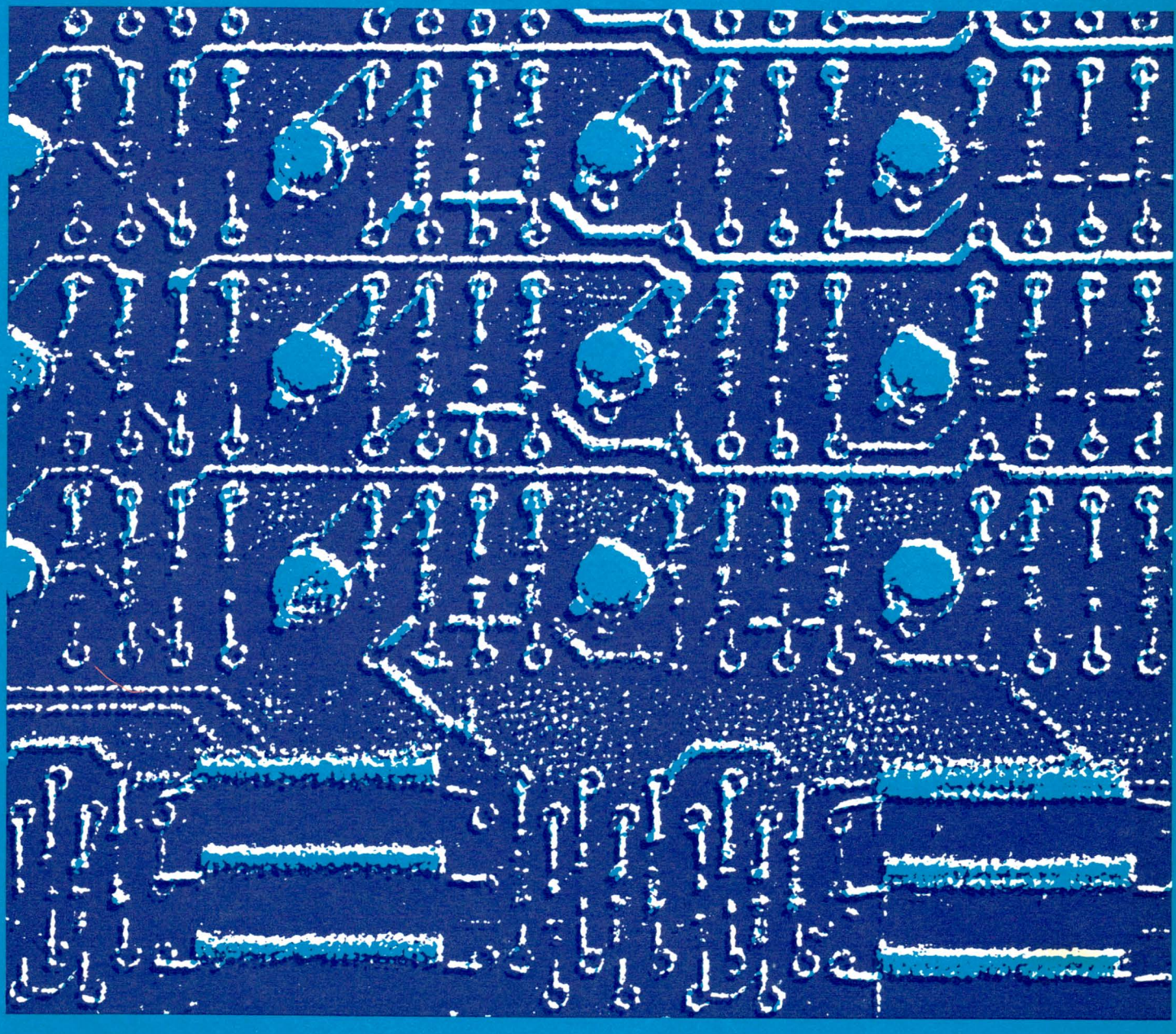
P800M Programmer's Guide 1

Volume VI: Cassette Operating System

preliminary



PHILIPS Data Systems

PHILIPS

P800M Programmer's Guide 1

Volume VI: Cassette Operating System

preliminary

A publication of

Philips Data Systems B.V.
Marketing Group Small Computers
Apeldoorn, the Netherlands

Publication Number 5122 991 27361

March, 1976

Printed in the Netherlands.

Preface

This manual contains a description of all the software necessary to use the P800M Cassette Operating System, a cassette tape-oriented system which includes cassette file management according to ECMA standards.

Other books to be used in conjunction with this manual are:

P800M Programmer's Guide 1 & 2, Vol. II (Instruction Set)
P800M Software Reference Data
P800M Full FORTRAN Reference Manual
P800M Operator's Guide
P800M Interface and Installation Manual

Great care has been taken to ensure that the information 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 book, he is invited to send his comments, written on the sheet provided at the end of this book, to:

Manual Writing Small Computers

at the address on the opposite page.

The Cassette Operating System is a cassette oriented software system on which programs, written in assembly language or FORTRAN, can be developed and executed.
The Cassette Operating System is stored entirely on magnetic tape cassettes, and it uses cassette tapes as its main input/output medium. The cassette tapes used by the Cassette Operating system have two tracks, with a capacity of 338,000 characters each, on which data are recorded serially with a density of 800 bits per inch. For read and write operations, the cassette drive unit moves the tape at a speed of 7.5 inches per second.

The main feature of the Cassette Operating System is the Cassette File Management Package, which is part of the monitor. The Cassette File Management Package handles files (i.e. source, object or load modules or sets of data) on cassette tapes according to the three labelling systems recommended in the ECMA standards: basic, compact and extended. In the basic labelling system one or more files can be recorded on one volume, the compact and the extended labelling system can handle multi-volume files and multi-file volumes.

The Cassette Operating System consists of the following components:

- Cassette Operating Monitor
- Assembler
- Linkage Editor
- FORTRAN Compiler, Transcoder and Library
- Cassette Update Package
- Debugging Package
- utility programs

Cassette Operating Monitor

The Cassette Operating Monitor supervises loading and execution of processors and user programs. It consists of several modules, which are defined at system generation time and include facilities for communication with the user, cassette file management and I/O drivers.

Assembler

The Assembler converts source modules written in assembly language into object modules suitable for linking to other modules or for loading and execution.

Linkage Editor

The Linkage Editor links separate object modules, either for direct loading and execution or for output, to be loaded later or used in a further linkage process.

FORTRAN Compiler, Transcoder and Library

The FORTRAN Compiler translates source modules written in FORTRAN into object modules to be processed by the Linkage Editor.
The FORTRAN Transcoder translates FORTRAN object modules into machine code instruction modules, which also must be processed by the Linkage Editor.
The FORTRAN Library contains routines which must be linked to FORTRAN object modules or transcoded FORTRAN modules, to produce an executable FORTRAN program.

Cassette Update Processor

The Cassette Update Processor enables the user to copy, delete, insert and list files and modules, and to insert or delete lines in files and modules.

Debugging Package

The Debugging Package enables the user to stop the execution of a program at specific points, so that the contents of memory locations and/or registers can be checked or changed.

Utility Programs

The utility programs available in the Cassette Operating System are the Initial Program Loader (IPL), which can load programs independently, a dump program which produces an ASCII memory dump on the line printer or the operator's typewriter, and a premark program, which premarks cassette tapes to be used with the Cassette File Management Package.

The Cassette Operating System is delivered on two generation cassettes, which contain a set of system generation programs, a library of monitor modules, and the software processors. The user creates his own system, by selecting and linking the required monitor modules and copying the required processors.

The minimum configuration required for the Cassette Operating System is a CPU with 16k words of memory, an ASR typewriter, and two cassette drive units.
This configuration can be extended with other peripherals, such as additional cassette drive units, a high speed punched tape reader, a line printer, etc.

PART 1 MONITOR

1 Principles of Operation

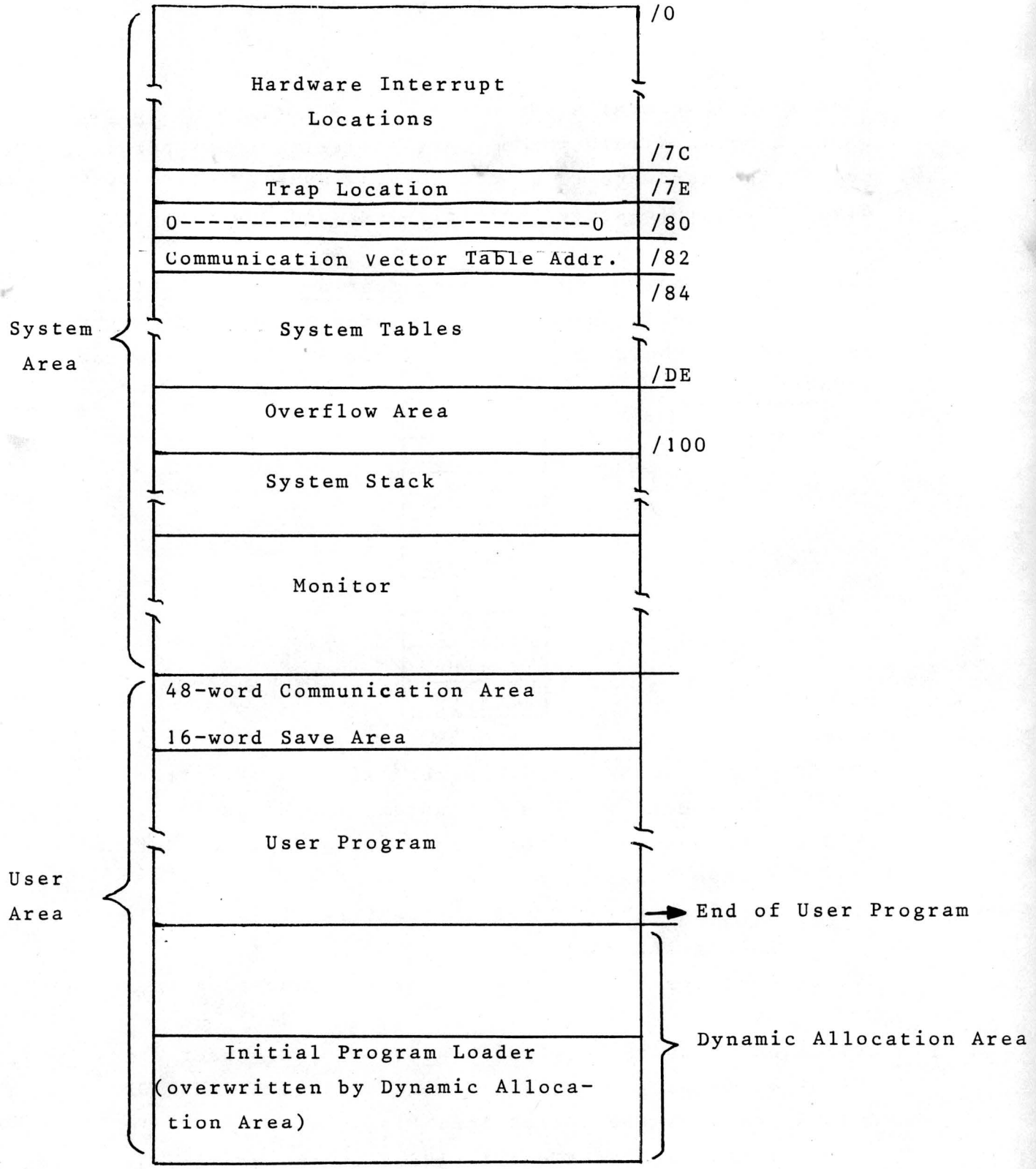
The Cassette Operating Monitor (COM) is a monitor handling one program at a time, to be applied basically as a cassette tape oriented program development tool.
Its structure is modular, to allow the user to select, at system generation time, those monitor parts which he will need for his application, so as to increase efficiency and minimize memory occupation.

At system generation time, the user creates his own system cassette, on which the monitor is the first program, preceded by an Initial Program Loader (IPL). The monitor is loaded by this IPL, which is loaded by bootstrap according to the data switches on the CPU control panel. Details on this procedure can be found in the chapter on Operation.
Then the user program or a processor is loaded. If the operator communication package is used for this purpose, separate commands must be given to load and start. If the cassette file management package is included, one single command will be enough to seek, load and start a program.
The cassette file management package is used to handle I/O operations on cassette tapes according to ECMA Standard 41, which relates to the type of labelling of the tape. The CFM will accept three types of increasing complexity: Basic, Standard and Extended. The system software is given in Compact type of labelling, which allows the handling by the COM of single-track, multi-track and multi-volume files as well as multi-file tracks and volumes. Files are preceded by headers and followed by EOF records; in case of file continuation tracks end with End-Of-Track records, volumes (i.e. one complete cassette) with End-Of-Volume records. This is all handled automatically by the cassette file management package; related to it are a number of operator commands by means of which the user can write or search headers, run a program, etc.
The monitor itself handles the standard interrupt signals and I/O operations and executes functions requested by the user in his program by means of monitor requests, e.g. requesting and releasing temporary buffer space in memory, waiting for

events, making exits, etc.
The monitor modules are centered around a dispatcher, which determines on the basis of interrupt signals and priority levels which routine or program must be executed.
Although the COM is designed to handle one program at a time, a form of multi-tasking can be achieved by using scheduled label routines. These routines are attached to the specification of a monitor request and enable a program to run concurrently with, for example, an I/O operation.

Although the COM is cassette tape oriented, other peripherals can be handled as well. This can all be determined by the user at system generation time, as explained in the chapter on System Generation.

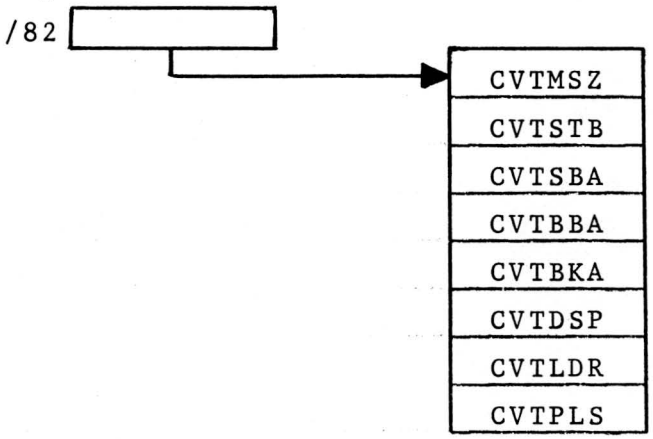
The memory layout is as follows:



- Locations /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, i.e. 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 store, multiple load and double shift).

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



where:

CVTMSZ contains the machine memory size in characters. If the memory size is 32k words, the value is 0.

CVTSTB contains the system stack base address as defined at system generation time.

CVTSBA contains the address of the first free location following the user program.

CVTBBA contains the address of the last location in the user area.

CVTBKA contains the beginning address of the user area.

CVTDSP contains the address of the dispatcher (M:DISP).

CVTLDR contains the option load flag:
1 if the loader was included in the monitor at sysgen
0 if it was not

CVTPLS contains the clock pulse which is normally 1 (20msec. clock).

- 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, PSW and P-register are stored here by hardware and a number of registers by software. The number of registers 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 remaining after the user program area is reserved for dynamic memory allocation after termination of all loading procedures. From this area, blocks of memory space may be requested by the system or by the user. The user must send a 'Get Buffer' monitor request for this purpose. When he does no longer need the buffer, he must send a 'Release Buffer' monitor request.

Programs and routines under the Cassette Operating Monitor run on the basis of an interrupt and priority system which consists of up to 64 levels. These are subdivided as follows:

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

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

HARDWARE INTERRUPT LINES

The interrupt lines are connected to memory locations /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 level at system generation time.

The following priorities are strongly recommended for the various interrupt lines, because at system generation they are assumed to be standard. The user can define the levels differently during the COMGEN phase of sysgen, if he wishes:

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

DISPATCHER

The dispatcher is a monitor module (M:DISP) running on level 48 with the other interruptable monitor service routines, 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 or equal to 48, such as the I/O interrupt or monitor request handlers.

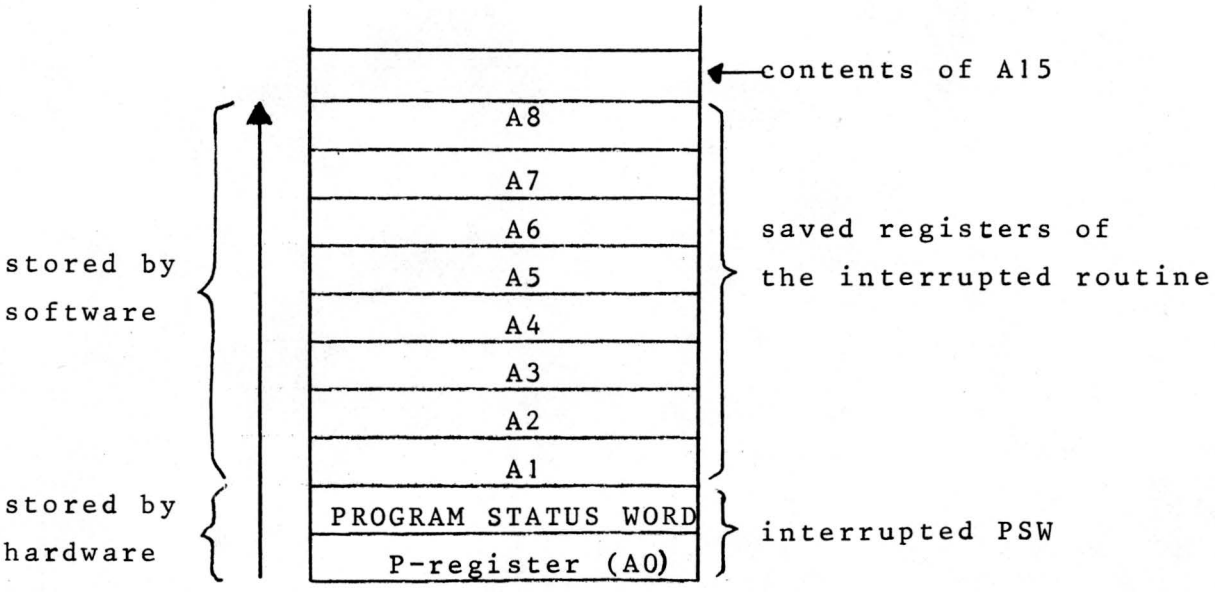
SOFTWARE PRIORITY LEVELS

The user program is connected to priority level 62. It is activated after loading, by the operator command ST or automatically ifit was loaded by the cassette control command RN. The operator communication package, handling the operator commands and the abort module and idle task also operate on software priority levels.

