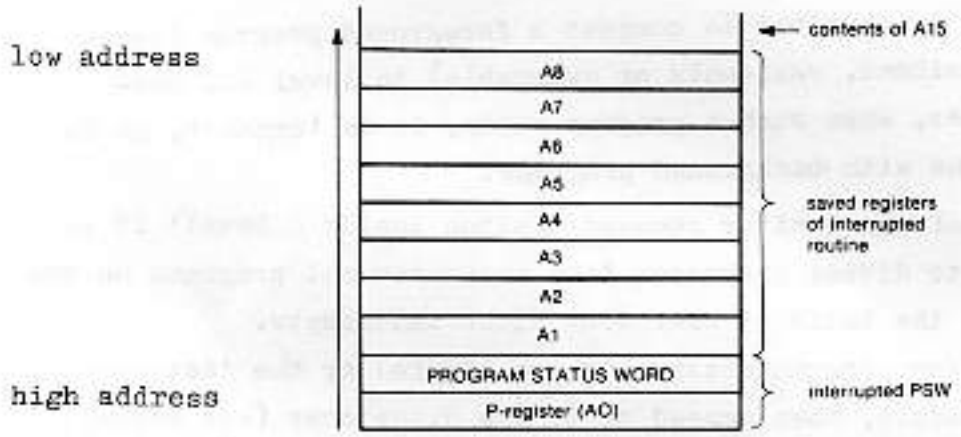
The dispatcher is a monitor module (M:DISP) running on level 48, which divides central processor time by starting programs according to their priority.
The dispatcher can be entered only from an interrupt routine, i.e. from a level below 48, such as the I/O interrupt and monitor request handlers.

System Interrupt Stack

When an interrupt occurs, certain information about the interrupted program or routine must be saved before the interrupt can be serviced.

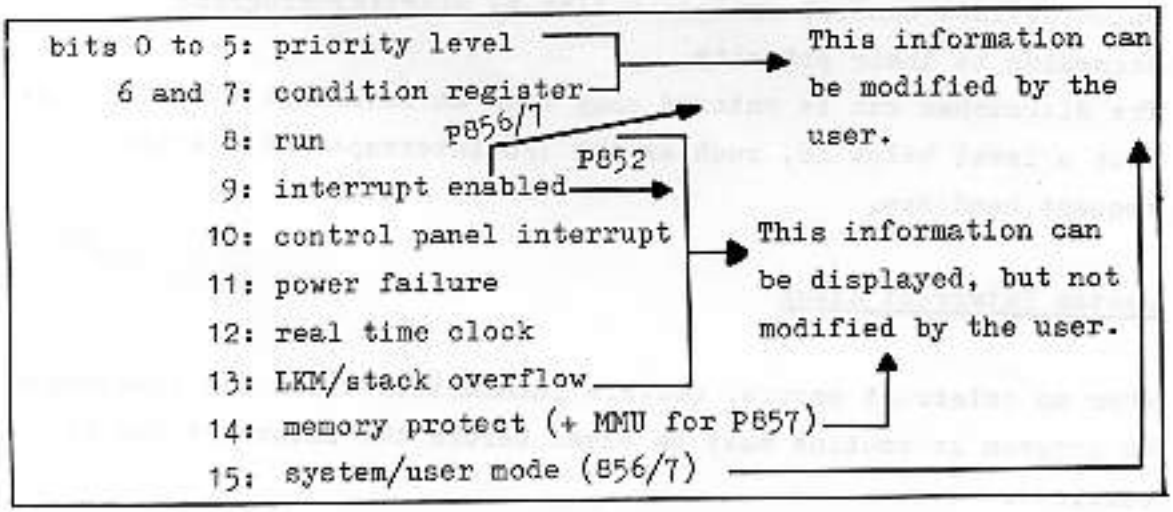
This is done in the system stack, the address of which is held in register A15. The start address of this stack is defined at system generation time and it is built in a downward direction in memory, i.e. towards the lower addresses. The A15 register always points to the first free location in the stack.
Upon interrupt, PSW and P-register are always saved in this stack and also a number of registers:

- <u>any number</u> if the interrupt routine runs in inhibit mode and takes care of restoring the registers itself;
- <u>registers A1 to A8</u> 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:



There are several types of user-written programs:

- interrupt routines, servicing one of the hardware interrupts and connected to any of the hardware levels 0 to 47
- subroutines (reentrant or non-reentrant); they are called through a CF (Call Function) instruction and run on the level of the calling program
- programs, connected to any of the software levels 49 to 63. This category is subdivided as follows:
  * foreground programs are connected to any of the levels 49 to 61, programs on level 62 operate in the background area of memory, level 63 is reserved for the idle task.
  * foreground programs can be either core-resident, i.e. always present in memory, or read only or swappable, i.e. loaded on request. This implies that for each of these program types a different memory partition must be used.

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. It can also be done via the LD command (see chapter 9).

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-
STR     A8,A15          lected at sysgen for P852, the sequence is:
                        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

Software level programs under the DRTM are those application or standard system programs which are connected to any one of the levels 49 to 63.
The programs required for a real time process may be used by one or more other programs. They are loaded into memory as separate programs and the connections between the programs which are related to each other are supplied by monitor requests in the individual programs.

A program is connected to a level either at loading time (CN Control Command) or by a monitor request (Connect a Program to a Level - LKM20). The level is registered in the PSW (bits 10-15). Program activation is also possible in two ways:

- by Control Command, at initialization time.
- by an 'Activate' monitor request (LKM12) from another program.

After activation, the dispatcher will start the programs according to their priority.
Besides being connected to a level, programs can also be connected to a timer, so that they may be activated at a certain time or after regular intervals, according to parameters which the user specifies in a monitor request 'Connect a Program to a Timer' (LKM10).

It is also possible to share central processor time between various levels and between memory-resident programs of the same level (including background programs, once they have been loaded into memory) by time slicing, i.e. the monitor request 'Switch Inside a Software Level' (LKM13), see below. The 'scheduled label' is another feature which extends the possibilities for real time programming, see below.

After a program has been loaded, all relevant information about it is stored in a Program Control Table (see Part 2, Chapter 2) in the PCT Pool inside the monitor area in memory.
The program remains under monitor control until it is disconnected from its level, during which time it passes through various states, as recorded in the PCT:

- inactive: the program has been connected to a level, but it has not been called yet;
- active: the program has been called, and is not yet terminated;
- wait for execution: the program is ready to use central processor time when it has the current highest priority;
- wait for an event: the program has given up control voluntarily with a 'Wait for an Event' monitor request (LKM2) to wait for the occurrence of a particular event.

Memory resident Programs

Aside from interrupt routines, some high priority foreground programs may also require to be memory resident. They should be so, when their activation, in response to a request, cannot be delayed by the time required for loading such a program. These programs are loaded at initialization time by control command LD, which implicitly defines them as memory resident. They occupy a special memory partition (see Chapter 2).
A 17-word save area is allocated in the program save area (see below). When scheduled labels are used, 17 + 1 + 16 + n words are saved, where n is the maximum number of scheduled labels allowed in queue.

Read only Programs

Programs, for which the response time is not so critical, can be declared read only by the control command RO, given at initialization time. They are loaded later on, on request, into a special partition (see Chapter 2), which must be at least /880 words long. Only one program at a time occupies this partition. When a higher priority read only program is activated, the current read only program is erased by loading the higher priority one. After this one has terminated and made its exit, the former read only program is resumed.
CPU time is therefore not shared between programs in this partition. Since a read only program can be interrupted and erased at any time, certain provisions must be made.
These programs are read only, so they may not contain any locations that may be modified, any work areas, I/O buffers. For these, the dynamic allocation area must be used, so that this information is always safe in case a read only program is erased by another one. Space in the dynamic allocation area may be obtained and released by means of the monitor requests Get Buffer (LKM4) and Release Buffer (LKM5).
Moreover, the system provides for a save area partition.
In this area the contents of the erased read only program's registers and PSW are saved. The size of this area must be specified

at system generation time and must be large enough to contain the save areas of all user and system programs (see Part 2).
This save area is managed by the system.
Finally, Read only programs should not give a Wait request for an event which must be set by another read only program of lower priority, for any Wait request will lock the read only area to other read only programs of lower priority.