SYSTEM INTERRUPT STACK

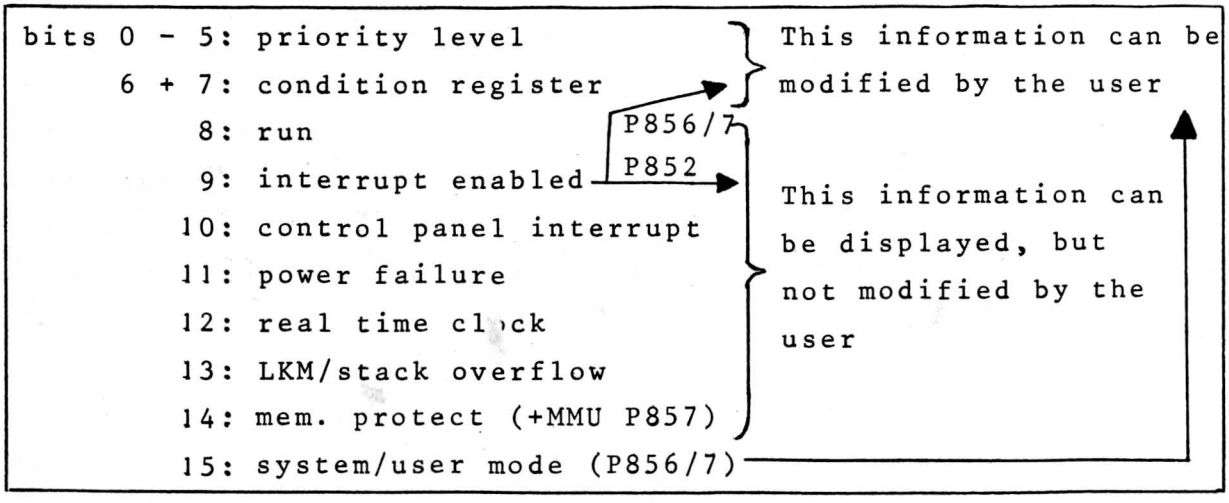
When an interrupt occurs, certain information about the interrupted program or routine must be saved before the interrupt can be serviced.
For user programs, this is done in the system stack pointed to by register A15 or, when there is a priority change from one program to another, in a 16-word save area reserved in front of each program, where P-register, PSW and registers A1 to A14 are stored.
For interrupt routines it is done in the system stack.
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 memory 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 moreover a number of registers:

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



When the stack pointer (A15) reaches the value /100 or when it becomes lower, an interrupt will be given, for /100 is the last location before the stack overflow area.

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



The PSW can be displayed on the control panel.

There are three types of user-written programs:
- programs (user-written coding sequences), connected to software level 62
- subroutines, called through a CF (Call Function) instruction and running on the level of the calling program
- interrupt routines, servicing one of the 63 possible hardware interrupts and connected to any of the hardware levels 0 to 47.

SOFTWARE LEVEL PROGRAMS

These are the user programs, connected to level 62. They are loaded in the user area, behind the monitor in memory. When such a program is loaded into memory, the system reserves a communication area of 48 words and a save area of 16 words in front of the user program. The communication area can be used by the user to store information and the save area is used by the system to store the program's register contents in case of a scheduled label interrupt in the main program.
At the start of the user program, register A1 contains the address of the first location of the user program area and register A2 contains the first free location following the user program.
Monitor requests allow the user program to request certain functions from the monitor by means of an LKM instruction followed by a DATA directive with a number to specify the function. Certain parameters may have to be loaded into the A7 and/or A8 registers first. Monitor requests may be combined with scheduled labels (see below).

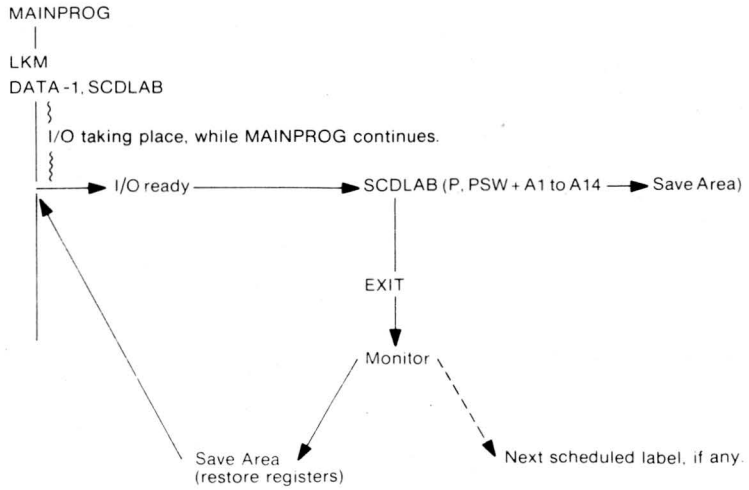
SCHEDULED LABELS

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 is to be executed upon completion of the request. For example:

```
normal I/O request:   LDK    A7,CODE
                      LDKL   A8,ECBADR
                      LKM
                      DATA   1
with sched. label:    LDK    A7,CODE
                      LDKL   A8,ECBADR
                      LKM
                      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 attachedto 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 of the main program are stored in the 16-word save area in front of the user program.
When entering the SCDLAB routine, no register value is significant, so it is not possible to pass parameters to the scheduled label sequence. 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 called FILLAB. The maximum number of scheduled label addresses which may be stored in this table at any one time is defined at system generation time.
Queued scheduled labels are treated on a first-in-first-out basis.

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

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. Another method, providing a relocatable routine, is shown in the following example:
A routine, labeled INTRO is to be connected to interrupt level 10, so the starting address of the routine has to be loaded into memory location /20:

```
        IDENT   INTRO
          |       |
          |       |
        LDKL    A1,ROUT
        ST      A1,/20
          |
          |
ROUT      |                          }
          |                          } INTERRUPT ROUTINE
          |                          }
          |
        END
```

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 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 that 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 to A14 can be used, provided the user first takes care of storing their old contents in the system 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
 |      |             selected at sysgen for P852, the sequence is:
 |      |
STR   A8,A15          MSR  8,A15
 |                        |
 |                        |
coding                  coding
 |                        |
 |                        |
LDR   A8,A15          MLR  8,A15
LDR   A7,A15          RTN  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

Communication Vector Table), 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 by the dispatcher.

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 registers A1 to A8, so any other registers used must have been restored before making the branch.
- In case of an interrupt routine for an internal interrupt (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.

DYNAMIC MEMORY ALLOCATION

The user can make temporary use of the dynamic allocation area of memory if his monitor contains the modules handling the following two monitor requests:

- Get Buffer, to reserve a user buffer in the dynamic allocation area;
- Release Buffer, to release the area created by the Get Buffer request.

Memory allocation is done in blocks, the block length being specified by the user. Preceding each block is a control block containing links and other parameters which must not be destroyed.
For further details, see the chapter on Monitor Requests.

All I/O operations initiated by the user program are the result of giving an I/O monitor request (LKM DATA 1). Other I/O operations may be the result of operator commands or cassette file management control commands.
At system generation time, the necessary modules and tables must have been requested and built to enable the use of the I/O monitor request.
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 three types of I/O request as specified in register A7:

- Basic I/O Requests: for these requests the monitor will not do any character checking or data conversion, so they are used in the 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.
- Standard Object I/O Requests: by means of standard conventions facilities are provided for error control characters, checksum and data conversion from external format to internal 16-bit format.

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. Depending on the type of labelling used for the cassette tape, the number of facilities offered in this respect varies for each type of labelling, basic, compact or extended. See also the chapter on Cassette File Management and the description of the I/O monitor request.

In the Event Control Block, pointed to by register A8, the user specifies the file code (logical address) of the device 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 concerning the result of the operation in this ECB, so that it may be verified by the user program.
In the Appendix Peripheral I/O, details are given about the different types of I/O for different devices.

The file codes are a means of giving a logical address to a file or a peripheral unit. They are defined with the physical device addresses at system generation time.
The device addresses consist of two letters followed by two digits. The standard device addresses are:

- operator's typewriter:	TY10	
- punched tape reader:	PR20	
- tape punch:	PP30	
- line printer:	LP07	
- card reader:	CR06	
- magnetic tape cassettes:	TL05	(TL means I/O via the
	TL15	CFM package. If stan-
	TL25	dard I/O: TK05,etc.)
- magnetic tape:	MT04	
	MT14	
	MT24	
	MT34	

The file codes consist of 2 hexadecimal digits from /01 to /FF. The following are standard file codes:

- source input (TL05):	/01
- listing output (LP07):	/02
- punch output (TL15):	/03
- object input (TL25):	/04
- typewriter I/O (TY10):	/05

These file codes are the default values in the operator control commands (see chapter 7) and are used by COM as follows:

- file code /01 is used by the COM and the processors. COM uses it in the following Operator Control Commands: SH (Search Header) and CL (Clear File).
- file code /02 is used by the processors. The COM uses it for dump memory.
- file code /03 is the standard output file for the processors and the COM. The COM uses it in the Operator Control Commands WH (Write Header), WF (Write First Header).
- file code /04 is used by the COM to load programs. This file code is normally assigned to a cassette tape. The COM uses this file code for the Operator Control Command RN (Run Program) and the operator command LD (Load Program).
- file code /05 is used by the COM and the processors. The COM uses it for the input of commands and the output of system messages.

The remaining file codes, up to /FF (or the maximum declared at system generation time) can be assigned by the user to any other files or devices, either at system generation time or later on by means of the operator command AS, by means of which the file codes can be reassigned as well.
The file codes are contained in a system table, the File Code Table (T:FCT), containing the addresses of the Device Work Tables (DWTs) of the peripheral devices. File codes assigned to TL devices are contained in the Logical File Table (LFT). The place in the table determines the file code, i.e. the first address in the table gets file code /01, the second /02, etc.

ECMA Standard Access

I/O operations are performed through the standard I/O package of the monitor or through the Cassette File Management Package (CFM). If it is done through the CFM package, the cassette tape units used must have been declared as TL (device name) units at system genration time. If they have been declared as TK units, an AS cassette control command must first be given to allow ECMA standard access. Then the cassettes can be accessed as labelled according to the Basic, Compact or Extended system of labelling. For each of these labelling systems, there are different access possiblities, which are described under the I/O monitor request (LKM DATA 1), where the standard I/O facilities are also described. To enable ECMA standard access the cassette tape must also have been premarked for one of the three systems of labelling.

6 Cassette File Management Package

The Cassette File Management Package handles input/output operations on cassette tapes according to the ECMA standards. The package uses the three systems for data recording recommended in the ECMA standards:the basic, the compact and the extended labelling system.
The basic labelling system uses tape marks (i.e. blocks containing two characters of 8 zero bits) to delimit files, tracks and volumes.
The compact and the extended labelling system use labels to identify, characterize and delimit files, tracks and volumes. For the compact labelling system a label is a 32-character block, for the extended labelling system the label length is 80 characters.
In all three systems , files may be recorded on a single track, or split up in file sections recorded on consecutive tracks. In the basic labelling system files must be recorded on one or two tracks of one volume, in the compact and the extended labelling system files may be recorded on one or more volumes.

The input/output operations of the Cassette File Management Package are initiated by I/O requests to the monitor for the basic labelling system, and by I/O requests or operator control commands for the compact and the extended labelling system. The I/O requests are described in detail in chapter 8, the operator control commands in chapter 7.

The Cassette File Management Package can only be used for those cassette drive units which have been specified as 'CFM units' during system generation. Cassettes which are to be used with the Cassette File Management Package must have been premarked previously for one of the three labelling systems by means of the premark program. The premarking procedure is described in part 9.

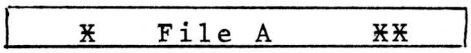
BASIC LABELLING SYSTEM

The basic labelling system can handle one or more files on one volume. The system uses single tape marks to separate files, and to indicate start of track and end of track, and a double tape mark to indicate end of file. When a new file is written after a file closed previously with a double tape mark, the second of the two tape marks is erased, so the end of file mark changes into a file separator.
The tape mark at the start of a track is written when the cassette is premarked for the basic labelling system, all other tape marks must be written by the user program.

File structure

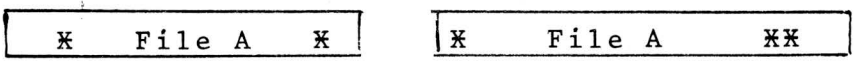
In the examples of file structuring shown below, a tape mark is indicated by ¤, and a double tape mark by ¤¤. Each box represents one track of a volume.

One file on one track



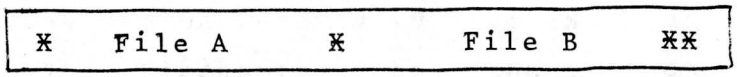
The file starts with a single tape mark indicating start of track, and ends with a double tape mark, which indicates end of file.

One file on two tracks



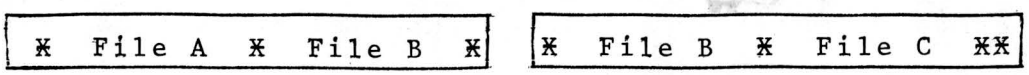
When a file extends over the end of the first track of a volume, the status 'end of tape' is sent to the user program when the end of tape marker at the end of the track is detected. The track must be closed with a single tape mark for intermediate end of track. When track B has been loaded, the read/write head is positioned after the first tape mark, and the read or write operation can continue. The file must be closed with a double tape mark, specifying end of file.

More_than_one_file_on_one_track



If one track contains more than one file, the files are separated by a single tape mark. The track starts with a single tape mark, and ends with a double tape mark, as in the case of a single file on one track.

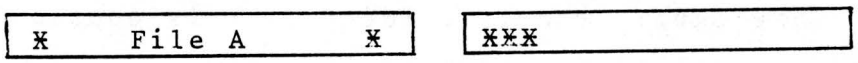
More_than_one_file_on_two_tracks



On a volume with more than one file on two tracks, the files are separated by single tape marks. Track A must be terminated with a single tape mark after the end of tape marker has been detected. Track B starts with a single tape mark, and the last file is closed with a double tape mark to indicate end of file.

Coincidence_of_end_of_file_and_end_of_tape

On each track, an end of tape marker indicates the end of the track. If the end of tape marker is detected when the last block of a file is written, the track must be terminated with a single tape mark. After the tape mark already present at the start of the next track, a double tape mark must be written to indicate end of file. The resulting file structure is shown below.



If a new file is written on the second track, the last of the three tape marks must be erased, so the second track will start with one tape mark for intermediate start of track and one file separator, as shown below.

COMPACT LABELLING SYSTEM

The compact labelling system can handle one or more files on one or more volumes. The system uses labels to identify and recognize files, and to indicate end of track, end of volume and end of file. Tape marks are used to separate labels from file data and from other labels.

Label types

In the compact labelling system four label types are used:

- file header label (HDR)
- end of track label (ETR)
- end of volume label (EOV)
- end of file label (EOF)

The general layout of the four label types is shown in the table on the next page.
In each label type, the label identifier is filled in by the system. The volume identifier is copied from the first file header label, written by the premark program. The user has to specify the file identifier for the file header label, for the other labels the file identifier is copied from the file header label of the current file.
The system only recognizes the first 13 characters of the label, the other fields must be handled by the user program. The layout of the fields 14-32 as shown in the table is the layout recommended in the ECMA standards.

File header label

The file header label is used at the beginning of a file, and at the beginning of a file section if the file is recorded on more than one track.
A file header label can be written by means of the operator control command WH (write file header label), or through an I/O request with order /23 (write header). The label identifier is filled in by the system, the volume identifier is copied from the first file header label of the volume, and the file identifier field is filled with the file name specified by the user in the control command or in a buffer of which the address is specified in the I/O request.
For files split up in a number of file section, the user only has to specify the file identifier for the file header label of the first file section. The file identifier for the file header labels of the following file sections is copied from the label at the beginning of the first file section.
To search a file header label, either the operator control command SH (search file header label) or an I/O request with order /27 (search header) can be used. If the system has been put in 'enable access for labels' mode by a previously issued I/O request with order /29 (enable access for labels), the contents of the file header label are read into the label buffer specified in that request.

End of track label

The end of track label indicates that the current file extends over the end of track A of a volume. When during a write operation the system recognizes the end of tape marker of track A, the write operation is stopped temporarily, and the system writes an end of track label. The label is filled by the system with the proper label identifier, all other fields are copied from the file header label of the current file.

character position	field name	length	contents
1	label identifier	1	'1' (/31) for HDR '3' (/33) for ETR '7' (/37) for EOV '9' (/39) for EOF
2-5	volume identifier	4	volume name
6-13	file identifier	8	file name
14-15	file section number	2	two-digit number, specifying the sequence number of the file section
16-20	creation date	5	two-digit number specifying the year, followed by a three-digit number for the day of the year
21-23	retention period	3	three-digit number for the number of days the file must be kept
24-27	block count	4	0000 for HDR four-digit number for ETR, EOV and EOF, equal to the number of blocks in the preceding file or file section
28	data code indicator	1	1
29-32	reserved	4	reserved for user applications

After the end of track label has been written, the system sends the message xx END, in which xx is the device address of the cassette drive unit on which end of tape has been detected, to the operator's typewriter. The write operation of the user program continues when track B has been loaded. If during a read operation the system detects an end of track label, the message xx END is printed. The read operation continues after track B has been loaded.

End of file label

The end of file label indicates end of data in a file.
An end of file label is written by means of an I/O request with order /22 (write end of file label). The label identifier is filled in by the system, the volume identifier is copied from the first file header label of the volume, and the file identifier is copied from the file header label of the current file. If the system runs in 'enable access for labels' mode, the other fields of the end of file label are filled with the contents of the label buffer.
The end of file label is preceded by a single tape mark,and followed by a double tape mark. If a new file is added after the end of file label, the second of the two tape marks is erased.
When an end of file label is read, bit 15 of the software status is set to 1. In 'enable access for labels' mode, the contents of the label are copied into the label buffer.

End of volume label

The end of volume label indicates that a file extends over the end of track B of a volume.
An end of volume label is written when the system detects the end of tape marker on track B during a write operation.
The write operation is stopped temporarily, the system writes the end of volume label, and the message xx END is written on the operator's typewriter. The system writes the proper label identiefier, the other label fields are copied from the file

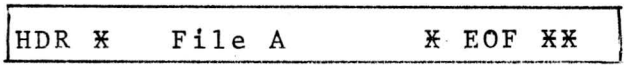
header label of the current file. The label is preceded by a single tape mark, and followed by a double tape mark. Control is given back to the user program when a new volume has been loaded and is ready to receive data.
When an end of volume label is read, only the message xx END is printed. Reading of data continues when a new volume has been loaded.

File structure

The structuring of the files is handled by the Cassette File Management Package, but the user must write the file header label at the beginning of a file, and the end of file label at the end.
In the examples of file structuring shown below, a tape mark is indicated by ✱, each box represents one track of a volume, and the labels are represented by the following mnemonics:

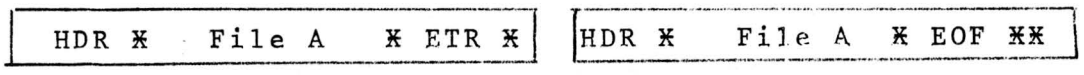
HDR - file header label
EOF - end of file label
ETR - end of track label
EOV - end of volume label

One file on one track



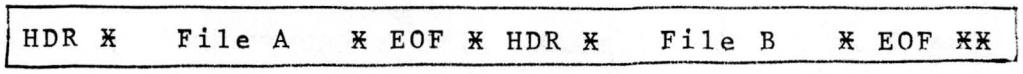
The file starts with a file header label, followed by a single tape mark. The file is closed by an end of file label, preceded by a single tape mark, and followed by a double tape mark indicating end of data.

One file on two tracks



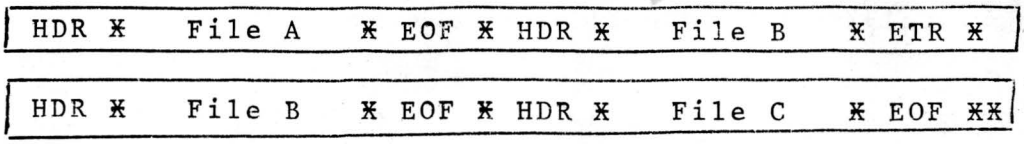
When a file extends over the end of a track, the file section on the first track is closed with an end of track label. The second file section starts with a copy of the file header label, in which the file section number field can be used to specify the sequence number of the file section.

More_than_one_file_on_one_track



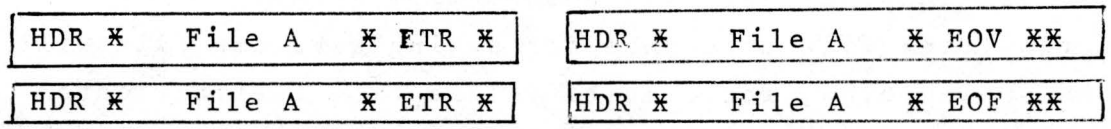
When one track contains more than one file, each file starts with a file header label and ends with an end of file label. Only the last end of file label is followed by a double tape mark.

More_than_one_file_on_two_tracks



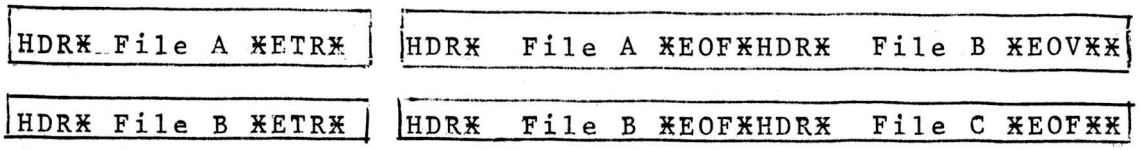
The first file, which is completely on track A, starts with a file header label, and ends with an end of file label. The first file section of the second file ends with an end of track label, the second file section on track B starts with a file header label, and it is closed with an end of file label. The last file of the volume ends with an end of file label followed by a double tape mark.

One_file_on_more_than_one_volume



When one file is recorded on more than one volume, each track starts with a file header label. Track A of each volume ends with an end of track label, track B of the first volume ends with an end of volume label, and track B of the second volume ends with an end of file label.

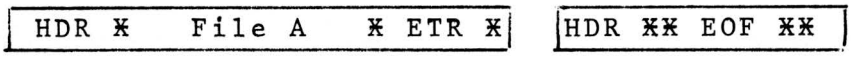
More_than_one_file_on_more_than_one_volume



When several files are recorded on more than one volume, each file and each file section starts with a file header label. Track A of each volume ends with an end of track label, track B of the first volume with an end of volume label, and the last file on the last volume is closed with an end of file label, followed by a double tape mark.

Coincidence_of_end_of_file_and_end_of_tape

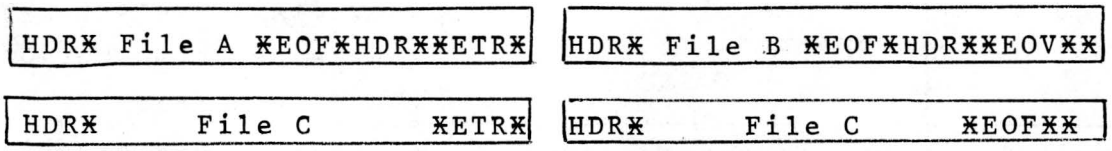
If the end of tape marker is detected when the system is writing the last block of a file,the track will be closed in the normal way with an end of track label if it is track A, or an end of volume label if it is track B. The message xx END, in which xx is the device address of the cassette drive unit, is printed on the operator's typewriter.
When the next track or volume has been loaded, the system writes the file header label of the current file, followed by a tape mark, and an end of file label preceded by a single tape mark and followed by a double tape mark.
The resulting file structure for a single volume file will be as shown below, for a multi-volume file the last data block will be followed by an end of volume label and a double tape mark.



If the end of tape marker is detected when the system is writing the end of file label after a file, the label will be written completely, and the file will be closed with a double tape mark.

Coincidence of beginning of file and end of tape

If the end of tape marker is detected when the system is writing a file header label, or if a new file header label is written on a track on which the end of tape marker was detected when the end of file label of the preceding file was written, the file header label will be followed by an end of track label or an end of volume label. The file header label and the end of track or end of volume label will be separated by a double tape mark, indicating a file section in which no data blocks are present. The next track starts with a copy of the file header label.
The resulting file structure for the coincidence of beginning of file and end of tape on track A and track B are shown below.



EXTENDED LABELLING SYSTEM

The extended labelling system can handle one or more files on one or more volumes, in the same way as the compact labelling system. The extensions to the compact labelling system are the greater label length (80 characters instead of 32), and two additional labels, the volume header label and the start of track label.

Label types

In the extended labelling system six label types are used:

- file header label (HDR)
- end of track label (ETR)
- end of volume label (EOV)
- end of file label (EOF)
- volume header label (VOL)
- start of track label (STR)

The layouts of the file header label, the end of track label, the end of volume label and the end of file label are identical, except for the label identifier. The general layout of these labels is shown below.

character position	field name	length	contents
1-3	label identifier	3	HDR for file header label ETR for end of track label EOF for end of file label EOV for end of volume label
4	label number	1	1
5-21	file identifier	17	file name
22-27	file set identifier	6	file set name
28-31	file section number	4	four-digit number, giving the sequence number of the file section
32-35	file sequence number	4	four-digit number specifying the sequence number of the file in a file set
36-39	generation number	4	four-digit number specifying the number of times the file has been created
40-41	generation version number	2	two-digit number distinguishing between file versions within one creation
42-47	creation date	6	one space character,a two-digit number for the year, and a three-digit number for the day of the year
48-53	expiration date	6	same as above
54	accessibility	1	alphanumeric character, indicating restrictions on file access, space character if no restrictions.

character position	field name	length	contents
55-60	block count	6	000000 for HDR six-digit number for ETR,EOF and EOV, specifying the number of blocks in the preceding file or file section
61-73	system code	13	alphanumeric characters, specifying the system that recorded the file
74-80	reserved for future standarization	7	space characters

The layouts of the volume header label and the start of track label are identical, except for the label identifier and the track number. The general layout of these labels is shown below.

character position	field name	length	contents
1-3	label identifier	3	VOL for volume header STR for start of track
4	label number	1	1
5-10	volume identifier	6	volume name
11	accesibility	1	same as in HDR,etc.
12-37	reserved	26	space characters for VOL, in STR cp.12=2 (track number)
38-51	owner identifier	14	owner name or identification
52-79	reserved for future standarization	28	space characters
80	label standard version	1	1 or 2, indicating the version number of the ECMA standard.

In all the labels, only the fields containing the label identifier, the label number , and the file or volume identifier are recognized by the system. The other fields must be handled by the user program.

Volume_header_label

The volume header label is used at the beginning of track A of a volume. The label is written by means of the premark program, when a tape is premarked for the extended labelling system. The label identifier field, the label number field and the volume identifier field are filled with the data specified during the premarking, all other fields are filled with zeroes.
The label is read by the system when the cassette is loaded into the cassette drive unit.
The volume header label must be followed immediately by a file header label, without a tape mark separating the two labels.

Start_of_track_label_

The start of track label is used at the beginning of track B of a volume. The label is written by the premark program. The label identifier, the label number and the volume identifier are filled with the data specified during the premarking, the other fields are filled with zeroes. The start of track label is read and checked by the system when track B of a volume is loaded.
The start of track label must be followed immediately by the file header label of the next file section, without a tape mark separating the labels.

File_header_label

The file header label is used at the beginning of a file or file section. The label identifier field and the label number field are handled by the system, the user has to supply only the file identifier. The file header label is managed by the system in the same way as for the compact labelling system.

End of track label

The end of track label indicates that the current file extends over the end of track A of a volume. The label is read and written in the same way as the end of track label for the compact labelling system.

End of file label

The end of file label indicates the end of data in a file. The end of file label is read and written in the same way as described in the section on the compact labelling system.

End of volume label

The end of volume label indicates that a file extends over the end of a volume. The label is handled in the same way as described in the section on the compact labelling system.

File structure

The file structure for the extended labelling system is identical to the file structure for the compact labelling system, except that track A of each volume starts with a volume header label, and track B with a start of track label.

DYNAMIC CATALOGUE

For the compact and the extended labelling system the Cassette File Management Package creates a dynamic catalogue, in which the names of the files recorded on a single volume are stored. The sequence of the file names in the dynamic catalogue corresponds to the sequence of the files on the volume for which the catalogue was created.
The file sequence stored in the dynamic catalogue enables the monitor to search forward or backward for a file header label, depending on the position of the tape at the read/write head, and the position of the file on the tape.

The dynamic catalogue is updated each time a search or write header request or command is issued. Only the first six characters of the file identifier are kept in the catalogue. The file identifiers for a specific volume are erased when the volume is reinitialized through a Write First Header command.

If the operator communication package has not been included in the monitor at system generation time, processing is carried out according to the instructions in the program.
If it has been included, the operator has additional control over program execution by means of a set of operator control commands which are described in this and the following chapter.

Control commands are entered via the operator's typewriter. The operator initiates communication with the system by pressing the INT button on the CPU control panel. This causes the monitor to type out M: to which the operator replies with a control command, followed by (CR) (LF).
Each command consists of two characters to identify it, followed by a space which is possibly followed by a number of parameters. These parameters may be separated by spaces or commas. Some parameters are optional.

Notes:

- The operator communication (OCOM) package is an optional module. If this module is not included in the monitor, the operator must take care not to press the INT button on the CPU control panel, because then then the system will issue a Halt instruction.
- Every numerical value in an operator control command is assumed to be hexadecimal, whether it is preceded by a slash (/) or not.
- If a message contains an error, the system types out the message ER. In that case the operator may press the INT button and try to enter the message again.

In the following paragraphs, the syntax and use of the available control commands are described, where Backus Normal Form is used as the notation for the syntax description:

| means: or

⌈ ⌉ means: optional component; any or all items within these brackets may be omitted, e.g. ⌈+| -⌉<integer> could be +<integer>, -<integer>, or <integer>

⌊ ⌋ means: alternative components; one of the items within these brackets must be selected, e.g. ⌊+| -⌋426 must be either +426 or -426

< > means: these brackets contain a syntactical item

␣ means: space.

The first five of the following control commands provide for management of the cassette tapes according to ECMA standards. They are part of the Operator Communication Package, but are directly related with Cassette File Management(CFM), and therefore selected at sysgen only when CFM is selcted as well. They are:

WF (Write First Header)
WH (Write Header)
RN (Run Program)
SH (Search Header)
CF (Clear File Code).

[WF] WRITE FIRST HEADER [WF]

Syntax: WF␣⌈</file code>,⌉⌈<file identifier>⌈,<option>⌉⌉

where:

</file code> consists of a slash (/) followed by 2 hexadecimal digits specifying the file code of the cassette magnetic tape onto which the first header must be written. Default value: /03 or the file code used in the previous Cassette Control Command, if any.

<file identifier> is a string of not more than 8 alphanumeric characters, identifying the file which will follow this header.

<option> is a string of not more than 19 alphanumeric characters containing information to be stored in the fields of the file header label (see Labelling System).

If neither <file identifier> nor <option> are specified in the command, the current cassette tape will become a working cassette tape (see chapter 10).

Note: Only the first 6 characters of the file identifier will be taken into account by the system.

Use: By means of this command, the user can reinitialize a cassette by writing a new file header label at the beginning of the cassette tape. The volume identifier is preserved.

The cassette tape must first have been premarked for Compact Labelling and is, after this command, recognized as such by the software system.

All the names in the dynamic catalogue corresponding to this cassette are erased.

Note: It is not possible to write a first header on the system cassette tape.

Error messages:

ER01: cassette tape not assigned as TL cassette
ER06: I/O error on tape
ER07: incompatible tape system or unloaded cassette on <f.c.>
ER09: wrong command syntax.

WH WRITE HEADER WH

Syntax: WH␣⌈</file code>,⌉<file identifier>⌈,<option>⌉

where:

<file code> consists of a slash (/) followed by 2 hexadecimal digits specifying the file code of the cassette magnetic tape onto which the file header must be written. Default value: the file code used in the previous Cassette Control Command, if any, or else /03.

<file identifier> is a string of alphanumeric characters identifying the file which will follow this header. In the compact system of labelling, this string may not be longer than 8 characters, in the extended system of labelling this string may not be longer than 17 characters.

<option> is a string of alphanumeric characters containing information to be stored in the fields of the file header label block (see Labelling System). In the compact system of labelling this string may not be longer than 19 characters, in the extended system of labelling this string may not be longer than 33 characters.

Default value: the fields of the file header label are filled with zeros, except for the label, volume and file identifier fields.

In the extended system of labelling <option> represents the fields of the file header label starting

from character position 22 up to and including character position 54 (see layout of file header label for extended system under Labelling System).
Note: Only the first 6 characters of the file identifier will be taken into account by the system.

Use: By means of this command the user can write a file header label onto the cassette tape identified by <file code>, to identify a file which will be written onto this cassette behind this file header label. The system will look for the last file on this cassette tape and write the file header label after its EOF. This command can be given only for cassettes which have been premarked for the compact or extended systems of labelling.
After the command WH the system is ready to accept the write orders for the file opened by the new header. All other orders compatible with the tape labelling system will be accepted, except another WH command, which will not be accepted until the file just opened is closed with an EOF.
Note: If an end-of-tape marker is encountered while the system is writing the file header label, the label will be written, but the system will close the file as an empty file and write an ETR (end-of-track) or EOV (end-of-volume) label. A message END is then output on the operator's typewriter and the user must change the track or volume. The system will then execute a new WH command and the correct status will be returned to the user program or control command.
When the header is written, the dynamic catalogue is updated with the new file identifier.

Error messages:

ER01: cassette not assigned as TL cassette
ER04: incorrect labelling
ER05: file already catalogued or previous file not yet closed with a write EOF
ER06: I/O error on tape

ER07: incompatible tape system
ER09: wrong command syntax
END : ETR or EOV encountered on the tape:
- if current track is side A: change to side B
-if current track is side B: load new cassette.

RN

RUN PROGRAM

RN

Syntax: RN␣ [</file code>][,<name>]

where:

<file code> consists of a slash(/) followed by 2 hexadecimal characters specifying the file code of the cassette tape containing the program which must be executed. Default value: the file code used in the previous Cassette Control Command, if any, or else /04.

<name> is a string of not more than 6 alphanumeric characters identifying the program which must be executed. Default value: the first file on the tape.

Use: By means of this command the operator may start the execution of the program specified in the command. The system will search the file header containing the program name specified, load the program and start it. If the file header containing the name is not found, an error message is sent.
The file code must have been assigned to a cassette tape.
This command performs the function 'inhibit access for labels'.

Error messages:

ER01: cassette not assigned as TL cassette
ER06: I/O error on tape
ER07: incompatible tape system or unloaded cassette on <f.c.>
ER08: unknown name
ER : loading impossible.
EC : erroneous cluster.

SH

SEARCH HEADER

SH

Syntax: SH␣[</file code>][,<file identifier>]

where:

<file code> consists of a slash (/) followed by 2 hexadecimal digits specifying the file code of the cassette tape on which the file header label must be searched. Default value: the file code used in the previous cassette control command, if any, or else /01.

<file identifier> is a string of not more than 6 alphanumeric characters matching the first 6 charac-thers of the file identifier in the file header label which must be found. Default value: first file on the tape.

Use: By means of this command the user can find a file catalogued on a cassette tape and preceded by the file header label specified in the command.

When the system has found the file header, via the dynamic catalogue, the tape is positioned after the tape mark following the file header label and the file will be accessible to the user.

After the SH command, the system is ready for a read operation on the file. Attempts to write on the file are rejected, unless this is the last file on which a write operation had not yet been performed after it had been opened by means of a WH command.

The SH command can be executed only for tapes labelled in compact or extended mode.

Note: This command performs the Inhibit Access for Label function (see I/O monitor request).

Error messages:

ER01: cassette not assigned as TL cassette

ER06: I/O error on tape

ER07: incompatible tape system or unloaded cassette on <f.c>

ER08: unknown name

ER09: wrong command syntax.

CF

CLEAR FILE

CF

Syntax: CF ␣[</file code>]

where:

<file code> consists of a slash (/) followed by 2 hexadecimal characters specifying the file code of the cassette on which this command must be executed.
Default value: the file code used in the previous Cassette Control Command, if any, or else /01.

Use:

When, during a search, an End-Of-Volume or End-Of-Track is encountered, this condition will be set in the status in ECB word 4 and the user will be requested to change track or volume. If this is not wanted, i.e. if the search is to be abandoned, this command can be used to erase this request and enable the user to load a new cassette.
The logical file code tables related to the cleared cassette will be erased and the status End-Of-File Set will be returned in ECB4.
The command is therefore comparable to a Release Device command.
A CF command implicitly performs an unlock function.

Error Messages:

ER01: cassette not assigned as TL cassette
ER07: incompatible tape system or unloaded cassette on f.f.
ER09: wrong command syntax.

ASSIGN FILE CODE

AS

Syntax: AS␣</file code>,<dev.name><dev,address>
where:
</file code> consists of a slash (/) followed by 2 hexadecimal digits specifying the file code assigned to the device specified by
<dev.name>< dev.address>.

Use: By means of this command a file code can be assigned to a device, or a previously assigned file code can be modified.

Note for cassettes:

During the COMGEN phase of system generation the user can specify, under the question CFM UNITS, the cassette units on which he wants to do I/O operations according to the ECMA standards, by specifying them with the device name TL. If he has not given a file code to these units under the question SPECIFY FILE CODES or wants to modify them, he can use this AS command.
If no CFM UNITS were specified, TK must be used as the device name in the AS command. It is not possible to give both device names (TL and TK) to the same unit.
If the user wants to change the access mode on a CFM unit from ECMA to standard I/O access or vice versa, he can do this by AS command also.
If an assignment is given as TL for a previous TK unit, or vice versa, an unlock function is performed.
The assign function must be performed before the cassette tape is loaded into the drive. If the assignment is done after the cassette has been loaded, the user must push the (black) load button on the drive, take out the cassette, push the load button again and put the cassette back in; the cassette will then be rewound and the new identification sequence will be started.

AB — ABORT A PROGRAM — AB

Syntax: AB

Use: This message can be given to terminate a program before its normal end.
All buffers are deallocated.

DM — DUMP MEMORY — DM

Syntax: DM␣<address1>␣<address2>

Use: This message produces a hexadecimal dump on the standard listing output device in full lines, so that <address1> and <address2> are included.
Each address consists of up to 4 hexadecimal characters.

Example: DM 0002 0004
causes a full line to be listed of the values contained at addresses 0000 to 000E inclusive.

HD — HALT DUMP — HD

Syntax: HD

Use: This message is used to terminate a memory dump before it reaches its specified end. This can be useful, for example, if the dump is being output on the typewriter, a slow device for large scale output, and continuation of the dump becomes unnecessary.

LD — LOAD A PROGRAM — LD

Syntax: LD␣[<value>]

Use: This message is used to load a program into memory. It is loaded from the standard object input device.
<value>: displacement value (in hexadecimal) relative to the beginning address of the user area. If specified, the program is loaded <value> characters after the beginning of the user area.
When a program is loaded, its identification, loading address and length are printed on the typewriter.

MC

MANUAL DEVICE CONTROL

MC

Syntax: MC␣<file code>␣<software order>[␣<repeat factor>]

Use: This message is given when the operator wishes to perform a manual operation on a magnetic tape device.

<file code> is the file code assigned to the device, consisting of 2 hexadecimal characters.

<software order> consists of 2 hexadecimal characters with the following significance:

16: skip forward to EOF mark
22: write EOF mark
24: write EOV mark
26: write EOS mark
31: rewind to load point
33: backspace one block
34: space one block forward (not allowed for cassette)
36: skip backwards to EOF
38: unlock

<repeat factor> allows to have the required function performed as many times as specified here, with only one MC message.

Example: MC 15 33

The device with file code 15 is to backspace one block.

PS

PAUSE

PS

Syntax: PS

Use: To temporarily stop the processing of the program. To restart it the operator must type RS.

RD

RELEASE DEVICE

RD

Syntax: RD␣<device address>

Use: Following an unsuccessful I/O operation, the monitor may type out a PU error message.
In that case the operator may give the RD command

if he wishes to release the operation from the device on which the error occurred.
<address> is the physical address of the device to be released, in 2 hexadecimal characters.
Control is returned to the user with the status in the ECB.

RS — RESTART A PROGRAM — RS

Syntax: RS␣<new A7>

Use: To restart a program stopped temporarily by a PS command or a Pause monitor request.
If the program has been stopped by a monitor request, it can be restarted with a new value in register A7.

Example: RS 81E
The program which was stopped by a Pause monitor request, is to be restarted with the value /081E in register A7.

RY — RETRY I/O OPERATION — RY

Syntax: RY␣<device address>

Use: Following an unsuccessful I/O operation, the monitor may type a PU error message.
The operator may then type the RY command to retry the operation after taking any necessary steps.
<address>: the physical device address (2 hexadecimal characters) where the operation has to be retried.
If the operation is now successful, control is returned to the user with the status in the ECB, otherwise the monitor will type out a new error message if another retry is possible.

Example: RY 02
Retry the last I/O operation on the device with physical address 02.

ST — START A PROGRAM — ST

Syntax: ST

Use: To activate the last loaded program.

WM — WRITE INTO MEMORY — WM

Syntax: WM␣ <address>␣<value1>[␣<value2> ....␣<valuen>]

Use: This message can be used to correct one or more memory locations.

<address> is the first address to be altered (up to 4 hexadecimal characters).

<value1> to <valuen> are values to be entered in the memory locations starting at <address>, i.e.

<value1> is placed in <address>

<value2> is placed in <address> plus 2.

Example: WM 4FE 44F 3FE4

The value /44F is placed in memory location /4FE

The value /3FE4 is placed in memory location /500.

The user program can request the monitor to execute certain functions. A request takes the form of an LKM (Link to Monitor) instruction followed by a DATA directive.
The directive has a number as operand which specifies the function to be executed. If this number is negative, the user is scheduling a label on completion of the request.
Preceding a request, certain parameters may need to be loaded into the A7 and A8 registers.
After the monitor has processed the request it loads a return code in the A7 register. If the requested module is not available, A7 is always -1.
The following requests are available:

DATA operand	Request	Page
1	Input/Output	1.54
2	Wait for an Event	1.60
3	Exit	1.61
4	Get Buffer	1.62
5	Release Buffer	1.64
6	Pause	1.65
7	Keep Control on Abort Condition	1.66

Note: It is possible to attach a scheduled label to a monitor request. For details, see chapter 4, Programming.

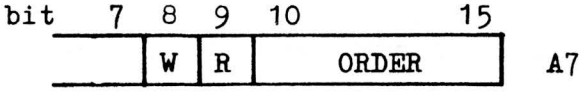
I/O REQUEST

Calling Sequence

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

Use

The user can ask the system to start a particular I/O operation on a peripheral device.
Register A7 is loaded with a CODE specifying the details of the I/O function, as follows:



W and R specify the mode of operation:

W = 1: the requesting program wants to wait for the completion of the requested I/O operation. Only after completion of the requested function, will the return to the calling program take place.

W = 0: a return to the calling program will be made as soon as the transfer has been initiated. The program will give a Wait request later on for synchronization.

R = 1: the program itself will process any abnormal condition concerning the requested operation (possible only with Basic Read/Write). The system will return the hardware status in ECB word 4. No retry is possible.

R = 0: any abnormal conditions will be processed by the system. The software status is returned in ECB word 4.

ORDER specifies which I/O function is required, by giving one of the following hexadecimal values:

There are a number of I/O orders. For devices other than cassette, they are handled by the Standard I/O Package. For cassettes, they are handled either by the Standard I/O Package or by the Cassette File Management Package, depending on whether the cassette drive used has been declared as CFM UNIT (TL) or not, i.e. TL: Cassette File Management Package
TK: Standard I/O Package.
(See also the AS control command.)

The following orders are handled by the Standard I/O Package:

-For cassettes and other devices:

/01: Basic Read

/05: Basic Write
For Basic I/O requests the system does not provide for character checking or data conversion, only for control command initialization and end of operation signals.

/02: Standard Read

/06: Standard Write
Standard (ASCII) I/O requests provide, by means of standard conversions, for special features such as error control characters, conversion from external code to internal ASCII and vice versa.

/08: Object Write 8+8
Object Write requests provide, by means of standard conversions, for special features such as error control characters, checksum and data conversion from 8+8 punched tape format to internal format.

/16: Skip forward up to tape mark

/22: Write tape mark / Write EOF

/24: Write EOV (End-Of-Volume) Label

/26: Write EOS block

/31: Rewind to load point

/33: Skip backward one block

/36: Search backward for tape mark

/38: Unlock

-For devices other than cassette:

/14: Read up to EOS

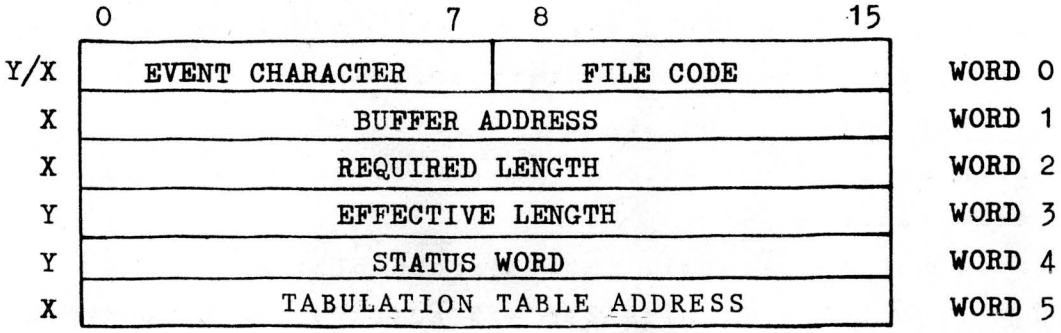
/34: Space one block forward

The following orders for cassette are accepted by the Cassette File Management Package, where a difference exists for the different types of ECMA standard labelling used:

BASIC LABELLING SYSTEM	COMPACT or EXTENDED LABELLING SYSTEM
/01: Basic Read	/01: Basic Read
/02: Standard ASCII Read	/02: Standard ASCII Read
/05: Basic Write	/05: Basic Write
/06: Standard ASCII Write	/06: Standard ASCII Write
/08: Object Write 8+8	/08: Object Write 8+8
/16: Skip forward to tape mark	
	/21: Get type of Labelling
/22: Write tape mark	/22: Write EOF label
	/23: Write File Header label
/26: Write EOS block	/26: Write EOS block
	/27: Search File Header label
	/29: Enable access for labels
	/2A: Inhibit access for labels
/31: Rewind to load point	/31: Rewind file
/33: Skip backward one block	/33: Skip backward one block
/36: Search backward for tape mark	/36: Search for first file
	/37: Search for next File Header label
/38: Unlock	/38: Unlock

Details on the functions of these I/O orders, relative to the various devices on which they can be used, are given in Appendix B.

The Event Control Block, of which the address must have been loaded into the A8 register, has the following format:



X: these words must be filled by the user

Y: these words are filled by the monitor

where:

WORD 0: event character:
bit 0 = 1: end of operation has occurred for the ECB.
The other bits remain unused.

WORD 1: address of the user buffer

WORD 2: requested length to be read or written, in words (basic read on card reader) or characters (other devices). The first character is always the character given by the buffer address. For standard write on typewriter or line printer, two characters must be added, at the beginning of the buffer.

Note: If magnetic tape or cassette tape handling orders are given in this monitor request (/16 and /31 to /38), it may be useful to make a separate ECB for these orders, as in these cases only ECB word 0 is important (file code). The contents of ECB1 and 2 are then irrelevant, *but must be unequal to zero.*

WORD 3: effective length which has been transmitted, in words (basic read on card reader) or characters (others). Stored here by the monitor upon completion of the I/O operation.

WORD 4: status word, stored here by the monitor upon completion of the requested I/O operation.
There are three types of Status word:
- hardware status (see Appendix E)
- software status for successfully completed I/O operations (see Software Status A below)
- software status for unsuccessful I/O operations due to an error in the calling sequence (see Software Status B below).

For Basic orders, the status is given as follows:

- operation successful: bit 0 = 0, the other bits contain the hardware status.
- operation not successful, due to error in calling sequence: bit 0 = 1, the other bits contain Software Status B.

For the other orders, the status is given as follows:

- operation successful: bit 0 = 0, the other bits contain Software Status A.
- operation not successful, retry impossible: bit 0 = 1, bit 1 = 0, bits 2 to 15 contain the hardware status.
- error detected in calling sequence: bit 0 = 1, bit 1 = 1, the other bits contain Software Status B.

SOFTWARE STATUS A (operation successful, bit 0 = 0)
bit 6 = 1: CFM: unknown file header label
bit 7 = 1: CFM: no data on cassette
bit 8 = 1: CFM: wrong labelling Standard I/O: End - Of-Volume
bit 9 = 1: CFM: end of file set Standard I/O: End-Of-Tape
bit 10= 1: Standard I/O: beginning of tape
bit 11= 1: Standard I/O: end of input medium
bit 12= 1: Standard I/O: incorrect length requested or checksum error
bit 13= 1: Standard I/O: illegal character code
bit 14= 1: an EOS mark has been read
bit 15= 1: an EOF mark has been read

SOTWARE STATUS B (error in calling sequence, bit 0=bit 1= 1)
bit 8 = 1: I/O request incompatible with file status (for a read request this means that the file has not been opened by a search header order, for a write request it means that the file has not been opened by a write header order).
bit 11 = 1: function is unknown or not compatible with the device
bit 12 = 1: illegal buffer size
bit 13 = 1: illegal buffer address
bit 14 = 1: device is attached to another program.
bit 15 = 1: an illegal file code has been used.

WORD 5: this word is used by the user to store the tabulation table address in the case of a standard read operation on ASR or punched tape equipment. This tabulation table has the following format.

Number of Tackets	First Tacket
Second Tacket	Third Tacket

The tackets indicate an absolute position in the print line. Characters up to the following tacket are filled with blanks.

Example:

3	10
20	30

Input line: LABEL \OPER \OPERAND \COMMENT

Line in buffer:

LABEL␣␣␣␣OPER␣␣␣␣␣␣OPERAND␣␣␣COMMENT

1 10 20 30

At completion of the input, the buffer is filled with spaces, but the returned length is the length effectively entered and stored, including the spaces replacing the tabulation codes (\).

Note: If word 5 is used for tabulation, the required length in word 2 must contain both the characters and the blanks in the tackets, i.e. for the above example word 2 must be filled with 36.

WAIT FOR AN EVENT

Calling Sequence