Program Save Area

The program save area is reserved to store the context (P-register, PSW, registers A1 to A14) of programs running at one of the levels 49 to 63, in case of a priority change. This may be the case when control is given to a higher priority program or, if the current program is waiting for an event, control may also be given to a lower priority program.
(If a program is interrupted, its context is saved in the system stack addressed by register A15).
The save area consists of 17 words for programs not using any scheduled labels or 17+1+16+2n words for programs using scheduled labels, where n is the maximum number of labels which may be scheduled at the same time (see also Part 2).
The save area is allocated by the system command language SCL, each time a foreground program is declared (or loaded) or a background program is activated.

Swappable Programs

A swappable program is executed one at a time, in a special partition in memory, from which it may be swapped back onto disc to make room for another swappable program. Program swapping may occur in this area when a swappable program of a higher or equal priority level is activated and, at the same time, a time slice has elapsed.
A time slice is a predefined period of time during which a swappable program is allowed to run. The swappable program which is in

memory is swapped at the end of a time slice for another one with the same or a higher priority level, unless it is the only program on the current level and this is also the highest active level, in which case the program continues for another time slice.
There is a difference with read only programs in that swappable program are not erased but written back onto disc integrally. However, swappable programs may have their buffers and ECB's allocated in the dynamic allocation area, e.g. for wait with swap-out, non-disc I/O. The response time is much the same as with read only programs, however, because the swap operation will take a certain amount of time, depending on the speed of the disc and on the size of the swap program.
Any foreground program from level 49 to 61 may be declared swappable. This is done when it is loaded, by means of the control command SW. The length of the time slice can be defined at System Generation, in the SW command or by means of a separate command: TS.

Background Programs

These are programs of the lowest possible priority level (62), i.e. time consuming programs of which it does not matter if their execution is delayed.
Background programs are loaded automatically when a request for their activation is made. They need not be connected to a level before activation, for they will automatically be loaded as level 62 programs. It is possible to have more than one program in the background area at the same time, so multiprogramming is possible between background programs. When execution of a background program is terminated, the memory space it occupied will be released automatically.

More detailed information about the internal management of the memory partitions is given in Part 2.

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
```

Under DRTM this sequence can be replaced by another one, for example measuring the system's performance and computing the CPU idle time. It must have the same interface as other user programs, with the following exceptions:

- No monitor requests must be used which will put the program in wait state, explicitly (Wait for an Event) or implicitly. Therefore, I/O to output calculation results should preferably take place in a program at another level, e.g. 62.
- The idle task may not use the Exit request, since it must consume all idle CPU time and must loop inside the program.
- The entry point must be IDTASK and it must be processed with the user Library before the system Library at SYSGEN.

If no monitor requests are used, the response time of the system will not be affected by the idle task.

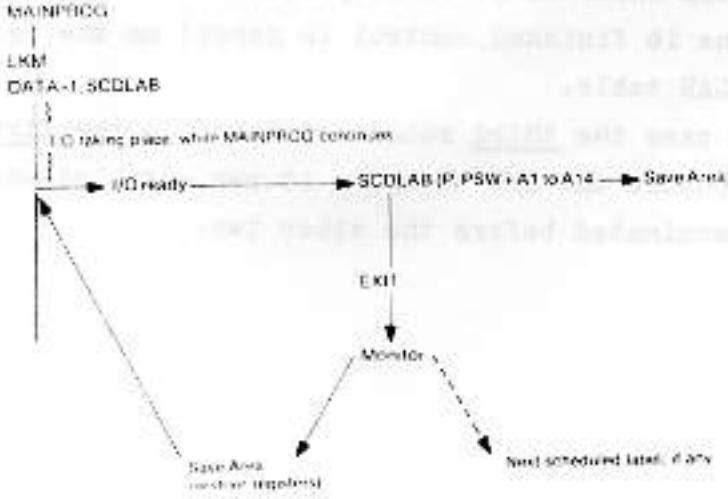
It will be clear from the above subdivision that, although a program may be declared in any class, the main reasons for deciding on program declaration will be response time and calling frequencey.

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 contents of A8 is significant: ECBADR) 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 system genera tion time. 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 execution of 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.

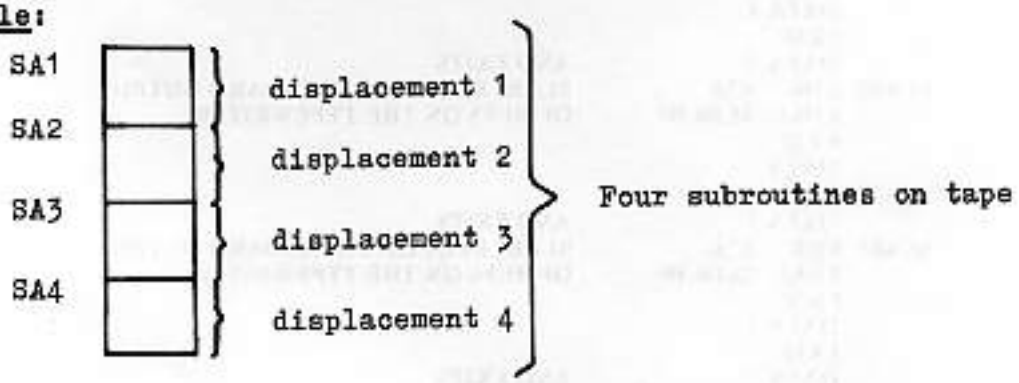
```
        IDENT SLAB
BUF1    DATA '1234567890'       SPECIFYING THE CHARACTERS TO BE
BUF2    DATA 'ABCDEFGHIJ'       OUTPUT
BUF3    DATA 'ZYXWVUTSRQ'
BUF4    DATA '1A2B3C4D5E'
BUF5    DATA 'BUF5'
BUF6    DATA 'BUF6'
DCB1    DATA 5,BUF1,5,0,0,0     DECLARING THE EVENT CONTROL
DCB2    DATA 5,BUF2,40,0,0,0    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
        LDK1  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
        LDK1  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
        LDK1  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
        LDK1  A8,DCB2           OF BUF2 ON THE TYPEWRITER
        LKM
        DATA 1
        LKM
        DATA 3                  AND EXITS
SLAB2   LDK   A7,6              SLAB2 REQUESTS STANDARD OUTPUT
        LDK1  A8,DCB5           OF BUF5 ON THE TYPEWRITER
        LKM
        DATA 1
        LKM
        DATA 3                  AND EXITS
SLAB3   LDK   A7,6              SLAB3 REQUESTS STANDARD OUTPUT
        LDK1  A8,DCB6           OF BUF6 ON THE TYPEWRITER
        LKM
        DATA 1
        LKM
        DATA 3                  AND EXITS

        END   START
```

Re-entrant Subroutines

Re-entrant subroutines are subroutines which can be used by more than one program. They can be interrupted and restarted for a higher priority program at any time.
These subroutines must be loaded into memory in such a way, that the programs which will use them, can find their start addresses. To this purpose they can be link-edited with each other (a dummy operation). At the end of the process the Linkage Editor will print a MAP. This map will enable the user to find the displacement of each of the subroutines.
Now the user can write a dummy program with the start addresses of the subroutines used by his program.
This dummy program is loaded and then link-edited with all the programs which use the loaded subroutines, so that these programs know the subroutine start addresses.

Note: Direct addressing is not possible, it is recommended to use the registers for this purpose.

Example:



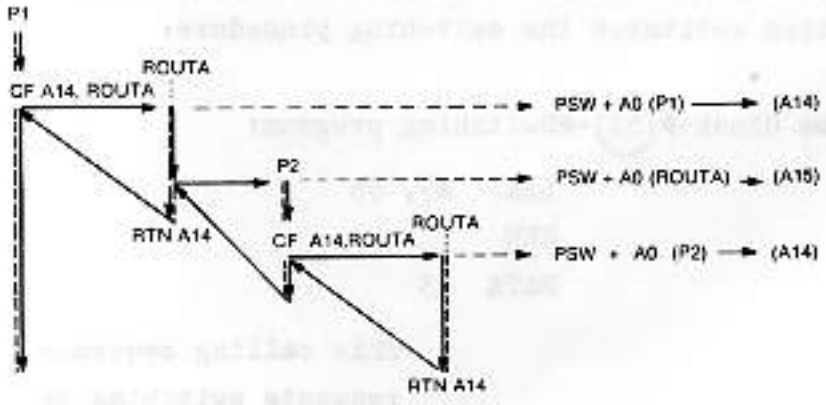
Dummy program:

```
        IDENT   DUPR
        ENTRY   SA1
        ENTRY   SA2
        ENTRY   SA3
        ENTRY   SA2
SA1     EQU     /2500
SA2     EQU     SA1 + 'displacement 1'
SA3     EQU     SA2 + 'displacement 2'
SA4     EQU     SA3 + 'displacement 3'
        END
```

This dummy program must be linked with each program using any of the four subroutines. The programs, for example, are as follows:

```
IDENT     PROG
|
EXTRN     SA1
EXTRN     SA4
|
  CF      A14,SA4
|
  CF      A14,SA1
|
 END
```

If a re-entrant subroutine wishes to call another re-entrant subroutine, it must reserve two words in the dynamic allocation area through the 'Get Buffer' monitor request. The subroutine obtains any other memory space it requires, in the same manner.
Re-entrant subroutines cannot use scheduled label routines.
If a subroutine is interrupted for another program of a higher level, requiring the same subroutine, we get the following situation:



- Program P1 starts running.
- CF is given, PSW + P of P1 are stored in stack, of which the address is given in A14 and subroutine ROUT is started.

- Interrupt for program P2, PSW and P of ROUT are stored in the stack specified in A15 and P2 starts running.
- CF is given, PSW + P of P2 are added to the stack specified in A14 and subroutine ROUT is started and allowed to terminate with RTN A14.
- P2 is resumed and allowed to terminate normally.
- Via A15 subroutine ROUT is resumed and allowed to terminate with RTN A14.
- P1 is resumed.

Note: the stack indicated by A14 has to be reserved by the user; the stack indicated by A15 is reserved and handled by the monitor.

Time slicing

For some software level programs time slicing is possible, i.e. allotting processor time to several programs connected to one level, under control of a timer and by turns. This is allowed only for memory-resident programs and background programs once they have been loaded and functions as follows:

Let us assume that level 52 is connected to a timer, e.g. the real time clock, and that a program on level 52 contains the monitor request which activates the switching procedure:

Real Time Clock → (52) → Switching program:

```
LDK   A7, 55
LKM
DATA  13
```

This calling sequence requests switching on level 55.