```
LDKL   A8,ECBADR
LKM
DATA   2
```

where:

ECBADR gives the address of the Event Control Block (see I/O requests). The first character of the ECB is the event character. If the first bit of this character is set to 1, the event has been completed.

Use

This request causes a program to stop and wait for the completion of an event which has to take place in another program (user or system). If the event has occurred, the dispatcher returns control to the requesting program. If the event has not occurred, the program is put in wait state, to be restarted when the event has occurred.

Note:

It is recommended not to use a Wait request inside a scheduled label routine, as this causes the whole program to be blocked temporarily.

EXIT

Calling Sequence

LKM
DATA 3

Use

This request is used to specify the end of a program. The program exit is effected after completion of all I/O operations and after all labels, if any, have been scheduled. A scheduled label exit passes control to the next scheduled label, if one is present, otherwise control passes to the main program.

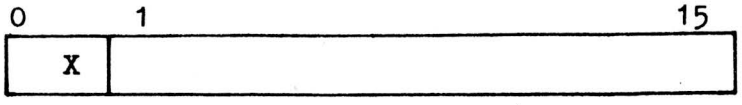
GET BUFFER REQUEST

Calling Sequence

```
LDK    A7, LENGTH
LKM
DATA 4
```

where:

LENGTH is the length, in characters, to be allocated to the buffer area (maximum 32k characters):



Bit 0 is not taken into account.

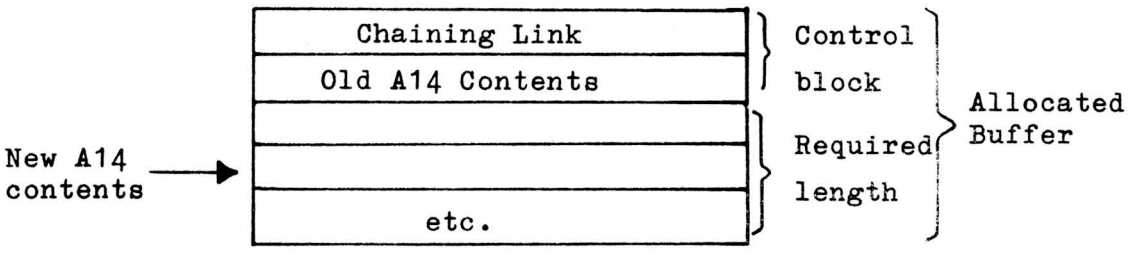
If 0 is loaded into A7, the monitor will return the highest memory address in A7.

If the memory size is 32k, 0 will be returned in A7.

Use

By means of this request, the user can allocate a memory area for temporary use, in the dynamic memory allocation area.

When the allocation is made, a control block is created by the system at the beginning of the allocated area. This block will contain a chaining link and the old contents of the A14 register:



The user must not destroy this control block

Upon completion of the request, the system responds as follows:

A7 = 0: the buffer is allocated

= 1: there is no memory space available (bit 0 in LENGTH = 0)

A14: contains the address of the fourth word of the allocated buffer, so that, as soon as the buffer is allocated, the user may give a Call Function instruction with the A14 register without having to update the A14 register first.

However, the user must then provide for stack handling.

RELEASE BUFFER REQUEST

Calling Sequence

```
LDKL   A14, BUFADR
LKM
DATA 5
```

where:

BUFADR points to the second word in the buffer as given in register A14 after the Get Buffer request.

Use

To release the memory space previously reserved by a Get Buffer request. The A14 register is reloaded with the value it contained before the Get Buffer request was made.
The system responds as follows:

A7 = 0: the memory space is released.

If the A14 pointer is incorrect, or if the buffer area has been destroyed, the system issues a Halt.

PAUSE

Calling Sequence

```
LDK    A7,MESLGT
LDKL   A8,MESBLK
LKM
DATA 6
```

where:

MESLGT is a constant, specifying the length of a message which the program may output on being put in Pause state (in characters).
MESBLK is the address of a message block containing the message to be output.

Use

This request causes a temporary halt of the running program. It is put in wait state and, if specified, a message is printed out on the operator's typewriter. The program can be restarted only by an operator control message RS. When the program is restarted, it may be given an additional parameter in register A7 (see RS operator control message).

Notes:

- It is very useful if the user mentions, in the output message, that the program now is in pause state.
- The message must start with a control character.

KEEP CONTROL ON ABORT CONDITION

Calling Sequence:

```
LDKL A7,PARAM
LDKL A8,LABADR
LKM
DATA 7
```

where:

LABADR is the address of a user label to be scheduled on abort. PARAM is the address of a 3-word block which will receive the parameters of the abort condition. It has the following format:

	Abort Code	0
Aborted PSW		1
Aborted Address (A0)		2

where:

- the abort code has the same meaning as in the regular AB abort message.
- word 1 gives the PSW of the aborted program.
- word 2 gives the address where the abort took place (register A0 contents).

Use

By means of this request the user can himself handle abort conditions, i.e. his own routine replaces the abort handling of the monitor. Thus, the program will not be declared aborted by the monitor, but control will be transferred normally to the user routine attached to this request (register A8). The parameter of the abort condition will be placed in a 3-word block and, according to the returned parameters, the user routine can take action. This request is to be given once in a program, at the point from where the user wants control over any abort conditions, so mostly at the beginning of the user program. It is usable only once, so

if an abort takes place and the program is restarted, the request has to be given again. When a user decides that, at a certain point in his program, he wants to return to regular abort handling by the monitor, he may do so by provoking an 'artificial' abort.

The abort code returned in the PARAM block may be one of the following:

01: simulated routine save area overflow

02: non-available instruction used

04: buffer area destroyed or block was bigger than 16k words

05: label could not be scheduled

Note:

An operator abort (AB) can never be blocked.

System messages are printed by the monitor on the operator's typewriter. For monitors in which the operator communication package has not been included, no messages are printed, but when abort and exit conditions arise, the P-register halts at a specific address.

Error in Operator Command

format: ER

meaning: the operator command contains an error. The operator must push the INT button and retype the message.

Errors in Cassette Control Commands

ER: loading impossible (RN command)

ER01: cassette tape not assigned (WF, WH, RN, SH and CF commands)

ER04: incorrect labelling (WH command)

ER05: file already catalogued / previous file has not yet been closed with an EOF (WH command)

ER06: I/O error on tape (WF, WH, RN and SH commands)

ER07: incompatible tape system (WF, RN, SH and CF commands)

ER08: unknown name (RN, SH commands)

ER09: wrong command syntax (WF, SH and CF commands)

XX END: END, ETR or EOV detected on tape XX (WH command)

Abort

format: ABORT␣<code>␣<address>

meaning: an error has been encountered and the program has been aborted.

<code> may be : 01: simulation routine save area overflow
02: illegal instruction
04: buffer area destroyed or block was bigger than 32k
05: label could not be scheduled
06: operator abort

<address> is the address at which the abort occurred.

Erroneous Cluster
format: EC
meaning: an erroneous cluster has been encountered on punched tape. Program loading stops.

End of File
format: :EOF
meaning: the loader has read the en-of-file record and stops loading.

End of Segment
format: :EOS␣<address>
meaning: the loader has read the end-of-segment record.
<address> is the address of the first free location after the loaded program.

Program Termination
format: EXIT
meaning: This message is printed when execution of a program is completed.

Program Identification
format: IDENT␣<program name>␣<address>
meaning: the loader has read the program identification
<program name> is the name given in the IDENT instruction.
<address> is the start address of the loaded program.

No Start Address
format: NS
meaning: no start address was specified in the END/START cluster (record type 7. See Appendix). The program can not be executed.

Overflow
format: OV
meaning: Insufficient memory available. Program loading stops.

Peripheral Device Error

format: PU␣<Device name and address>␣<hardware status>[␣RY]

meaning: a failure has been detected on a peripheral device. RY is printed if a retry is possible. (See operator commands).
If the operator releases the operation on the device (RD operator command), the system will consider the I/O operation completed and control will be returned to the user.
The <hardware status> can be found in Appendix E.

Input/Output Error

format: I/O ERROR␣<device name and address>␣<hardware status>

meaning: an error has occurred during an I/O operation on the device specified (e.g. throughput error, data fault), in connection with an MC operator command.
The hardware status can be found in the Appendix.

Errors during Cassette Loading

XX E 0A: unknown type of labelling

XX E 06: I/O error on tape

XX E 03: bad volume or track loaded

XX E 0B: impossible to load cassette because a Retry command has been given for it. The user must:
- release the device: M: RD TKXX
- and reload the cassette.

XX is the device address of the drive containing the erroneous cassette

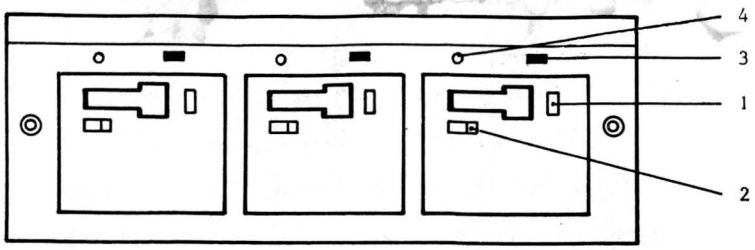
Dynamic Catalogue Overflow

E 02

THE CASSETTE TAPE DRIVE UNIT

The cassette tape drive units are mounted in an equipment shelf together with a control unit. The shelf may contain up to three drives, all of which are controlled by the same control unit which may be connected to the Programmed Channel or to the I/O Processor.

Controls and Indicators



For each cassette drive the following controls and indicators exist on the front panel of the cassette drive and on the panel of the equipment shelf:

- 'lock' lamp (1) : a white light which goes on when a cassette is inserted and locked into position inside the unit. This light remains on as long as the drive is locked.
- retrieval knob (2): this is a cassette release knob which must be pressed to retrieve a cassette from the drive unit. The retrieval knob is locked during the time a lock signal is fed to the unit from the external equipment and also during a rewind operation to prevent the cassette from being removed.
- load button (3) : this is a black button located above the right hand corner of the drive unit. It must be pushed before a cassette is inserted. When the cassette is locked into position, the lock lamp goes on and the retrieval knob is locked.
  If a cassette must be removed, the load button must first be pushed, extinguishing the lock lamp and releasing the retrieval knob.

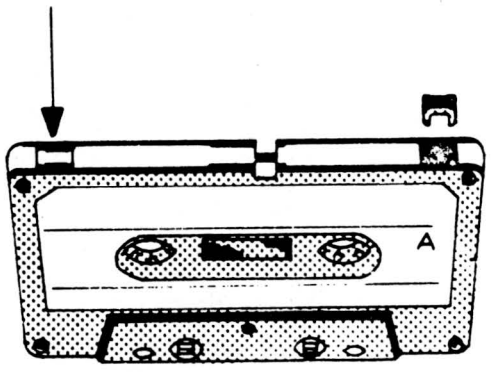
- orange lamp (4) : this light is always on when a cassette drive is empty. When a cassette has been inserted into a drive unit, it goes on during rewind, read and write operations.

The Cassette

The cassette has an A-side and a B-side corresponding to the tape A-track or B-track. The side inserted uppermost in the drive unit is the side in use. It is usual to start with the A-side first. In either case however, the unit detects which side of the cassette is uppermost by means of an asymmetrically positioned cut-out in the back of the cassette.

In the back edge of the cassette are two symmetrical cut-outs into which (red) write-enable plugs are normally fitted. The cut-out which is furthest away from the 'A' or 'B' marked on the cassette corresponds to the relevant track or side (see figure below). If a write-enable plug is not fitted to a cut-out, then the information contained on the corresponding track of the tape can not be accidentally erased. The plugs have no influence on the read process.

WRITE ENABLE PLUG A-TRACK



Loading a Cassette

With the unit switched on, load a cassette as follows:
- if required, fit a write-enable plug in the cut-out on the cassette, corresponding to the side to be used.
- push the black load button; the orange light now goes out.
- with the side to be used uppermost, load the cassette into the cassette slot with the thickest part of the cassette corresponding to the widest part of the slot.

- when the cassette has been inserted sufficiently far, the cassette holder will automatically bring the cassette into the correct position. The white lock lamp now goes on and the retrieval knob is locked.
- the unit will now operate automatically under the control of the computer. When it is in operation, the orange lamp will be on.

Removing a Cassette

A cassette can be removed only when the lock lamp is extinguished and the retrieval button unlocked. This can be done either by pushing the black load button or by operator command via the typewriter (MC command with order 38). After this, the retrieval knob must be pressed, which will cause the cassette to be ejected from the cassette slot under spring action sufficiently far for the final removal to be accomplished by hand. When the cassette has been removed, the orange light will go on again.

Types of cassette

System Cassette

The cassette tape which contains, as its first file, the IPL + Monitor is called the system cassette.
The data on the system cassette are always recorded with the Compact system of labelling (see chapter 6).
The system cassette is the only cassette handled by the Cassette File Management Package which does not have a File Header Label (HDR) or Volume Header Label (VOL) at its beginning.
The system cassette is created during system generation, when the IPL and monitor are recorded on it. (It is also possible to premark a cassette as system cassette and copy a monitor on it by means of the Update Package.)
When the monitor has been loaded from the tape into memory, the cassette can be removed and another one put in its place, unless the system tape also contains the system processors and these are going to be used, in which case the cassette must be taken out after monitor loading and put back into the drive, after which the processors can be searched.

User Tape

All cassette tapes onto which data are recorded according to ECMA standards are called user tapes.

When a cassette is loaded, the system immediately checks it to see according to which ECMA standard the tape must be used. This information has been put on the tape by the Premark program. All I/O operations for that tape will then be done according to the standard specified for it.

Working Cassette

The working cassette is a cassette using the compact system of labelling and capable of storing only one single file: a working file.

A working cassette is declared as such when the cassette is premarked or by means of the cassette control command Write First Header (WF).

For Read operations it is not necessary to give a Search Header command before starting the read, because the system automatically rewinds the cassette tape when it is loaded or when an EOF label is read.

For Write operations it is not necessary to give a Write Header command before starting to write, because the write operation is automatically started at the beginning of the file. When the cassette is loaded, the tape is implicitly rewound; this is also done when an EOF is written.

LOADING PROCEDURES

Three types of cassette can be loaded:

- system cassette
- user cassettes
- working cassettes

When a cassette is loaded, a ready interrupt starts the identification routine which reads the first block recorded on the tape. This block contains information about the type of labelling used for the cassette, which is printed out on the typewriter, along with the device address of the drive into which the cassette has been loaded. For a cassette on which the compact labelling

system is used, for example, it may print: 15 C::VOL9, indicating that a cassette with compact labelling has been loaded into drive 15 and that its volume name, given during Premark, is VOL9.
For Basic Labelling a B is printed and for Extended Labelling an E is printed, e.g. 15 E::CAND, where CAND is the volume identifier.

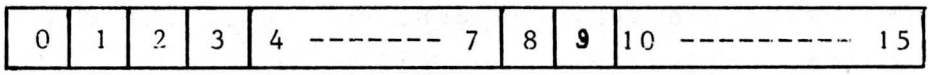
Note: When the system cassette is loaded, C::SYST is printed out on the typewriter.

Loading Bootstrap, IPL and Monitor

The bootstrap can be loaded in one of two ways, depending on whether the optional ROM bootstrap is included in the system or not.
If the ROM bootstrap is included, which is highly recommended, the procedure is as follows:

- switch on the CPU
- push the system cassette into the drive with file code 01 with side A up, after having pushed the black load button
- wait until the cassette has been rewound

- set up the device parameters on the toggle switches on the CPU control panel, as follows:



where:
bit 0 =0: no punched tape used or, if used format is 8+8
=1: punched tape format is 4 x 4
bit 1 =0: the device used is not a disc
=1: the device used is a disc
bit 2 : used only if bit 1 = 1
=0: the disc used is a fixed-head disc
=1: the disc used is a moving-head disc
bit 3 =0: the device is connected to the I/O processor
=1: the device is connected to the programmed channel
bits 4 to 7 are used to qualify the CIO Start command sent by the bootstrap and are transferred to the addressed control unit on lines BIO 12 to 15 when required

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

=1: the control unit involved is a multiple device control unit

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

bits 10 to 15 contain the device address

Example:

0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1

(hexadecimal 0785)

This indicates that the IPL must be loaded from the device with address 05: a cassette tape drive (on multiple device control unit) connected to the I/O processor.

- then push the MC button on the CPU control panel
- push the IPL button
- the bootstrap is now loaded into memory which immediately starts reading and loading the IPL from the device specified on the toggle switches, i.e. a cassette drive
- when the IPL has been loaded into memory, it outputs the message:
  OBJECT TAPE ON READER. THINK OF BASE!
  if the IPL contains a Halt.
  If it does not contain a Halt, it continues automatically and loads the monitor. If it has stopped after output of the message, and the monitor is to be loaded, which is on the cassette immediately following the IPL, the user must push the RUN button to have the monitor loaded into memory:
- the monitor name and start address are output as follows:
  IDENT XXXX
  When loading is terminated,
  :EOS XXXX
  :EOF
  is output, XXXX being the first free address after the monitor in memory.

- the monitor is now ready for use and when the INT button on the control panel is pushed, it types out
  M:
  and waits for the user to type in a command.

When the ROM bootstrap is not included in the system, the procedure is as follows:

- switch on the CPU
- load the bootstrap into the first 64 memory locations manually, by means of the toggle switches and the Load Memory button on the control panel. The 64 bootstrap values can be found in Appendix F.
- check by reading these locations out
- put the system tape cassette in the drive with file code 01 and wait until it has been rewound
- push the MC button
- push the RUN button
- now the IPL is loaded, which will then load the monitor.

Loading and Running Programs

When the monitor has been loaded, including the operator communication package, the user may type in commands after he has pushed the INT button on the CPU panel. When this is done the system types out

M:

and waits for the user to type in a command.

To load and start an object program, for example a processor, the user can type in the command RN. If the program is on the device assigned to file code 04 (standard load file code) it suffices to type RN <program name> , unless an other file code was used in a previous cassette control command, in which case that file code becomes the default value. In any case, it is possible to load the program from any cassette unit, as long as the file code is specified. The RN command performs a search header, load and start function. This implies that the user may also give an SH, LD and ST command successively, for programs in load module format on cassette tape.

For programs on other devices, the user must position the program to be loaded correctly on the reading device, e.g. tape reader, magnetic tape, and then give LD and ST commands successively to load and start the program.

PART 2 ASSEMBLY LANGUAGE

This part contains a description of the Assembly Language. In this description it is made clear how the programmer can write his programs using the instructions of the P852M Instruction Set as well as the directives which guide the assembly process when the program is input to the Assembler.

Programs for the P852M computers are written in a symbolic language closely related to the machine code. Each statement (or line) of the program relates to a single machine instruction or to a data item to be taken into account by an instruction.

To write programs in the Assembly Language, the user should be familiar with the syntax of the instructions, which are devided in the following main groups:

- Load and Store instructions
- Arithmetic instructions
- Logical instructions
- Character handling instructions
- Branch instructions
- Shift instructions
- Control instructions
- Input/Output instructions

Programming in Assembly Language requires certain rules to be acceptable to the Assembler.
A source program may consist of one or more modules each of which starts with an identification IDENT and terminates with an END (see directives). The whole source program must be terminated by an "End Of File" mark (:EOF).

NOTE: If a source program consists of several modules the modules need not be separated by :EOF marks but by :EOS marks (End Of Segment).

Each module of a program consists of a number of characters grouped into lines and each statement in a module is made up of the following characters:

Letters: A to Z inclusive

Digits: 0 to 9 inclusive

Delimiters:

+	plus
–	minus
*	asterisk
=	equal
'	apostrophe
,	comma
	blank
/	slash
(	left parenthesis
)	right parenthesis
.	period
:	colon

Location counter

The Assembler maintains a location counter which is a software counter used to assign a relative or absolute memory address to program elements. The location counter starts with a relative value equal to zero, or it starts at an absolute address defined by the AORG directive, at the beginning of an assembly. The value of the counter is incremented by 2 or a multiple of 2 depending on the kind of instruction given.

The current value of the location counter is referred to by an * in the operand field (see below). In absolute program sections * has an absolute value. In that case the value is incremented in the normal way and the value may be changed by a RES or RORG directive.

The location counter may take neither a negative relative value nor an odd value.

Symbols

A symbol is a character or a string of characters used to represent addresses or values. Symbols may appear in the label field as well as in the operand field of a statement.

Their syntax is the same as for the label (see under label field). Some symbols are predefined and have a special meaning for the Assembler e.g. * indicates the current value of the location counter, P is the instruction counter etc.

The following symbols are used to define the syntax of the P852M Assembly Language.

< > to enclose syntactic items
| the vertical stroke has the meaning of or
::= is composed of
[ ] the syntactic items between these brackets may be omitted
[ ] select one of the items between these brackets
␣ space

The following list contains the definition of all items used.

<statement> ::= [<label>]␣<operation code>[<operand>][<comments>][*<comments>]
<label> ::= <identifier>
<operation code> ::= <mnemonic>[S|(<cnd>)|L][*]<directive>
<operand> ::= [+|−]<term>[+|−]<-term>[+|−]<term>
<comments> ::= <characters>|*<characters>
<identifier> ::= <letter>|<identifier><letter>|<identifier><digit>|<identifier>
<mnemonic> ::= <letters representing operation code>
<S> ::= <store indicator>
(<cnd>) ::= <numerical condition value>|<condition mnemonic>
<numerical condition value> ::= 0|1|2|3|4|5|6|7
<condition mnemonic> ::= Z|P|N|O|E|G|L|A|U|NA|NR|NZ|NP|NE|NG|NL|NN
<L> ::= < load indicator>
* ::= indirection
<directive> ::= <IDENT, END etc.> see chapter on directives
<DATA defined hexa constant> ::= <see DATA directive>
<module name> ::= <symbol>
<symbol> ::= <characters representing address or value>
<predefined expression> ::= <max. of two defined symbols>
<entry point name> ::= <identifier within reference module>
<external> ::= <identifier defined in other module>
<common-field definition list> ::= <commen field definition>,<common field definition> .....
<common field definition> ::= <common field name>[<common field length>]
<common field name> ::= <identifier>
<common field length> ::= predefined (absolute) expression
<internal symbol list> ::= <internal symbol>,<internal symbol> .....
<internal symbol> ::= <identifier>
<field definition> ::= <field length definition>|[=|:]<field value definition>