```
LKM
DATA  3
```

'Exit' monitor request to go to level 55.

Level 55 has three programs connected to it: A, B and C.
At an interrupt of the real time clock (every 20 milliseconds), the program connected to level 52 performs the switching request and exits. The dispatcher then activates one of the programs A, B or C on level 55, to which ever one it was pointing. This program then starts running:

- until an interrupt starts an interrupt routine (level 0 to 47) or until it is required to start level 53 or 54, e.g. because an I/O operation for which one of them was waiting is ready.
- until 20 milliseconds have passed. Then the real time clock gives another interrupt and control is returned to the program on level 52. Another request is performed, level 52 exits and, barring higher priority interrupts, the next program on level 55 is started, to run until another 20 msecs. have passed or a higher priority interrupt occurs.

This sequence continues until the programs A, B and C have had enough processor time to be executed.
If, for any reason it is desired to limit the interrupts during switching as must as possible, the switching program can be connected to level 50 and the programs to be switched to level 51. In this case, only hardware interrupts can interfere.

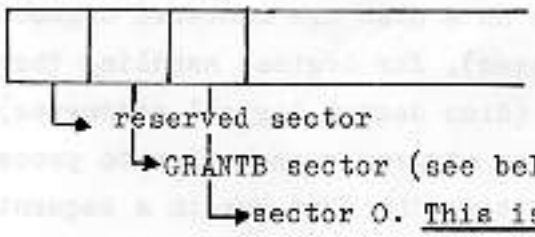
Note: This request is also used by some of the monitor modules, i.e. those which handle the monitor requests Activate, Exit, Detach Device, Release Buffer and Input/Output.

In a swappable program the time slice counter can be forced to zero by some monitor requests:

- Wait for an Event/Wait for a Given Time: if bit 15 of ECB/Param Block is set to one, the time slice is set to zero.
- I/O with Wait: if bit 7 of the order is set to one, the time slice counter is set to zero.

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.

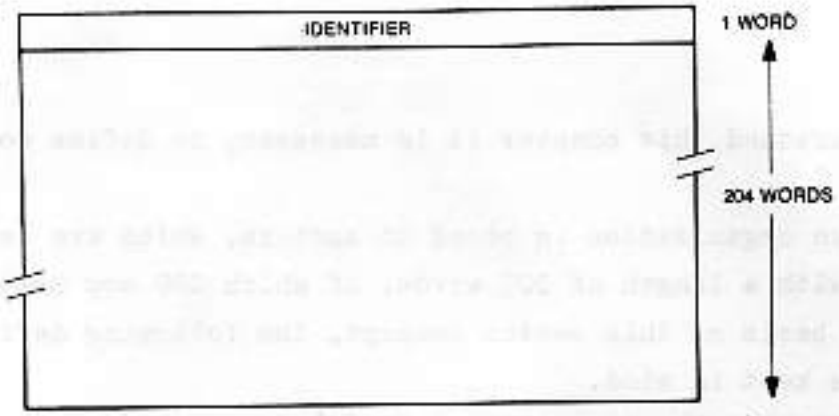
- Disc Sector Physical Address (DSPA):
  the physical address of a sector on the disc, i.e. the address of a sector in a consecutive arrangement.
- Disc Sector Logical Address (DSLA):
  the address of a sector on the disc as calculated by interlacing, which is ordering the sectors in such a way as to 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:



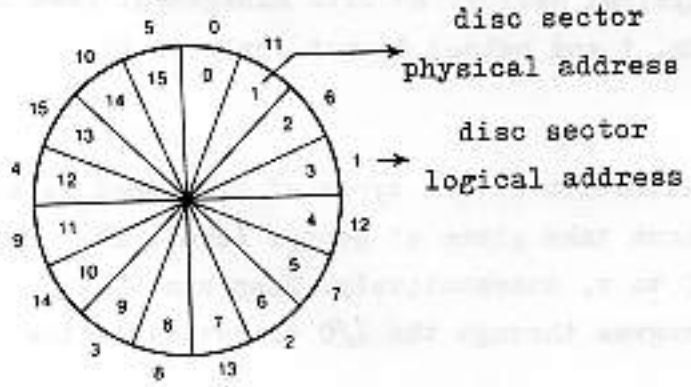
The types of disc which can be used in a P800 configuration are X1215 and X1216 discs, CDC discs (40M and 80M) and flexible disc. The flexible disc is considered as only a physical device, so Data Management (see Ch. 7) and the DAD concept (see Ch. 1 and below) do not apply to it.

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 only on X1215 or X1216 discs by the Disc PREMARK program. On CDC discs no identifier is written. 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 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, e.g. on a factor-3 basis for discs with 16 sectors per track (X1215/16).



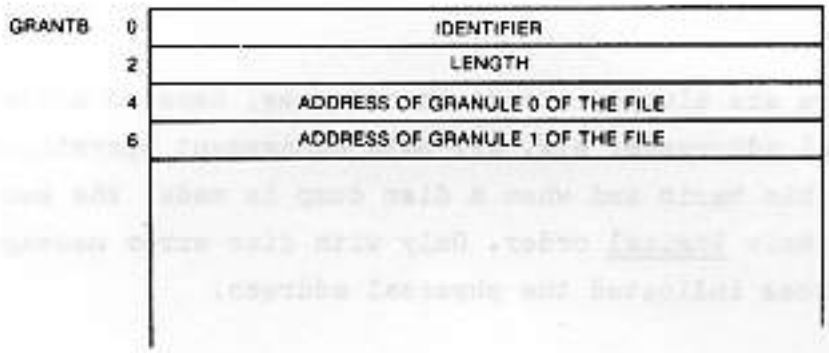
16 sectors

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.

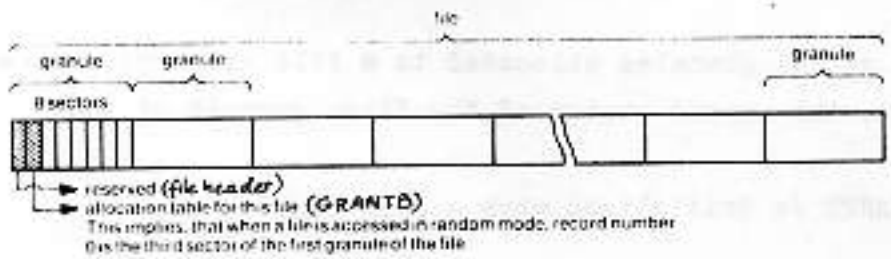
For sequential access, when an attempt is made to write onto a granule which has not yet been allocated, a new one is allocated and GRANTB is updated. The system manages a table called BITAB which contains one bit for each granule on the disc or DAD. This table is copied into memory when a disc pack 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.



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.

Note: Random access files of more than 200 granules are possible, but when such a file is deleted under the Disc Operating Monitor (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).

The first two granules on a disc or DAD 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.
- 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 DRTM, the requested number of granules is allocated and the file cannot be extended, unless it is completely rewritten.
- Random access files should be Assigned, Kept and Deleted under DRTM, not under DOM. See Note on previous page.
- records are accessed directly by specifying the file sector address.
- records may be retrieved, updated and restored individually.

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

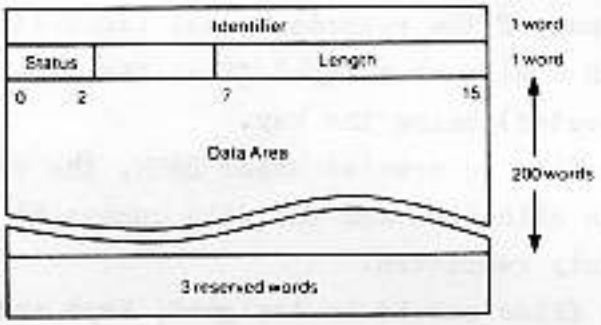
logical sequence of the records is identical to the physical sequence.
- to retrieve a record, the whole file must be scanned up to the desired record.
- to update a record, the file must be processed as a whole.

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.

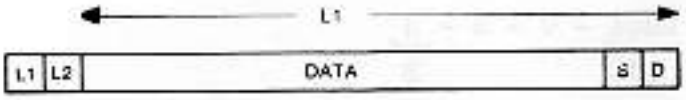
In sequential files a sector may contain a logical record or it may be part of a logical record. Sectors have the following format in these files:



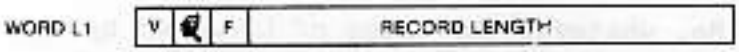
- status consists of 3 bits:
  bit 0: if 1, this sector has been deleted
  bit 1: if 1, an EOS (End-of-Segment) record has been written in this sector. This record is the last one in the sector, for the first record following an EOS always starts at a new sector.
  bit 2: if 1, EOF (End-of-File) record 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
Q = 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 - C - S - D
(4004 = L1)

:EOF: 2004 - C - 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 DRTM, the following file types can be distinguished:

- all programs must be kept under load module format and they are identified as file type LM. Such files must be created either under DOM or at system generation.
- other kinds of files are considered as data files and are identified as file type UF. These files can be created and kept under the DRTM itself.

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

- All system read only programs are smaller than /960 words, so each one requires one granule.
- The monitor will generally require from 5 to 12 granules.

D:CI File Size

This file is used by the system to keep all read only and swappable programs in core image format.
The size of this file is declared during the DEMGEN phase at System Generation time.
Each sector contains 200 program words, so for each system or user read only or swappable program, the required number of sectors in the D:CI file is (SRO - 1)/200+1
SRO being the program size in words.

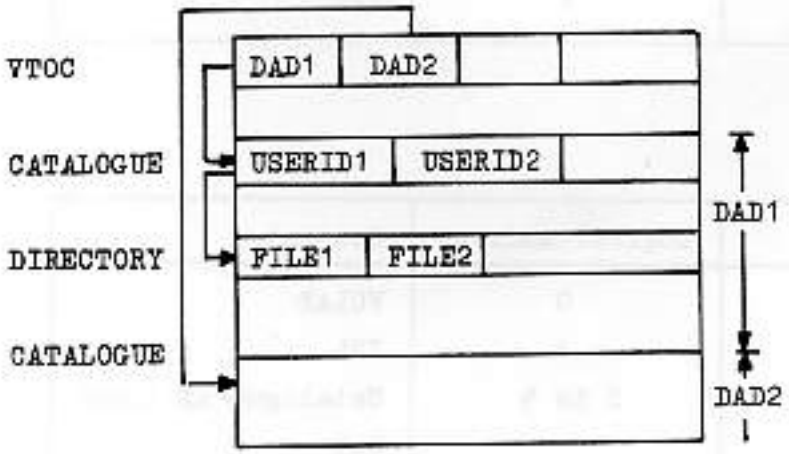
- As all system read only programs are smaller than /880 words, each one requires 6 sectors on the D:CI file, i.e. 8 granules for the 10 system programs.

DISC STRUCTURE with DADS

In such a structure there are three levels of disc access:

- to the physical disc, performed by the driver.
- to a DAD (Direct Access Device), part of the physical disc, but logically considered as a complete disc. The logical structure may vary from one DAD to another. It may consist of up to 32768 physical consecutive records of 410 characters each. All DADs on one physical disc are catalogued in the VTOC (Volume Table Of Contents) at the beginning of the disc. They are defined and assigned a file code in the range CF to C0 at sysgen. These are internal file codes. CF is the system DAD. The user can assign any user file code to the sysgen-defined DADs.
  A DAD is divided into granules (sets of 6 sectors).
- to files, belonging to a DAD. All files of one user are kept in a User Library. Files can be accessed randomly or sequentially at physical or logical record level.
  At the beginning of a DAD a catalogue of user directories is created, containing entries to identify users (Userids) and the addresses of user file directories. For each Userid 1 granule is reserved to contain file entries.

The physical disc structure is therefore as follows:



Physical Structure

The first DAD on the disc starts at cylinder 0.
The first granule contains the disc organization characteristics:

- VOLAB (volume label):
  - sector 0 on X1215/X1216 discs
  - sector i on CDC discs, where i is the number of interlaces on the first DAD.
- IPL (Initial Program Loader):
  - sector i on X1215/X1216 discs, where i is the number of interlaces on the first DAD.
  - sector 0 on CDC discs.

The logical sector addresses (relative address in the first DAD) of these sectors are:

X1215/X1216:

Physical Address	Logical Address	Contents
0	0	VOLAB
i (interlacing)	1	IPL
	2 to 5	Catalogue of first DAD
6i	6	Bad track list
7i	7	VTOC

CDC:

Physical Address	Logical Address	Contents
i (interlacing)	0	VOLAB
0	1	IPL
	2 to 5	Catalogue of first DAD
	6	Bad track list
	7	VTOC

Note:- For both CDC and X1215/X1216, logical sector 0 contains the VOLAB and logical sector 1 contains the IPL.
- Physical access is possible only on a DAD.

Although physical sector 0 is used differently for the various disc types, 5 words in this sector must always be set as follows, to be able to initialize a disc pack when it is mounted.

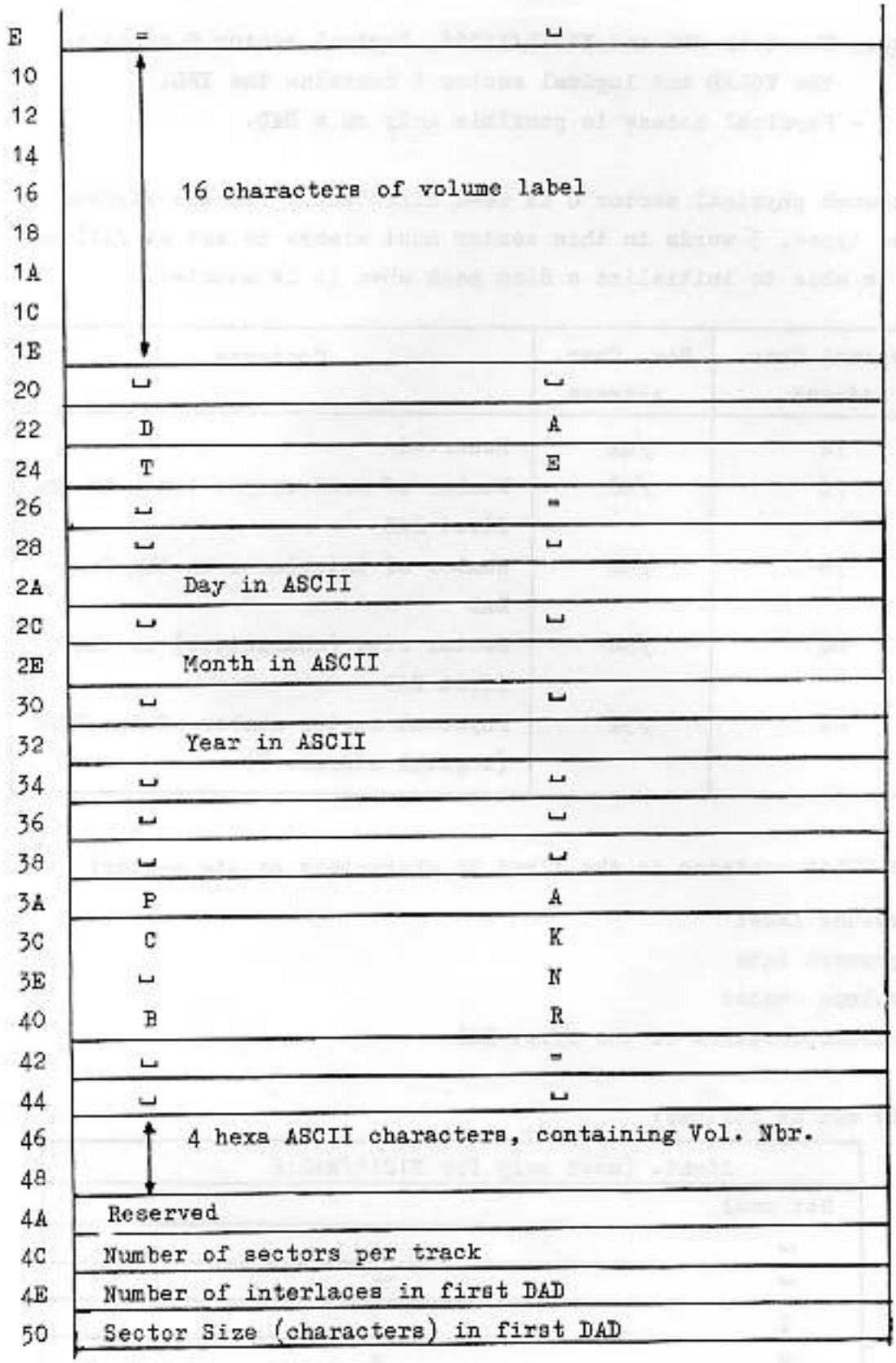
Decimal Char. Address	Hex. Char. Address	Contents
74	/4A	Reserved
76	/4C	Number of sectors per track in the first DAD
78	/4E	Number of interlaces in the first DAD
80	/50	Sector size (characters) in the first DAD
82	/52	Physical sector number of the VTOC (logical address 7)

The VOLAB contains in the first 86 characters of its sector:

- volume label
- premark date
- volume number
- characteristics of the first DAD

laid out as follows:

/0	ident. (used only for X1215/X1216	
2	Not used	
4	␣	␣
6	␣	␣