```
<field length definition>      ::= <number of bits>
< = field value definition>    ::= <value to be placed in field>
<:field value definition>      ::= <address of word>
<field number>                 ::= <decimal integer>
<term>                         ::= <constant>|<symbol>
<constant>                     ::= <decimal constant>|<hexadecimal
                                   constant>|<character constant>
<decimal constant>             ::= <digit>|<integer>
<hexadecimal constant>         ::= <hexadecimal integer>
<character constant>           ::= <letter>|<digit>|<delimeter>
<letter>                       ::= A|B|C|D|E|F|G|H|I|J|K|L|M|-
                                   N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<digit>                        ::= 0|1|2|3|4|5|6|7|8|9
<delimiter>                    ::= +|−|*|=|'|,|␣|/|(|)|.|:
<integer>                      ::= <number>
```

1 Format of source statements

A source module consists of a sequence of statements. The Assembler interprets each line as it is presented.
Statements can be divided in the following fields:

- label field
- operation field
- operand field
- comments field

< statement > :: = [< label >]␣ < operation code > ␣[< operand >]␣
[< comments >]
*[< comments >]

Each field has to be separated from the other by one (or more) blank character(s). Blanks may not appear in the fields themselves except when specified in a character constant or in a comments field. Instead of blanks a backslash may be used for separation (see page 2-12). One or more blanks at the beginning of a statement indicate that there is no label field.

If there are more than ten blanks after the operation field all following characters are considered to be belonging to the comments field.
An * (asterisk) at the beginning of a statement identifies that line as a comments line.

Statements punched on tape which are to be read by the ASR punched tape reader have to be terminated by LF XOFF CR, which switches the reader off, followed by a Null character, e.g. Rub-out, to allow for a proper reading and processing of the next usable character.

PHILIPS

PHILIPS data systems

ROBLEM Example of programming paper DATA PAGE OF PROGRAMMER

label 1 6	operation 10 18	operand 19 20 30 40 50 60 70 72	identification 73 80
1 0 D 1 F 1	E Q U	*	
	L D K L	A 1 0 3 B U F R 8 0	W O R K A D R E S S
	L D K L	A 1 2 3 0	
1 0 D 1 F 1	L D K L	A 1 2 3 B U F 0 U 1	
	A D R	A 1 1 3 A 1 2	
	L D R	A 9 3 A 1 1	
	L D K L	A 1 3 3 M O V E	

LABEL FIELD

< label > :: = < identifier >
< identifier > :: =
< letter > | < identifier > < letter > | < identifier > < digit > | < identifier >

Labels (or identifiers) in a module are used for reference purpose to other statements in a module.
The Assembler assigns, in most cases, to each label a word address value which is the numerical equivalent (absolute or relocatable) of the label.

The maximum number of characters in a label recognised by the Assembler is six. The first of those must always be a letter. A label, however, may contain more than six characters but the additional characters will not be taken into account. If the label has already been allocated to another statement an error message is output.
Period signs in a label are not significant, e.g.

L.A.B.E.L. has the same meaning as LABEL

The value of a label is normally regarded as relocatable, except when:

- an absolute address is equated by an EQU directive
- the label appears in an absolute program section (defined by the AORG directive and which is not equated by an EQU directive to a label previously defined as relocatable).

OPERATION FIELD

< operation code > :: = < mnemonic > [S|(< cnd >)|L] [*] < assembly directive >

where:

< **mnemonic** >

The operation field normally contains the mnemonic of a standard instruction. It is possible, however, to generate one's own instruction mnemonic by means of the FORM, XFORM and GEN directives (only with the Extended Assembler).

S
Allowed after the mnemonic of certain register to register and memory reference instructions. It indicates that the result of the operation must be stored in a memory word (bit 15 of the instruction is set to 1). In fact, S had to be considered as a part of the instruction mnemonic.
e.g. C1R and C1RS instructions are to be considered as two different instructions.

NOTE: It is allowed to have the S preceded by a period sign though the Assembler does not take this sign into account.
e.g. AD.S␣ = ADS␣

(<**cnd**>):: = <numerical condition value>|<condition mnemonic>
<numerical condition value>:: = 0/1/ . . . . . /7
<condition
mnemonic>:: = Z|P|N|O|E|G|L|A|R|U|NA|NR|NZ|NP|NE|NG|NL|NN

This indicator specifies the condition under which a conditional branch instruction is to be performed. The table below shows how in the Extended Assembler the conditional mnemonics and numerical condition values may be used. On the Basic versions only numeric notation can be used.

contents of CR / Instruction type	CR = 0		CR = 1	
	Condition mnemonic	Numerical condition value	Condition mnemonic	Numerical condition value
arithmetic compare I/O	Z (zero) E (equal to) A (accepted)	0	P (positive) G (greater than) R (refused)	1
	CR ≠ 0		CR ≠ 1	
Compare I/O	NZ (not zero) NE (not equal) NA (not accepted)	4	NP (not positive) NG (not greater) NR (not refused)	5

	CR = 2		CR = 3	
	Condition mnemonic	Numerical condition value	Condition mnemonic	Numerical condition value
	N (negative) L (less than) —	2	O (overflow) — ␣ (address unknown)	3
	CR ≠ 2		CR ≠ 3	
	NN (not negative) NL (not less)	6	— —	7

.L
Allowed after the instruction mnemonic of a constant instruction. It specifies that the operand is contained in 16 bits i.e. that the instruction must be assembled as a "long" instruction.

*
Indicates the indirect addressing mode in a register to register or a memory reference instruction.

OPERAND FIELD

The operand field may contain an address expression, a register expression or constants associated with the current machine instruction or assembly directive or a combination of those.
The structure and meaning of the operand depends on the type of instruction and directive and is explained below.
All operand expressions must be separated by a comma.

Expression

<expression> :: = [+ | −]<term>[[+ | −]<term>[[+ | −]<term>]]
<term> :: = <constant> | <symbol>

NOTE: * is considered to be a symbol.

An expression may not refer to more than 2 symbols and may not refer to more than one register name. In the latter case it may not contain any other term.

Address expression

The address specified in a memory reference instruction can be either absolute or relocatable.
An *absolute address* is the actual address in memory where the information the user needs can be found.
A *relocatable address* is relative to the beginning of the program in which it appears.

The address expression may contain any of the following terms or a combination of them:

*	asterisk, which is a predefined expression representing the current value of the location counter. This counter is incremented by two or a multiple of two depending on the length of the instruction.
symbol	used to refer to an instruction or data word with the same identifier in its label field. The Assembler will convert the symbol to a relative address.
displacement value	which can be attached to * or <symbol> to indicate a word not labeled by an identifier.

Predefined expression
A predefined expression is an expression consisting of not more than two symbols, each of which is defined i.e. has been assigned a value. Some symbols are implicitely predefined in the Assembler (see page 2.32).

An expression may contain only one external reference. The remainder, if any, of such an expression must have a predefined absolute value. The combination of an external reference and a predefined absolute value may only be used for specifying the value of a 16-bit field. The table below shows the result of a combination of positive and negative absolute or relocatable values:

1st term / 2nd term	+ R	− R	+ A	− A
+ R	E	A	R	R
− R	A	E	E	E
+ A	R	E	A	A
− A	R	E	A	A

where:

R = relocatable
A = absolute
E = erroneous

Register expression
Register expressions are regarded as predefined expressions and consist of one or two characters. The register expressions recognised by the Assembler are:

P	P-register or instruction counter
A1 . . . A14	Registers 1 to 14 (general purpose registers)
A15	Register 15 (stackpointer)

Constants
A variety of constant types may be specified in the operand of an instruction or directive.

<constant> :: = <decimal constant> | <hexadecimal constant> | <character constant>

Decimal constants
<decimal constant> :: = <digit> | <integer>

The decimal constant is a digit or integer contained in an 8-bit character or 16-bit word whose value may range from 0 to 32767.

Hexadecimal constants
<hexadecimal constant> :: = /<hexa integer> | X' <hexa integer>

The hexadecimal constant is considered to be hexadecimal value or bit string in the range from 0 to /FFFF.

Character constants
<character constant> :: = '<character>[<character>]'

A character constant is composed of a character string enclosed in single quotation marks. The string is composed of the characters described in the character set on page 2.3.

A character constant can be used with a machine instruction only if the constant consists of either one character (short constant) or two characters (long constant). Longer strings can be specified in a DATA directive. A single quote mark (') used as a character is specified by two consecutive single quote marks.

COMMENT FIELD

Comments are only for the programmer's benefit. They are included in the assembly listing but not in the generated object program.
A line is considered to be a comment line when the first 10 characters of that line are blanks or when the line starts with an asterisk.

INPUT OF SOURCE STATEMENTS AND CORRECTIONS

The user may type in the statements and corrections from the operator's typewriter. He may do so by counting the number of characters to obtain a neat output on the listing device.

Example:

1st col		10th col		19th col		40th col
label	␣	opcode	␣	operand	␣	comments

may be typed as follows:

label\opcode\operand\comments

without having to count for the first column of each field.

Example:

```
DATAF\LDK\A4,4
\ABL(7)\HALT
DEVUN\LDK\A4,5
\ABL(7)\HALT
ADDIT\LDK\A1,0\SET INDEX REGISTER FOR BUFFER.
\LDK\A3,00FF\LOGICAL CONSTANT INTO A3
```

ADDRESSING MODES

In the Instr. Set manual we see how addressing takes place from a hardware point of view. The condition an instruction must fullfill to meet the requirements of the Assembler is explained on the preceding pages. Specific examples, with source statements and explanation concerning the arithmetic instructions AD and ADR are given to show the operation within the CPU.
See for the hardware operation of those instructions that manual. The order in which the examples are given is in accordance with the description on those pages.

Direct addressing

AD A1,LABEL	The contents of the memory location with symbolic address LABEL are added to the contents of register A1. The result is placed in A1.
ADS A1,LABEL	The contents of the memory location with address LABEL are added to the contents of register A1. The result is stored in LABEL.

Indexed addressing

AD A2,LABEL,A10	The contents of register A10 are added to the address LABEL. The result gives an address whose contents are added to the contents of A2. The result of the latter operation is placed in A2.
ADS A2,LABEL,A10	The contents of register A10 are added to the address LABEL. The result gives an address whose contents are added to the contents of A2. The result of the latter operation is stored in the address: LABEL + contents of A10.

Indirect addressing

AD* A2,LABEL	The contents of LABEL point to an address whose contents are added to the contents of register A2. The result is placed in A2.
ADS* A2,LABEL	The contents of LABEL point to an address whose contents are added to the contents of register A2. The result is placed in the contents of LABEL.

Indexed Indirect addressing

AD* A2,LABEL,A10	LABEL is added to the contents of register A10. The result points to an address whose contents are added to the contents of register A2. The result hereof is placed in register A2.
ADS* A2,LABEL,A10	LABEL is added to the contents of register A10. The result points to an address whose contents are added to the contents of register A2. The result hereof is placed in the address obtained of A10.

Register to Register operation

ADR A1,A2	The contents of A2 are added to the contents of A1. The result is placed in A1.

Register addressing

ADR* A1,A2	The contents of the address pointed to by A2 are added to the contents of register A1. The result is placed in A1.
ADRS A1,A2	The contents of the address pointed to by A2 are added to the contents of A1. The result is stored in the address pointed to by A2.

LOAD AND STORE INSTRUCTIONS

Load Instructions
Before the programmer can perform an operation on the contents of a memory location or a register its contents must be placed in one of the registers A1 thru A15.
Two load instructions are provided, allowing to load a 16-bit word from anywhere in memory or from any register into a specified register where the operation will take place, and one instruction to load a constant into a register.

Store instructions
Companion to the load instruction is the store instruction which may store the contents of a register, containing the result of an operation, into a memory location or a register.

ARITHMETIC INSTRUCTIONS

Arithmetic instructions perform the normal arithmetic functions such as add, subtract. The instruction operand operates upon the contents of the specified register.

LOGICAL INSTRUCTIONS

Instructions described under this heading are called logical instructions because they operate on binary information according to the rules of logic. The first operand which may be a memory location, a register (R1 or R3) or a constant is compared with the second operand, register R2. The result is placed in a register or possibly in memory. In the instruction set each logical instruction is given a description in which way the contents of a memory location is ANDed or ORed.

CHARACTER HANDLING INSTRUCTIONS

Character handling instructions operate on a character level. Characters may be exchanged, compared or 8 bits of a constant may be placed in 8 bits of a register.

BRANCH INSTRUCTIONS

These instructions cause a branch to an address in memory either when a certain condition is fulfilled or unconditionally.

In branch instructions on condition the instruction mnemonic is followed by a number ranging from 1 thru 6, enclosed in brackets. When the number is (7) or omitted, the branch is unconditionally.
These numbers are compared with the contents of the condition register set by the previous instruction.
The condition number has the following meanings:

(0) branch if CR	= 0	(4) branch if CR	≠ 0
(1)	= 1	(5)	≠ 1
(2)	= 2	(6)	≠ 2
(3)	= 3	(7) unconditional branch	

Example:

```
            —
            —
            LDK     A2,4
LABEL       SUK     A2,1
            RB(4)   LABEL
            —
            —
```

The Assembler allows to use, instead of a number, a condition mnemonic e.g. Z, E, A (see page 2.9).
Unconditional branches are made by the following instructions:
- absolute branch instruction or relative branch instruction without a condition indicator or when (7) is specified.
- CF, RTN, EX instructions.

Long format absolute branch instructions permit to branch, forward as well as backwards, to any address in the program. Short format absolute branch instructions may only branch to locations 0000 to 00FE. Relative forward and relative backward instructions may not skip backwards more than 127 locations and 128 locations forward.
The Assembler gives an error indication if the permissible branch range is exceeded.
The address to which control is to pass may be indicated in various ways:

1. By means of a symbolic address expression:
   ABL(3) LABEL

2. By an absolute address held in a register:
   ABR (7) A5

3. By using a constant to indicate an absolute memory address (short constant):
   AB /84

4. By means of a displacement value added to or subtracted from the instruction counter value (RB and RF instructions only). This displacement value is computed by the Assembler from an address expression used in the operand and may not exceed more than 128 words forward or 127 backwards:
   RB(0) ZERO

Another group of branch instructions are the Call Function and Return from Function instructions. The Call Function instruction provides a link to a subroutine by branching to the first instruction of the sub-routine. To be able to resume the execution of the main program after the subroutine has been executed the contents of the P-register and the Program Status Word are stored in the stack. When the last instruction of the subroutine (RTN) is executed the contents of P and PSW are restored.
A special group within the branch instructions is formed by the instructions EX, EXK and EXR.
These instructions allow to address a memory location of which the contents is the binary representation of another instruction. The latter instruction is executed before the program continues with the next instruction in sequence.

Example:

```
        —
        LDKL    A3,CIO
        LDKL    A4,SST
        —
        —
CIO     CIO     A1,1,TY
        EXR*    A4          EXECUTE SST
        RB(4)   *—2
        —
        EXR*    A3          EXECUTE CIO
        —
SST     SST     A7,TY
        RB(4)   *—2
```

The Execute instruction may not refer to other EX, EXK or EXR instructions or to Call Function, RTN or double format instructions.

SHIFT INSTRUCTIONS

Shift instructions operate on a bit level. These instructions allow to rotate the contents of one of the registers A1 thru A7 n positions in the direction and manner specified in the instruction.

CONTROL INSTRUCTIONS

These instructions perform the control of the program by allowing the program to be interrupted or not or to reset an internal interrupt. Except for the LKM instruction, control instructions should only be used in Stand Alone programming.
INH and ENB are two companion instructions. The program part between these instructions is not interrupted as INH inhibits all interrupts. ENB sets the machine status to permit interrupts.

Example:

```
        IDENT   TEST
        AORG    /1000
        RES     1
STACK   DATA    0
BUF     RES     2
START   INH                          ┐
        LDKL    A14,STACK            │
        LDK     A5,4                 │
        LDKL    BUF                  │
        CF      A14,INPUT            │  program inhibited
        HALT                         │
INPUT   LDR     A1,A5                │
        —                            │
        —                            │
        RTN     A14                  ┘
```

The RIT instruction is used to reset an internal interrupt which was previously set by an interrupt from the control panel, power failure/automatic restart, real-time clock or by a program error.
The programmer may specify a 5-bit hexadecimal value in the operand of this instruction to clear specific interrupts.

```
INTRTC  RIT     /1B     Reset the real-time clock interrupt
```

I/O INSTRUCTIONS

I/O instructions handle the data transfer between the cpu and peripherals, the operation of control units for these peripherals and status control.
In monitor controlled programs the I/O functions, initiated by these instructions, are taken over by a general I/O routine which is called each time a LKM instruction followed by a DATA directive is used.
The user need therefore not to write his own I/O routines. When the programmer is to write a Stand Alone program he has to write his own I/O routines.
Since there is no memory protection option the programmer must be careful not to overwrite parts of a program already in memory.
Two instructions, RER and WER, may be used to address an external register. The function of these instructions is described on page 2.35.

Directives are used to provide a framework for a program and to guide the assembly process. The directives are written in the program and are printed on the assembly listing if the listing option is specified in the Assembler option message (see page 3.3).

The table below gives a survey of which directives are accepted by which Assembler.

Directive	Meaning	page
IDENT	Program identification	2.18
END	End of assembly	2.18
ENTRY	Define entry point name	2.19
EXTRN	Define external references	2.20
COMN	Define common blocks	2.20
STAB	Define internal symbol table	2.23
AORG	Assign absolute origin	2.23
RORG	Assign relative origin	2.23
IFF	If false	2.20
IFT	If true	2.22
XIF	End of condition	2.22
DATA	Data generation	2.24
EQU	Equate symbol to value	2.25
RES	Reserve memory area	2.26
EJECT	Continue listing on new page	2.27
LIST	Resume listing output	2.27
NLIST	Suspend listing output	2.27
FORM	Format definition	2.28
XFORM	Extension of FORM directive	2.31
GEN	Generation directive	2.31

The directives can be divided in the following groups according to their function:

- Program framework : IDENT, END
- Linkage control : ENTRY, EXTRN, COMN
- Assembly control : IFT, IFF, XIF, STAB, AORG, RORG
- Value definition : EQU, DATA
- Area reservation : RES
- Listing control : NLIST, LIST, EJECT
- Symbol generation : FORM, XFORM, GEN

PROGRAM FRAMEWORK

The directives IDENT and END form respectively the first and last statements in the module. They are mandatory. The module punched on tape must be followed by :EOS or :EOF.
The IDENT directive is used for identification purposes and the END directive generates the END cluster after which the assembly process is stopped and a symbol table is printed.

IDENT — program IDENTification — IDENT

The IDENT directive specifies the name to be given to the object module output by the Assembler. It is used for identification purposes in selective loading or updating (see parts on Linkage Editor and Update Package).
This directive must always be present and must be the first statement in a module.

Syntax

␣IDENT␣ < module name >

where:

< module name >	A symbol which is specified according to the rules for a label.

END — END of assembly — END

This directive must be the last statement in a module and terminates the assembly process by output of an :EOS mark.

Syntax

[< label >]␣END␣[< predefined expression >][, < symbol >]

where:

< label >	The label is given a relocatable value equal to the length of the relocatable section of the generated object program. This length includes the length of the optional symbol table (see STAB directive, page 2.23). The value is 0 if this module is absolute.
< predefined expression >	This expression, if present, gives the address of the first instruction to be performed in the program after loading.
< symbol >	This parameter gives an entry point name to the internal symbol table of the generated object program when the STAB directive has been assembled.

LINKAGE CONTROL

Some modules which have to be grouped into one larger program contain references to identifiers defined in other modules.
By means of the directives ENTRY and EXTRN the user is able to refer to certain parts in other modules whereas the directive COMN allows to transfer data among several modules either written in Assembly Language or in FORTRAN.
By using a COMN the programmer can define one or more common blocks. Each common block may be divided in a number of subfields of varying length, each having a symbolic name which can be referred to directly but only in the module in which they are declared.
COMN blocks may be labeled or blank; a COMN block is labeled if a name is attached to it.
The Linkage Editor allocates a space to the blank common block at the end of the link-load or link-edit run (see Linkage Editor). This block is placed at the end of the entire program.
Labeled commons are placed at the end of the first module that refers to it.

The ENTRY, EXTRN and COMN directives must always follow immediately after the IDENT directive and in this order, though it is not necessary that the ENTRY as well as EXTRN and COMN are specified.

So: IDENT, ENTRY, EXTRN, COMN or
IDENT, EXTRN, COMN or
IDENT, ENTRY, COMN etc.

ENTRY **define ENTRY point name** **ENTRY**

The ENTRY directive is used to declare entry points, i. e. labels which are defined in the current module and used as operands of another module.

Syntax

␣ENTRY␣<entry point name>[,<entry point name>, . . ,<entry point name>]

where:

<entry point name> Can be referred to by an operand of an instruction in another module. The maximum number of entry points which can be specified in one ENTRY directive is determined by the length of one line.

Example (see also EXTRN)