8	L	A
A	B	E
C	L	␣


E
10
12
14
16
18
1A
1C
1E
20
22
24
26
28
2A
2C
2E
30
32
34
36
38
3A
3C
3E
40
42
44
46
48
4A
4C
4E
50
16 characters of volume label
D A
T E
Day in ASCII
Month in ASCII
Year in ASCII
P A
C K
N
B R
4 hexa ASCII characters, containing Vol. Nbr.
Reserved
Number of sectors per track
Number of interlaces in first DAD
Sector Size (characters) in first DAD

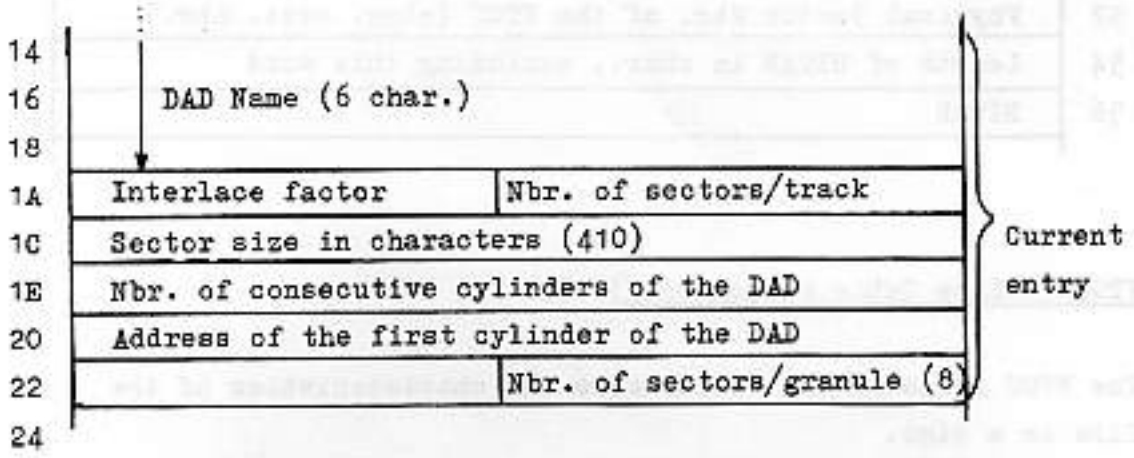
52	Physical Sector Nbr. of the VTOC (=log. sect. nbr.)
54	Length of BITAB in char., excluding this word
56	BITAB

VTOC (Volume Table of Contents)

The VTOC contains the coordinates and characteristics of the DADs on a disc.
Each entry consists of 8 words. The entries are written in the order in which the DADs are defined on the disc. I.e. when the address of the first cylinder of a DAD is not equal to the address of the last cylinder + 1 of the previous DAD, a number of free cylinders exists between these DADs. If possible, they will be used for a following DAD allocation before

The layout of the VTOC is as follows:

0	ident. (used only for X1215/X1216)	
2	Nbr. of char. used (this and last word included)	
4	V T	
6	O C	
8	␣ ␣	
A	0	First
C	Nbr. of tracks per cylinder	entry
E	Nbr. of cylinders on the disc	(dummy)
10	Address of first free cylinder	
12	Reserved	



where:

- word A is set to zero.
- word C is set at premark.
- word 10 points to the first free cylinder following the last DAD in the VTOC.

  Note: Cylinder addresses give the relative position of the cylinder from cylinder 0 of the disc.
- The interlace, word 1A, is used to compute the physical sector number as follows:
  - i = logical sector number in the track
  - physical sector number = i x interlace, modulo number of sectors per track.

Entries in the VTOC have the same order as the related DADs on the disc.

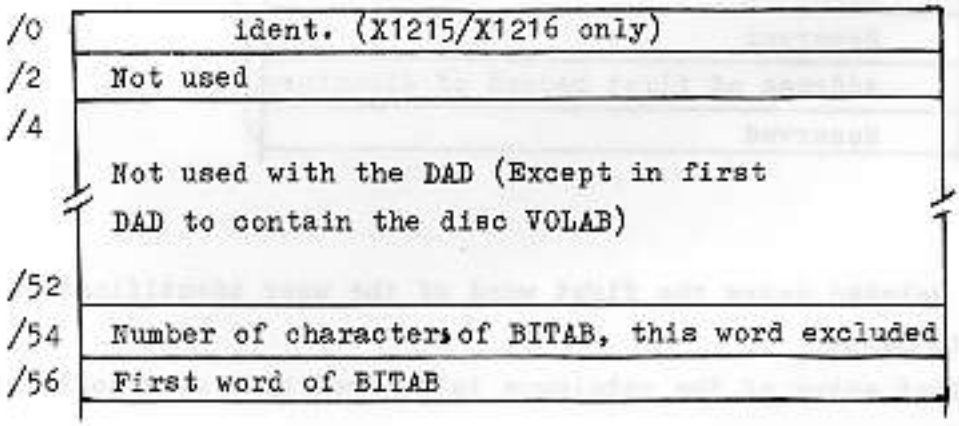
The last VTOC entry is followed by a word containing FFFF when the sector is not full.

First DAD Granule

The first cylinder of the first DAD is always cylinder 0 of the disc. As the first granule contains disc structure characteristics as well as DAD information, the user must re-Premark the disc when he removes the first DAD.

Sector 0

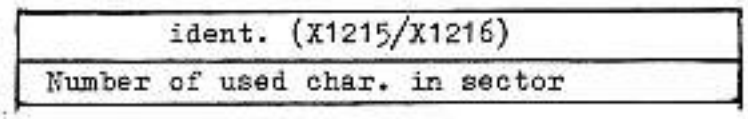
This sector contains the BITAB of the DAD. The BITAB gives the status of all the granules in the DAD. It starts at location /54:

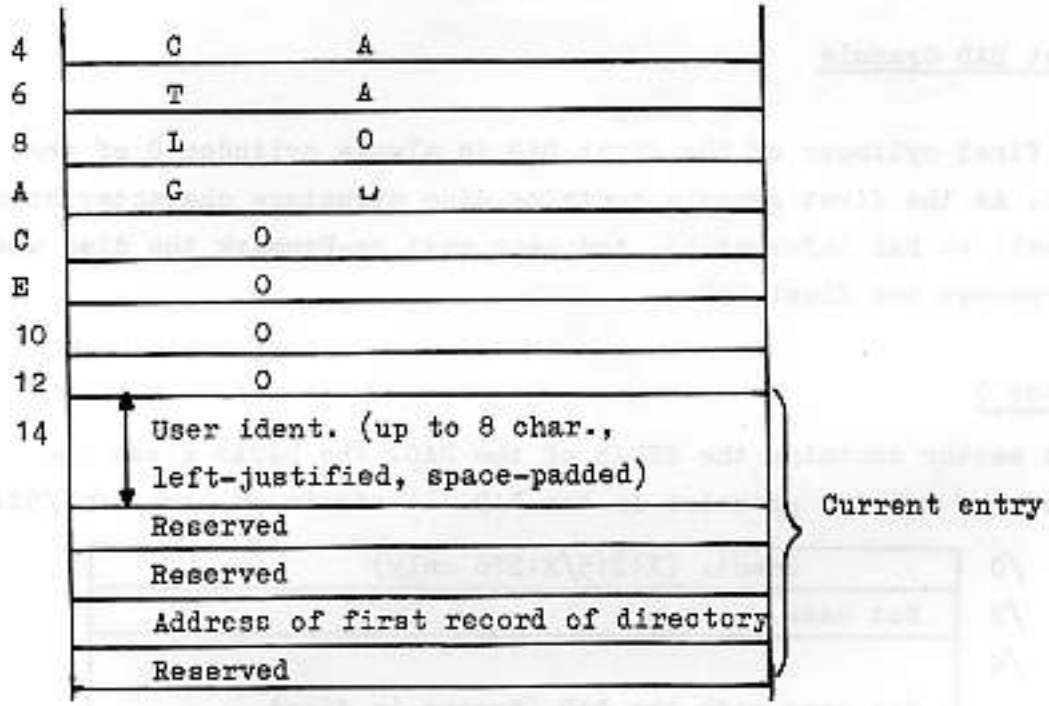


Each bit in the BITAB gives the status of the associated granule, 0 indicating an allocated or non-existing granule, 1 indicating a free granule.
BITAB length depends on DADsize and number of sectors per track in the DAD.
If any sector words remain unused, they are reset to zero, BITAB is created when the DAD is initialized.

DAD Catalogue (sectors 2-5)

The catalogue of the users of the DAD is contained in sectors 2 through 5.
Each entry consists of 8 words:





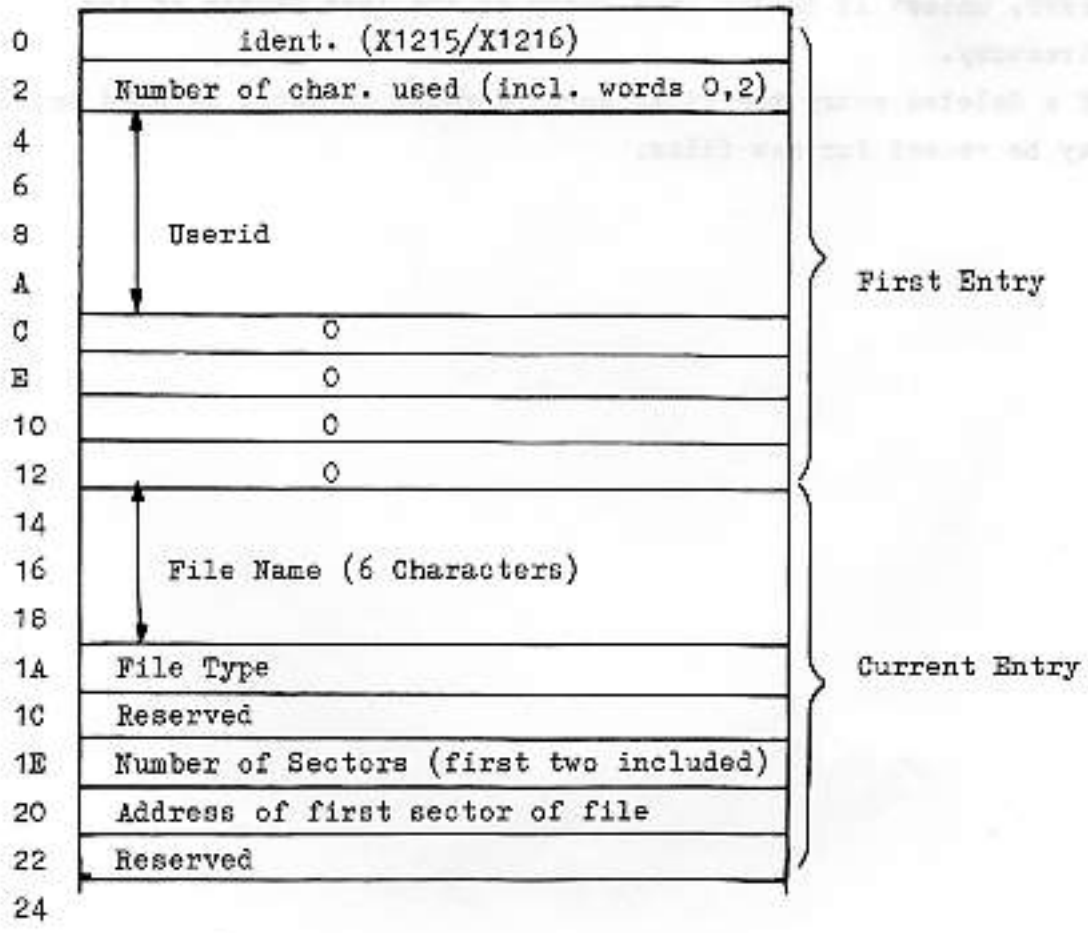
where:

- in a deleted entry the first word of the user identification is set to zero.
- the last entry of the catalogue is follwed by a word containing /FFFF.
- When a new entry is declared, the system will first check if a deleted entry can be used, before taking a free entry at the end of the catalogue.
- the number of used characters (word 2) includes the first two words of the directory and word /FFFF.
- sectors 6 and 7 are not used.
- if sector 5 is completely full, /FFFF is not written.

Remaining granules are used for the User Directory or for User Files. The status of the granules is given in the BITAB which is updated each time a file is catalogued or removed from the directory.

User Directory

A user directory occupies one granule and contains the names and characteristics of all the catalogued files belonging to the same user. One DAD may contain several users, as many as the Catalogue allows.
For each version of a file one entry is created. This is done when the file is catalogued. When the file is deleted, the entry is removed.
The first entry of a directory contains the name of the user.
The layout of a user directory is as follows:



where:

- the first three words of an entry contain the file name, up to 6 ACSII characters long, left justified and space-padded.

- the fourth word indicates the file type, in 2 ASCII characters.
  OB: object file
  SC: source file
  LM: Load module
  UF: user data file
  EF: extended file (DRTM only).
- the sixth word is reset to zero for non-consecutive files, or it contains the number of sectors of the file (8 times the number of granules).

The last entry of the directory is followed by a word containing /FFFF, unless it is the last entry of the last sector of the directory.
Of a deleted entry the first word is reset to zero. Deleted entries may be reused for new files.

DISC STRUCTURE without DADs

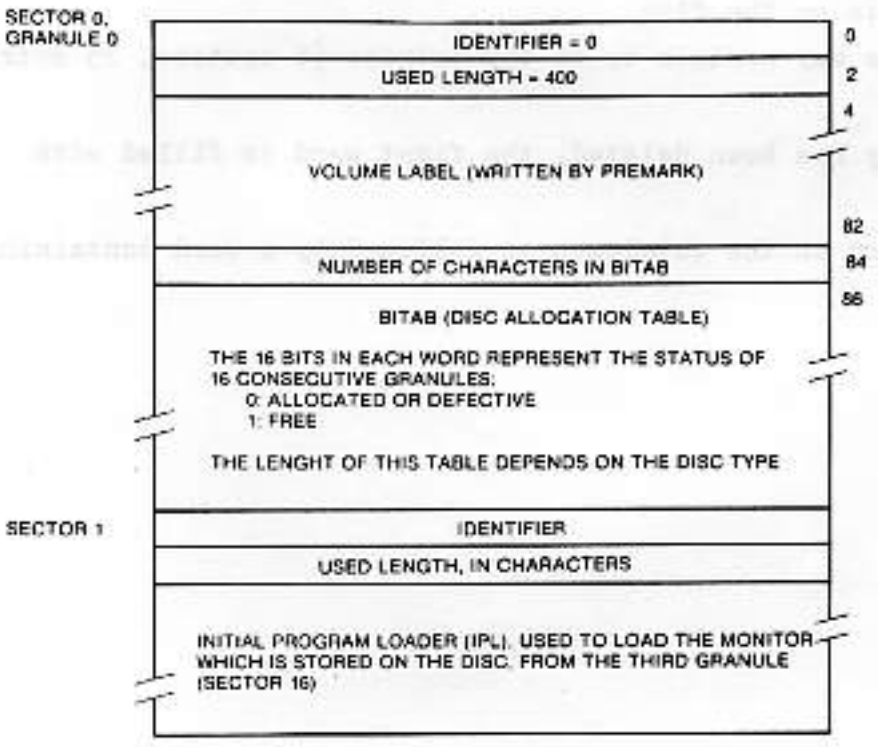
Catalogue and Library Structure

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.

An entry has the following format:

- words 0 to 3 contains the user identification

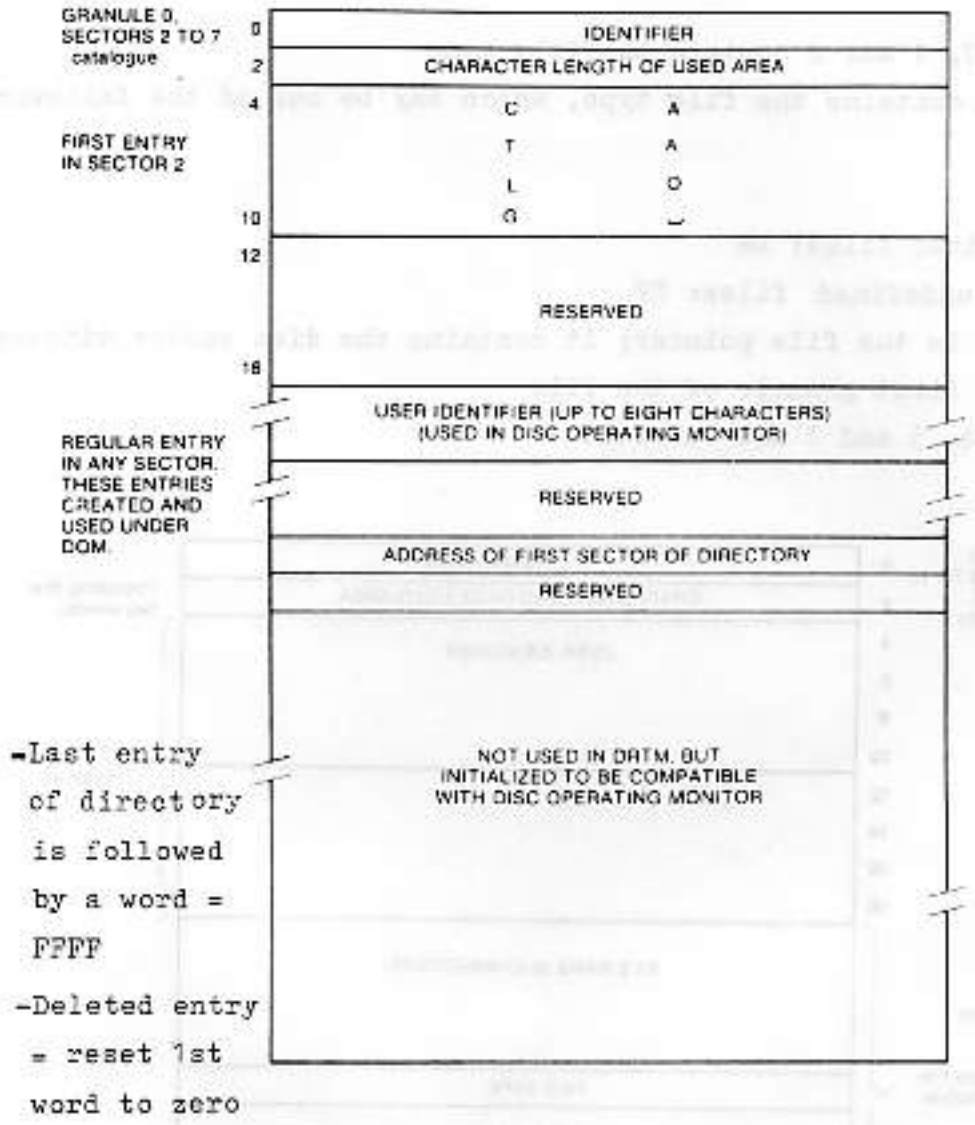
- words 4 and 5 are not used
- words 6 contains a pointer to the user directory; it is the disc sector address of the granule containing the user library directory.
- word 7 is reserved.

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.

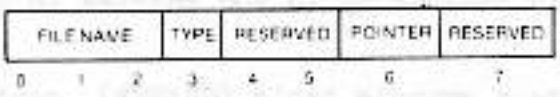
The last entry in the catalogue is followed by a word containing /FFFF.



Format of user catalogue (sectors 2 to 7) -

Directory Structure

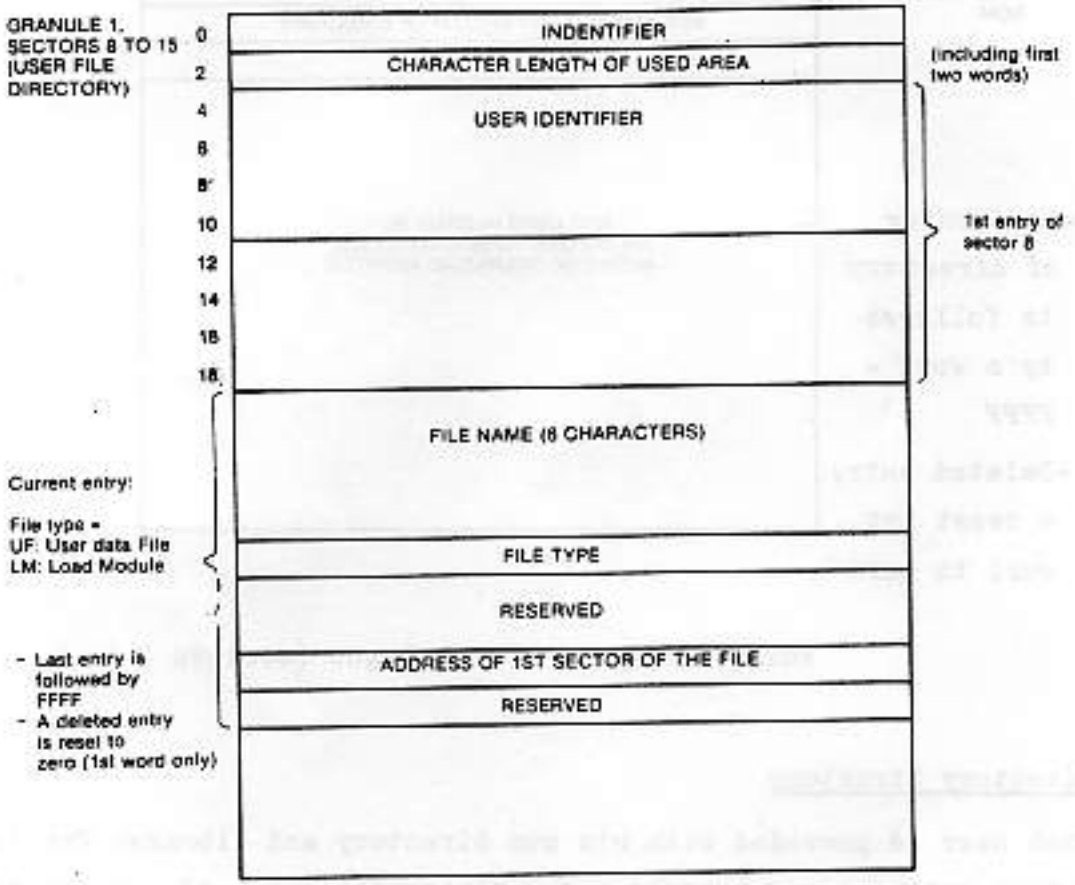
Each user is provided with his own directory and library. The directory occupies one granule and contains the names of and pointers to the user's files. The granule containing the user directory may be located anywhere on the disc, except when the user is SYSTEM. In this case it is the second granule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

Temporary File

These files are always of the type UF. They have to have a file code assigned to them either by monitor request or by Control Command. The access method for these files is defined as follows:

a) For random files, when a file code is assigned to the file, the required number of granules will be allocated on the disc.
b) The access flags (in the Logical File Description Table LFT) will be reset to zero.
c) One of these will be set when the first attempt is made to access the file, depending on the access method used:
   - If sequential access is used, any additional granules which may be required are allocated dynamically.
   - If direct access is used, the file is considered consecutive and no more additional granules will be allocated. Therefore, for this access method all the required granules must be reserved by the user program.

These temporary files may be made permanent by means of a Keep File control command.

Permanent Files

The names of these files are stored in a directory, of which there is one on each disc. Permanent files cannot be expanded, even if they are sequentially organized.
When a file code is assigned to a file, the system will check whether its granules are consecutive or not. If they are, random as well as sequential access may be used for this file. If they are not, only sequential access is allowed. The first attempt to access the file will set the access flag and thus define the access method for this file. This cannot be changed unless a new file code is assigned to the file. To remove a permanent file from the directory, the control command Delete File is used.

DRTM System Disc Structure

The System Disc contains all disc resident components required by the running system. The supervisor part of the monitor is memory resident, so does not have to be stored on the disc. However, if stored on disc, it must be stored on consecutive sectors, starting from sector /12, in load module format.
The System Disc must therefore have the following structure:

- Sector 0: Volume Label
- Sector 1: Initial Program Loader (IPL)
- Sectors 2 to 5: Catalogue (only one entry is used under DRTM)
- Sectors 8 to F: Disc Directory
- Granules 3 to n consecutively contain the supervisor if it is stored on the disc.

The structure of the System Disc is therefore compatible with that of the Disc Operating System (DOS).

The following components must be resident on the System Disc:
(DRTM Supervisor)

- D:USV1 (user monitor requests)
- D:USV2 (user monitor requests)
- D:USV3 (user monitor requests)
- D:OCOM (operator communication)
- D:DUMP (system debugging facilities: DM, WM, DD)
- D:ROOT (system command language)
- D:SEG1 (system command language)
- D:SEG2 (system command language)
- D:SEG3 (system command language)
- D:SEG4 (system command language)
- D:CI (core image file, used to contain core images of all swappable programs)

All these files, except D:CI must be catalogued before loading the monitor. The D:CI file, used to contain the core images of the swappable and read only programs, may be declared at system generation time on the system disc or on any other disc in the configuration. It will be catalogued automatically when the system is loaded for the first time, so it is not necessary for the user to catalogue it with a Keep File command.

FLEXIBLE DISC

The flexible disc can be accessed as a physical device with the following characteristics:

- random access at sector level