```
        IDENT   PROG
        ENTRY   NUMB1, NUMB2, NUMB 3
        —
        —
        —
NUMB1   LDKL    A3, LABEL
        —
        —
NUMB2   ST      A6, REFER
        —
        —
NUMB3   CF      A14, EOS
        —
        —
        —
        END     START
```

EXTRN — define EXTeRNal references — EXTRN

The EXTRN directive is used to declare externals i.e. operands which are used in the current module and defined as labels in another module.

Syntax

␣EXTRN␣ <external name>[,<external name> ...,<external name>]

where:

<external name> Name of external reference (label in other module). The maximum number of external names which can be specified in one EXTRN directive is determined by the length of one line.

Example (see also ENTRY)

```
        IDENT   ASMPRO
        EXTRN   NUMB2
        —
        —
        —
        CF      A14, NUMB2
        —
        —
        —
        END     START
```

COMN — declare COMmoN block — COMN

The COMN directive facilitates communication between modules written in Assembly Language or FORTRAN. The directive is written as follows:

Syntax

[<label>]␣COMN␣ <common field definition list>

where:

<common field definition list> ::= <common field definition>[,<common field definition list>]

where:

<common field definition> ::= <common field name>[<common field length>]

where:

<common field name> ::= <identifier>
<common field length> ::= <predefined absolute expression>

If the parameter <common field length> is omitted the default value assumed by the Assembler is 1. The field length must be given in words.

Example
A␣COMN␣FVAL1 (3), FVAL2 (3), INTGV (10)

which defines a labeled common, named A, having the length

3 + 3 + 10 = 16 words.

A is defined as an external reference and common block name. Either the common block name itself or the subfield names may be referred to in the same module. The subfield names are then considered to be equivalent to:

<common block name> + <absolute displacement>

so,

LD␣A1, FVAL2 is equivalent to LD␣A1, A + 6

and

ST␣A2, INTGV + 18 is equivalent to ST␣A2, A + 30

The displacements in this example are counted in characters.
Blank commons can only be referred to by the subfield names defined in the operand field.

␣COMN␣VAL1 (3), VAL2 (4)
␣COMN␣VAL3 (9), VAL4 (10)

These directives define a blank common of 3 + 4 + 9 + 10 = 26 words.

VAL2, for instance, may be used in symbolic expressions and is equivalent to:

<blank common "name"> + 6

ASSEMBLY CONTROL

When it is neccessary to check whether a certain condition is satisfied before assembling a number of source lines, the user may include the directives IFT, IFF and XIF. The assembly of the IDENT – END – XIF directives are never bypassed by IFT or IFF.

By means of the STAB directive the user may specify one or more internal symbols which are to be used for Debugging purposes. All these symbols must have been defined previously in the current module.
Common block names are handled as externals.

The RORG and AORG directives are used to reset the location counter to a relocatable or absolute value indicated in the operands of those two directives.

Those directives are only used in combination with the directive XIF to indicate that a block of instructions is to be assembled only if a certain condition is fulfilled. The assembly of the IDENT – END – XIF directives are never bypassed.

IFT (IF True)

The IFT directive specifies that the Assembler has to assemble the next source lines only if the condition stated by this directive is fulfilled.

Syntax
␣IFT␣ < predefined absolute expression > = < predefined absolute expression >

If the first parameter ≠ second parameter the source line(s) following IFT up to the next XIF directive are not assembled.

IFF (IF False)

Syntax
␣IFF␣ < predefined absolute expression > = < predefined absolute expression >

If the first parameter = the second parameter the source lines following IFF will not be assembled.

Syntax
␣XIF␣

This directive allows all subsequent statements to be assembled until a new IFT or IFF statement is encountered.

STAB — define internal Symbol TABle — STAB

The STAB directive outputs at the end of the relocatable program section of the generated module one or several internal symbols to be used for Debugging purposes (internal symbol is the address given to a symbol in the program after assembly). All symbols must have been declared previously in the current module.

Syntax

␣STAB␣ <internal symbol list>

where:

<internal symbol list> :: = <internal symbol> [, <internal symbol list>]

If the STAB directive does not contain a parameter in the operand field all internal symbols of the module will be included.
The programmer may not specify external reference names or commons. This directive is only taken into account when in the END directive the parameter <symbol> is specified which gives the name of the internal symbol table.

AORG — Assign absolute ORiGin — AORG

This directives assigns an even absolute value to the location counter. The location counter receives that value by specifying <predefined absolute expression>.
From the time AORG is given and until a RORG directive is given the location counter is incremented in the same way as if it were relative, i.e. by increments of 2 and 4 depending on the length of the instruction. All labels in an absolute module are given an absolute value unless they are equated to a predefined relative value by an EQU directive.
RB and RF instructions in an absolute program cannot refer to an address in a relocatable program section as the place from where this section will be loaded is not known.

Syntax

␣AORG␣ <predefined absolute expression>

RORG — assign Relative ORiGin — RORG

The RORG directive allows the user to specify the beginning of a relocatable module by assigning a relative value, which must always be even, to the location counter. Its value may never become negative. If RORG has no operand the location counter is given the last relocatable value it has previously received. This value is equal to the length of the relocatable module at the time this directive is assembled.

Syntax

␣RORG␣ [<predefined relocatable expression>]

VALUE DEFINITION

The directives DATA and EQU are used to define certain values in a module.

DATA **DATA generation** **DATA**

The DATA directive is used to assign a value to one or more words in the module, for inclusion in the object module.

Syntax

[<label>]␣DATA␣ <data expression>

where:

<data expression> :: = [<expression>|'<character string>'][<data expression>]

<label> refers to a symbol in the operand field elsewhere in the module.

<data expression> the data expression may be:

- a decimal or hexadecimal constant
- an address expression
- a character string consisting of one to thirty-two ASCII characters enclosed by single quote marks. A series of words is generated, of two characters each, which are left justified. When the number of characters is odd the rightmost character of the last word is a space.

Example

The expression may contain a number of parameters which, in total, may generate no more than 16 words in memory.

␣DATA␣'ABC',/0A0D, 1,/A, 2,'DEF'

will generate the following words:

4142	'AB
4320	C␣
0A0D	/0A0D
0001	1
000A	/A
0002	2
4445	'DE
4620	F'␣

Example
When the user wishes to make an ECB he may do so as follows:

ECB␣DATA␣1, BUF2, 6, 0, 0, 0,

Example
DATA␣—0128, + 12,/3AB,—/A, LABEL, 'TEXT:'

will generate the following:

FF80	-128
000C	+ 12
03AB	/3AB
FFF6	—/A
<value>	LABEL
5445	'TE
5954	XT'
3A20	:␣

EQU **EQUate symbol to value** **EQU**

Identifiers are normally defined by being assigned memory values as they appear in the label field of an instruction. The EQU directive may be used to define an indentifier in a direct manner by assigning to it the value of an expression in the operand field. The symbol in the label field is made equivalent to the value in that operand field. This value may be absolute or relocatable.
A symbol, provided it differs from standard mnemonics and FORM-defined mnemonics, may be used as an operation mnemonic but may not be followed by an operand. The Assembler generates one code word each time this mnemonic appears in the operand field.

Syntax

<label>␣EQU␣<predefined expression>

Example
CT␣ EQU␣/41C4 CT may now be used anywhere in the program to represent the value /41C4.

```
        —
        —
        CT
        LDKL␣A1, CT
```

Example
VAL␣ EQU␣10

```
        —
        —
        —
        LDK ␣ A1, VAL
```

Example
LAB ␣ EQU ␣ * LAB receives the value of the location counter.

AREA RESERVATION

The directive RES can ce used to skip over an area in memory. The RES directive saves a memory area of a given length, specified in the operand, advancing the location counter by twice the number of words specified.

RES **REServe memory area** **RES**

The RES directive is used to reserve a number of memory words. The programmer may specify this number in the parameter. The location counter is incremented or decremented depending on the positive or negative value of that parameter. If positive, a memory area of the specified value is reserved. If negative, a memory area of the specified size before the place identified by <label>.
The value of the latter is not changed but the location counter is reset to a lower value by subtracting twice the value specified.

[<label>]␣RES␣<predefined absolute expression>

where:

<label>	receives the address of the first word of the reserved area.
<predefined absolute expression>	specifies the length of the area to be reserved.

If <predefined absolute expression> is 0 the location counter is not updated and, if <label> is specified, the statement is equivalent to

<label>␣EQU␣*

Examples:

```
        RES␣4      Reserve 4 words
LAB1␣   RES␣-2     Reserve 2 words before LAB1
INS␣    RES␣0      INS receives the value of the location counter.
```

LISTING CONTROL

The Assembler normally produces an output listing for each assembly. By means of the directives EJECT, NLIST and LIST the programmer may determine which parts of the modules do not need to be listed.

EJECT **Continue listing on new page** **EJECT**

This directive causes the remainder of the current page of the line printer paper to be left blank and the listing to be continued at the top of next page.

Syntax
␣EJECT␣

NLIST **Suspend listing** **NLIST**

The NLIST directive causes the Assembler listing to be suspended from the point where this directive is given until either the END directive or a LIST directive.
Lines which contain errors will continue to be printed during this phase.

Syntax
␣NLIST␣

LIST **Resume listing** **LIST**

The LIST directive causes the Assembler to resume the listing after it has been suspended by a NLIST directive.

Syntax
␣LIST␣

SYMBOL GENERATION

Three directives allow the user to make a number of special instructions for a specific purpose or program, namely FORM, XFORM and GEN. In the FORM directive the user may define the bit configuration and the mnemonic of the special instruction.
If two FORM-defined instructions are to be specified which differ only in the contents of certain fields the programmer may use the XFORM directive.
The GEN directive allows to include the instructions, defined by FORM and XFORM, in the existing Assembler by extending the Assembler's symbol table.
A particular useful pseudo-instruction or system macro can be defined once for all times instead of having to be generated by a FORM directive in every program where it is used.

FORM — FORMat definition — FORM

This directive is used to define the format of a word or a group of up to 8 words named by an identifier which can be used as an instruction mnemonic later in the program.
The directive is written as follows:

Syntax
< label > ␣ FORM ␣ < field definition > [, < field definition >, < field definition > ... < field definition >] [/ < field number list >]

where:

< field definition > ::= < field length definition > [[= |:] field value definition >]
< field number list > ::= < field number > [, < field number list >]

and

< field number > ::= < decimal integer >

< **field length definition** > specifies the number of bits to be allocated to a field of the word and may range from 1 through 16. If several fields are defined inside a word the sum of the field lengths must be 16. The maximum number of consecutive words defined by a single FORM directive is 8.

< **field value definition** > can be used to place a value in the field to which it refers when the value is preceded by an equal sign (=).
If the value is preceded by a colon (:) the value indicates the address of a word in relation to the first word of the expansion defined by FORM. The value definition itself may be a predefined expression, an external reference without any displacement or a predefined absolute or relocatable expression. If a particular field has not received a value definition the field will be filled wit zeroes.

< **label** > defines the instruction mnemonic. The operand field of the directive must then contain values to be placed in any non-predefined fields. The last non-predefined value is default value.

Example
MNEM␣FORM␣16=/85A0,16:14,16=/8141,16=INST, 16, 16, 16

/85A0	→arithmetic or logical value
MNEM + 14	→address of word following this block
/8141	→arithmetic or logical value
INST	→identifier
0-0	
0-0	3 words containing zeroes
0-0	

The parameter 16:14 indicates a word address seven words form the beginning of the expansion defined by FORM. The programmer has to specify this address as the last three words are left zero.

Example
This example shows how the programmer may make an ECB if not all parameters are known. By using the FORM directive he does not have to write the instruction sequence:

```
LDK     A7, –
LDKL    A8, DECB
LKM
DATA    1
```

```
00000                           IDENT   FORM
00001                   INOUT   FORM    8=/07,8,16=/80A0,16,16=/2804,16=1
00002  0000             BUFFER  RES     10
00003  0014  0008       DECB    DATA    8,BUFFER,20,0,0,0
       0016  0000 R
       0018  0014
       001A  0000
       001C  0000
       001E  0000
00004  0020  0782       START   INOUT   /82,DECB
       0022  80A0
       0024  0014 R
       0026  2804
       0028  0001
00005  002A  2804               LKM
00006  002C  0003               DATA    3
00007                           END     START
       SYMBOL TABLE
BUFFER  0000 R  DECB  0014 R  START  0020  R
ASS.ERR.  00000
:EOF
A::EOF
EXIT
```

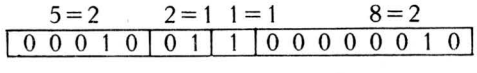
From now on the programmer may use INOUT␣/82, DECB instead of LDK␣A7,...

Field number list
If the programmer wishes to put the values of the operand field of the FORM defined mnemonic in an order different from that of the non-predefined field they are to occupy, or if the user wishes to alter the values held by any of the predefined fields, he must use the field number list parameter in the FORM directive.

Each field that is generated is given a number, beginning with 0 for the first field, 1 for the second field, n-1 for the nth field (n may not exceed 15).

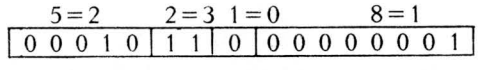
The field number list must be preceded by a / (slash) and be placed after the last field definition of the FORM directive.
All not predefined fields specified in the field definition list must also be specified in the field number list.
A field number is represented as a decimal integer.
If a field number list is specified after a FORM directive, the operand expressions following the pseudo-mnemonic will occupy the fields specified in the field number list in the given order. In this way, the contents of predefined fields may be altered while blank fields may be left blank.

Example:
Suppose the user has specified in his program, by means of a FORM directive, a 16-bit word of the following format:



field no 0 1 2 3

He wishes to have this word changed in:



field no 0 1 2 3

He may do so by using the following instruction sequence in his program using the field number list in the FORM directive

```
        IDENT   EXAM
        —
        —
WORD    FORM    5=2, 2=1, 1=1, 8=2/2,1,3
        —
        —
        WORD    0,3,1
        —
        —
        END
```

The Extended Assembler will now change the fields as follows:
field no 2 (1=1) will be changed to contain the value 0
field no 1 (2=1) will be changed to contain the value 3
field no 3 (8=2) will be changed to contain the value 1
field no 0 (5=2) will keep the value 2

The operand expressions following a pseudo-mnemonic are positional parameters. If one parameter is omitted (other than the rightmost one), its position must be indicated by a comma.

If a FORM defined mnemonic is identical with a standard instruction mnemonic, the pseudo-mnemonic is given priority.

XFORM — eXtension of a FORM directive — XFORM

Syntax

<label> ␣XFORM␣ <FORM-defined pseudo-mnemonic>,<field list>

The XFORM may be used each time two FORM-defined pseudo-mnemonics have to be defined which do not differ in the format but only in the values of the predefined fields.
The field list is a series of field definitions giving the format of the new pseudo-mnemonic and the contents of its fields.
The field length definitions must be the same as those of the FORM-directive referred to and appear in the same order.

Example

```
INST1␣FORM␣8=/FF, 4, 4, 16/1, 3, 2
INST2␣FORM␣8=/33,4,4,16/1,3,2
```

The XFORM directive combines the two and generates an INST2 instruction as follows:

```
INST2␣XFORM␣INST1,8=/33,4,4,16
```

GEN — GENeration directive — GEN

The GEN directive allows to extend the Assembler symbol table so that it recognizes and assembles a number of non-standard symbols in any program in which they are used.

Syntax

␣GEN␣

Restrictions
The GEN directive may only be used in the source program in which it appears if it fulfills the following conditions:

- GEN must immediately precede END
- only the FORM, XFORM, EQU and EXTRN directives are allowed in this program

The Assembler does not verify if those conditions are fulfilled. It checks only if:

- object code is produced
- assembly errors have occurred

Example

```
        IDENT␣FORM
INOUT␣  FORM␣8 = /07,8,16 = /80A0,16,16 = /2804,16 = 1
        GEN
        END
```

The following procedure must be followed to include the features provided by GEN:

— load Assembler
— place on the reader the user source module with GEN directive
— assemble this module to produce object output
— load Linkage Editor
— place the Assembler in the reader and have it processed by the Linkage Editor (P)
— place the object user program in the reader and have it processed (P)
— next Terminate (T).

The punched output of this link-editing is the original Assembler extended with one or more new mnemonics.

List of predefined symbols

NAME	MEANING	PREDEFINED VALUE	INTERNAL VALUE
P	Instruction Counter	0	0
A1	Register 1	1	2
A2	Register 2	2	4
A3	Register 3	3	6
A4	Register 4	4	8
A5	Register 5	5	10
A6	Register 6	6	12
A7	Register 7	7	14
A8	Register 8	8	1
A9	Register 9	9	3
A10	Register 10	10	5
A11	Register 11	11	7
A12	Register 12	12	9
A13	Register 13	13	11
A14	Register 14	14	13
A15	stack pointer	15	15

Note: P, A1, A2, A3 etc. can only be used to call the registers. If they are used for other purposes an error message will be output.

Data transfers between input/output devices and the central processor are controlled by device control units each of which may have one or several devices attached to it, depending on the type of device. Control units are attached to the central processor by an interrupt or break line, by address lines and other signal lines which are used by the computer to determine whether a data transfer can be performed.
Data transfers take place through a channel, the General Purpose Bus. The actual programming of the data transfers may be on a character or word basis, where each word or character is programmed and transferred individually via the Programmed Channel or the user may program blocks of words or characters via the I/O Processor. In the latter case external registers may be addressed.

Interrupt system
When working in interrupt mode each interrupt program may be connected to an interrupt level. As the actioning of an interrupt involves the direct accessing of the interrupt level's start address from its hardware interrupt location, the contents of this location must have been previously loaded with the correct address.
The start addresses loaded in these locations are not fixed and must be defined by the programmer.

interrupt level	*hardware interrupt location*
0 to 62	/0000 to /007C

where level 0 has the highest priority and 62 the lowest. The levels are defined at SYSGEN time (see App. A).

System stack
To save the contents of registers when an interrupt is made into the main program, the hardware interrupt routine automatically uses register A15. This register addresses the stack which is to hold the contents of the P-register and the Program Status Word at the time the program was interrupted. It is therefore necessary to reserve sufficient space for the stack and to load register A15 with its start address. This may be done by using the appropriate assembly directives and by defining the start address by means of an identifier. The start address is the highest address reserved as the stack is filled from the high towards the lower addresses.
Apart from the contents of the P-register and PSW, the stack may be used to save the contents of other registers as required by the program. These registers are saved by means of Store instructions (1 for each register). Before returning to the main program, Load instructions are required to restore the contents of the stack, prior to RTN. During the hardware action further interrupts are inhibited. If the user wishes to allow the specific routine to be interrupted he must give an ENB instruction.

User stack

We have seen that with the A15 stack the P-register, the PSW and any other registers are saved with Store instructions in this stack towards the lower addresses. Now, if a user calls a subroutine with a CF instruction the contents of the P-register and the PSW are automatically stored in a stack he has set up previously, for example as follows:

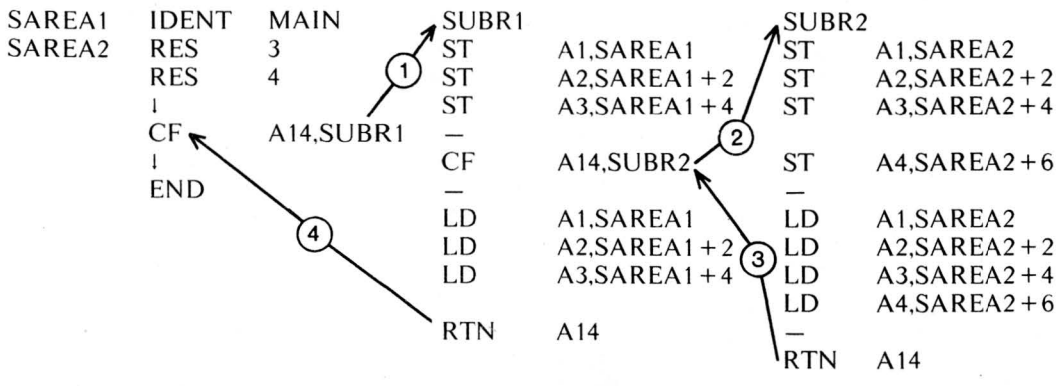
```
        RES     20
STB     EQU     *-2
        LDKL    A14,STB