- the length of a physical sector is 128 characters, all of which are available for data
- the length of a logical record may be from 1 to 512 characters. If a record comprises more than one physical sector, it is defined by the physical address of the first sector in the record. The number of sectors (1, 2, 3 or 4) is determined by the driver on analysis of the record size.
- One track contains 26 sectors of 128 characters. These sectors are not interlaced.
- A sector is defined by a sector number from 0 to 2001. This is the sector number given in ECB Word 5 for I/O operations.
- Bad tracks are handled automatically by the driver.

Physical and operational details about flexible disc and flexible disc drive unit are described in the P800M Operator's Guide.

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.

FILE CODES

Assignment is done by monitor request or control command. Some file codes are generated at system generation time, particularly those used by the monitor. The others may be assigned dynamically.
The file codes used by the DRTM system are:

02: used by the Dump routine, to output the disc or memory contents.

CF to Cx: DAD file codes used by system and defined at sysgen from CF down towards C0 for as many DADs as defined.

E0: used to read the SCL control commands from.

EF: used for operator communication.

F0 to FF: used for the physical disc units.

For Device Addresses, see Appendix I.

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 (AS control command) and writing onto it by running a program.

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 1-33.
The second sector of each file contains a granule table (GRANTB) with the addresses of all the granules of this file. See page 1-33.
Two types of file exist:

- load module (LM): only direct access possible
- undefined (UF): sequential and direct access possible.

SEQUENTIAL ACCESS METHOD

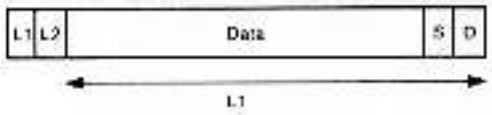
A file is sequential when the only relation between the different records is their sequence. When such a file is created, the records must be presented in the same order in which they must be written onto the disc. To access the file, it must be scanned sequentially until the desired record is found.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

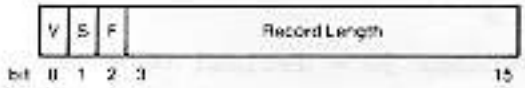
Logical Record Format

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:

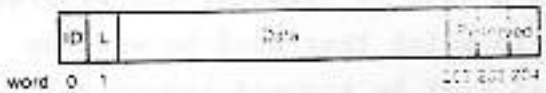
L1 is the record length, including the words S and D, but not including L1 and L2:



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: $\underbrace{4004}_{L1}$ - 0 - S - D

:EOF: $\underbrace{2004}_{L1}$ - 0 -S - $\underbrace{4}_{D}$

: Blank card: $\underbrace{0004}_{L1}$ - $\underbrace{0050}_{L2}$ - S - D (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 KF control command. 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 KF 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 AS 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.

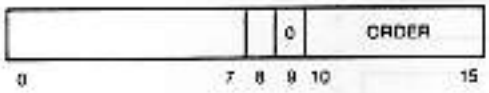
Data Management Requests

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)

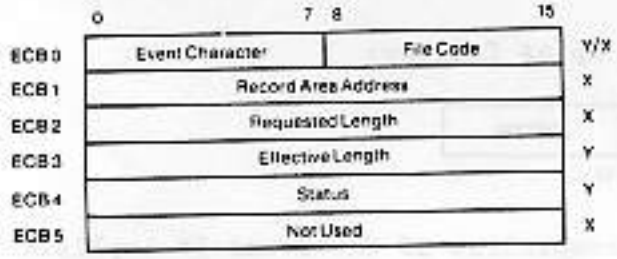
/05: Write a Record (Basic)

/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 (disc physical files F0 to FF) or DL (disc logical files) will be returned in ECB word 1 and the word LFTMOD1 (see Logical File Table, part 2) in ECB word 4; 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:



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 1-110). 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.

For DRTM, the requested number of granules is allocated straight away and extension of this number during file creation is not possible.
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 KF 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).

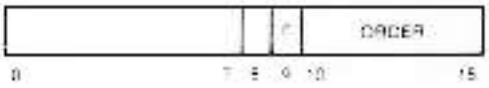
Data Management Requests

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:

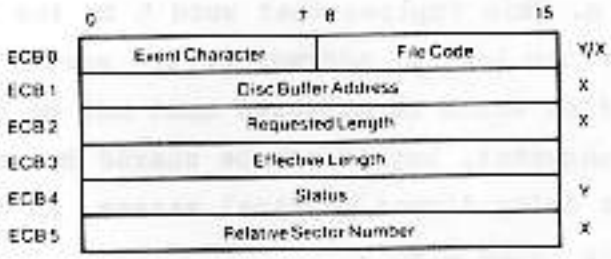
/0A: Read (Random)

/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 and the word LFTMOD1 (see Part 2, Logical File Table) in ECB word 4. 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: Remains unaffected.

ECB4: This word contains the status returned to the user program by Data Management when the sector transfer is terminated. See page 1-110. 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 and can only be be used on DADs. The Data Management module is not used for this type of access, but special orders are used in the LKM 1 monitor request:

/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 disk 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'.