        CF      A14,SUBR
```

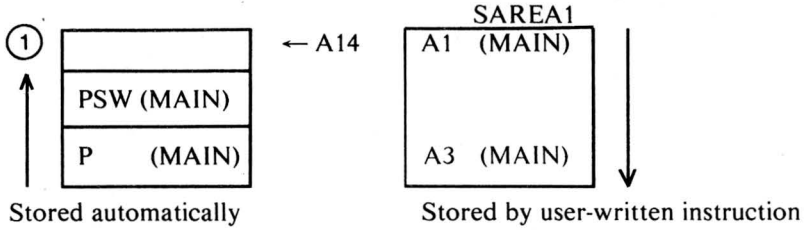
then the subroutine is called:

and P and PSW are stored in the A14 stack
(other registers may also be used as a stackpointer)

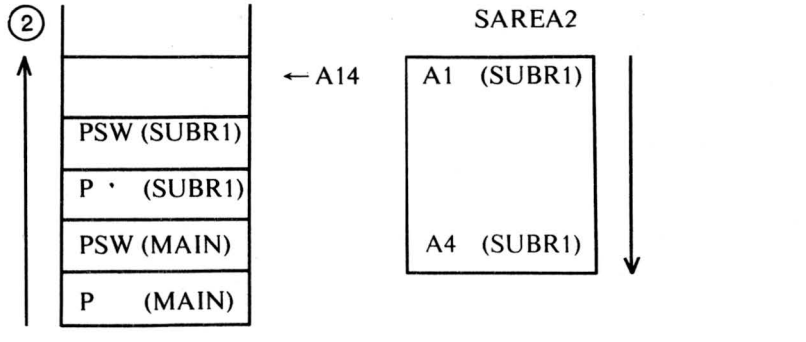
For example, for a program with two subroutines, one subroutine calling another one, the saving may be done as follows:



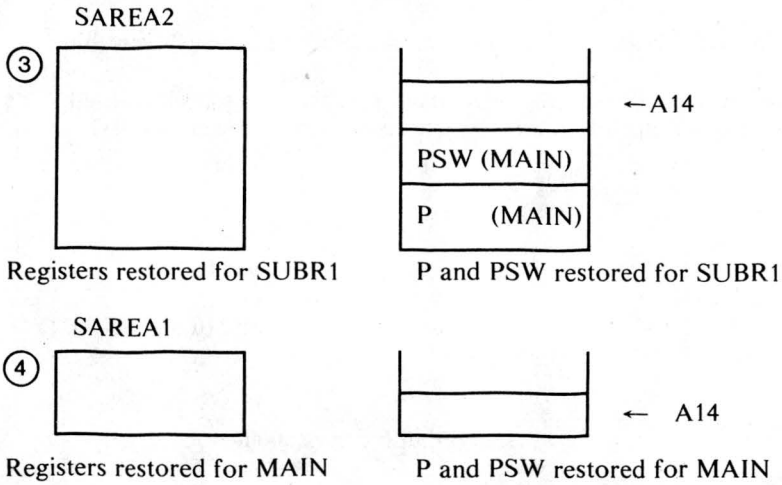
The following save operations take place in this example:



Stored automatically — Stored by user-written instruction

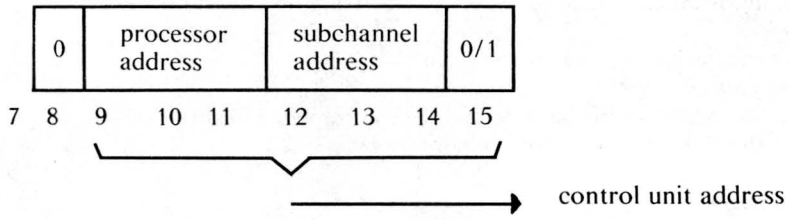


Stored automatically — Stored by user-written instruction



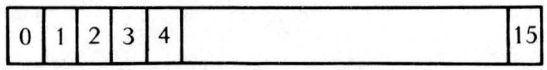
Note:
It is possible to return from SUBR2 directly to the main program but in such a case the user must update the A14 register content i.e. the stackpointer himself (with 4, in this case).

I/O Processor
The I/O processor allows the high speed transfer of variable length or fixed length data blocks between a suitable control unit and the processor.
Up to eight I/O processors may be connected to the General Purpose Bus each of which may control up to eight control units via eight subchannels.
Each I/O processor has implemented two working registers which are used to effect register to register exchanges with the CPU internal registers.
Before a data transfer can be realised the user has to specify two control words for two external registers. These external registers are addressed by 2 WER instructions in which the address part must be composed as follows:

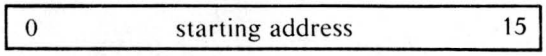


where processor and subchannel address are determined at system installation time. Both addresses, which may range from 0 thru 7, form together the attached control unit address. Bit 15 determines which control word is sent:
bit 15 = 0 1st control word
1 2nd control word

Format of control words
The format of the first control word is:



where:
bit 0 = 1 exchange is in word mode
0 exchange is in character mode
bit 1 = 1 exchange is from memory to control unit (output)
0 exchange is from control unit to memory (input)
bit 2 = 0
bit 3 = 0
bits 4 thru 15 specify the number of characters or words to be transferred.
The format of the second control word is:



When operating in *word mode* the 1st word of the block is always even (bit 15 = 0)
In *character mode*, and bit 15 = 1, the right hand character is addressed (odd address). When bit 15 = 0 the left hand character is addressed (even address).

Example:

```
—
—
LDKL    A1,/8032        word mode, input, 50 words
LDKL    A2,BUF          starting address of block
WER     A1,/02          send control words (0000010 and 0000011)
WER     A2,/03
—
—
CIO     A4,1,/01        start input (address: 000001)
—
—
```

The RER instruction may now be used to read a transfer's effective length after termination of the I/O operation.
When the exchange is completed an SST instruction should check the status of the control unit and set it to the *inactive* state. The control unit may now be re-initialised for a new transfer.

Trap action
Instructions input to the P852M computer are checked and decoded by the CPU's hardware.
If an unexecutable instruction is encountered a *trap* action is started which consists of a hardware and software operation. The hardware operation of the trap consists of the following actions:
— the CPU does not attempt to carry out the instruction
— interrupts are inhibited
— information which refers to the instruction's address and processor status (P and PSW) are saved
— an indirect branch is made to location /7E (start of trap routine).

The software operation of the trap consists of:
— save the address in P
— save the instruction's bit pattern and its second word, if any
— activate the Simulation routine (see below).

Simulation routine
The simulation routine allows the P852M user to run P855M software on his machine by simulating the following P855M instructions:

multiply	double shift
divide	multiple load
double add	multiple store
double subtract	

This routine, which is activated each time an illegal instruction code is met in the instruction sequence, consists of two parts. One part analyzing the bit pattern saved by the trap routine and one part executing the instruction listed above.
The routine may be interrupted.
If the instruction encountered is not one of the P855M instructions a branch is made to the system abort routine.

PART 3 ASSEMBLER

INPUT

The input to the Assembler consists of source modules written in the assembly language described in PART 2 and of control messages which are typed in from the operator's typewriter.

Once the Assembler is loaded and given control it types out on the operator's typewriter the message:

```
A:
```

The operator may now decide to type in an option control message or he may decide to use the standard options.

Option control message
The syntax of this message is:

[<4 hexadecimal digits>][N][R][Q][X]

where:

[<4 hexadecimal digits>]	Each of those digits (except for the second) indicate a file code. The highest allowed file code is F. For the description of file codes see Part1, Chapter 5.
first digit	source input file code.
second digit	not taken into account. type in 0
third digit	listing output file code. The operator may not type in 0 if this option is not required, because this is controlled by parameter N.
fourth digit	object output file code. type in 0 if no object output is wanted.

N specifies that no assembly listing is wanted, but the IDENT and END statements, the symbol table, and erroneous statements are always printed out.

R specifies that the user wishes to have the possibility to correct recoverable errors. See also page 3.5).

Q specifies that the object code output from the tape punch has to be in 4×4 format.

X specifies exit after assembly.

When an error was encountered in this option control message the Assembler asks for a new option message by typing out again:

```
A:
```

after which the correct option message has to be input.

Operator messages
If the operator decides not to use an option message he may type in, after the first time A: appeared:

LF CR

which makes the Assembler choose the standard options 1 0 2 3:
i.e.
standard source input
file code 1

standard listing output
file code 2

standard object output
file code 3

When an :EOS is read on the input file, the Assembler does not stop processing but starts reading the next record and goes on until an END instruction is read, when the Assembler outputs an :EOS label on the output file.
When an :EOF is detected on the input file, the Assembler types out the message
:EOF
and gives control to the operator, who may then type in one of two answers:

X
upon which the Assembler will exit, or:

:EOF
upon which the Assembler will write an :EOF label on the output file and then exit.

The Assembler starts reading the input file after the correct option message is typed in or when LF CR is given.

The statement lines are read one by one. Each statement line is evaluated on its separate components i.e. the label field, the operation field, the operand field and the comments field.

If the user has asked for a listing each read line is output on the listing device.

The Assembler builds clusters in order to output an object program consisting of those clusters (see Appendix C: Object code). As many object modules are produced as there are source modules.

Errors during processing
When a statement in a source module contains an error, and if the user has specified the error recovery option (R), the Assembler types out the control message

```
R:
```

The operator may then answer with

```
LF CR
```

if the error is not to be recovered it will become definitive.
The error routine includes in the object code the instructions: HLT
RB␣*−2

or with

a new source statement line which contains the correct items or a comment line (it may never be :EOS or :EOF), followed by LF CR.
On page 2-12 is described how the user may introduce his source lines from the operator's typewriter.

Error recovery not wanted
The processing is continued until :EOF and in the meantime an error counter is incremented by one each time an error is encountered.
The number of errors is given after the assembly of each module on the assembly listing. This value is in decimal notation.

Handling of special errors
Some errors are processed in a different way from the other errors whether R is specified or not.

- Illegal use of forward references is only mentioned at the end of the assembly.
- When the END statement is missing in the source module a standard END statement is built, which does not contain a label, start address or symbol table entry point name.
- If the IDENT statement is missing all statements following are read and bypassed until the next IDENT statement or END statement.

When core overflow occurs the assembly is stopped and an END cluster is output. Then all following statements are bypassed up to the next IDENT or END statement.

When an I/O error occurs, an error message is output on the operators typewriter and assembly is stopped. See also system messages in Part 1, Chapter 9.

Output of typewriter	Answer from the operator
A:	Type in the option control message or LF CR or :EOF
R:	Type in a source correction statement or LF CR
:EOF	An :EOF has been read. Type in :EOF or X.
I/O ERROR xxxx yyyy	This message appears only for the monitor controlled versions. xxxx hexadecimal representation of the file code on which an I/O operation was executed. yyyy hexadecimal representation of the status of the I/O device.

Assembly listing
The Assembly listing may be printed on either the operator's typewriter or the line printer, depending on the configuration and option control message.
Each source statement of the module is represented by all or some of the following components where:

line number	A decimal number indicating the place of the statement in the module.
L.C. value	Location counter value (hexadecimal). This field remains blank for all directives except for the RES and DATA directives.
code	Hexadecimal representation of the generated code.
R	The code is relocatable.
X	The code contains an external reference which has to be matched by the Linkage Editor.
F	The codes makes a forward reference.
source statement representation	The source statement is represented as it was written in the program, except when back slashes are used.

Symbol table
A symbol table is printed after each module when the END statement is processed. Each symbol is printed together with its value and its type. The value is given as a four-digit hexadecimal value. The type is indicated as follows:

A The symbol is absolute.
R The symbol is relocatable.
X The symbol is an external reference name.
** The symbol is undefined and the value is the value of the instruction counter at the time the instruction, in which the symbol appeared, was encountered.

The symbol table may be followed by one or more of the following messages:

UND.ENT ----- Indicating the number of undefined entry points

UND.LAB ----- References to missing labels in this particular module

FOR.O/R ----- The forward reference with address ----- contained one of the following two mistakes:
— the value > 255 for the 8 least significant bits.
— value specified was not absolute.

ASS.ERR ----- Giving the number of errors in this particular module.

Output of error messages in the assembly listing

Non-fatal errors
When a non-fatal error is detected during the assembly, two lines are printed whether the listing option is requested or not. The items printed have the following meaning:

* error code, sequence number, location counter value,/207F (Halt instruction) and the erroneous statement.
******* location counter value representation,/5F04 (RB␣*−2)

An * is printed under the place where the error is detected.

Fatal error
When a fatal error is detected the Assembler prints out an error message, on one line, in the following format:

******error code (one letter), sequence number, image of source statement. A list of all error messages is given on page 3.9.

The following error messages are output by the Assembler:

CODE	MEANING	DESCRIPTION
*I	Illegal identifier	The first character of a symbol must be a letter.
*C	Illegal constant	– "constant" overflow. – A constant with hexadecimal value must begin with X / and end with /. – A constant should not have been written here. – Hexadecimal constant written either X″ or /″.
*X	Illegal expression	– More than two symbols defined. – More than three terms in the expression. – An external reference and a forward reference have been specified in the same expression. – An external reference is preceded by a minus sign. – A plus or minus sign is not followed by a term. – A forward reference or external reference is specified in a requested predefined expression. – A register expression may not contain more than one term.
*R	Illegal relocation	– Either a predefined expression or predefined relocatable section has been input. – Too many relocatable symbols are added or subtracted from each other. – The expression is equal to the result of a subtraction of a relocatable part from an absolute part. – If an external reference is specified the displacement value must be absolute. – The instruction code operation defined by an EQU directive must be absolute.
*L	Illegal label	– The label has been defined previously as: – a symbol name – an external reference name – entry point name. – A label has been given where it was not allowed. – A label must be specified.

CODE	MEANING	DESCRIPTION
*P	Illegal parameter	– Too many parameters specified in the operand of an instruction, in the pseudo instruction or directive. – Not enough parameters specified in the operand of an instruction, defined pseudo instruction or directive. – A parameter in the STAB directive may not be an entry point name, a COMMON name or a forward reference. – The operand in a DATA directive may not give more than 16 code words. – " is not a character string. – Illegal use of a register name in a standard instruction operand.
*O	Overdisplacement	– Displacement value of parameter too large.
*E	Not an even address	– The specified start address is not even. – The specified AORG or RORG operand is not even.
*M	Unknown mnemonic	– Unknown mnemonic. – Unknown condition mnemonic.
*S	Illegal statement	– The ENTRY or EXTRN or COMN directive is no longer acceptable. – The directive does not need an operand. – The directive needs an operand. – Invalid character. – Invalid indirect addressing. – Invalid condition specification. – The label is not followed by an operation code. – '(' is not followed by')'. – The operand value of RES directive makes the instruction counter value negative. – GEN cannot produce any code as either any error occurred or the code word has already been produced.
*F	Illegal FORM or XFORM directive	– An XFORM declared symbol must be linked to a FORM defined pseudo whose name is the first parameter of the operand. – More than 16 fields specified. – Negative field length. – The length of this field cannot be contained in 16 bits. It is too long.

CODE	MEANING	DESCRIPTION
		— A displacement value is not allowed when the predefinition concerns an external reference name. — The : predefinition is only allowed for a 16-bit field. — Invalid predefined value of a field (overdisplacement or negative value for a less than 16-bit field). — The division of the current word of an XFORM declaration is not the same as the corresponding word of the linked FORM symbol. — The predefinition of the fields of the current word of an XFORM declaration are not the same as the corresponding word of the linked FORM symbol. — More than 8 words described by a FORM declaration. — More than the number of words described by the linked FORM symbol described by an XFORM declaration. — The field number specified in the syntax definition line is invalid. — Twice the same field specified in the syntax definition line.
******O	Core overflow	— Fatal error. Too many symbols or forward references used.
******E	End missing	— Fatal error. The END statement is missing.
******I	IDENT missing	— Fatal error. The IDENT statement is missing.

PART 4 LINKAGE EDITOR

A source program can consist of modules which perform certain functions. Each module of the program is separately assembled or compiled by the Assembler or FORTRAN compiler. The output of those language translators is a collection of object modules consisting of clusters.
Those modules cannot be loaded into memory as they are produced in an unexecutable machine code.

The function of the Linkage Editor is to join those modules, which make external references to one another, together. The product of this linkage is another, larger program, which may be loaded into memory during the linkage process when all external references are matched (link-load) or which may be output on an output medium for storage, further linkage or loading by the program loader (link-edit).

Link-edit mode
In this mode the Linkage Editor joins a number of object modules coming from the language translators, the Linkage Editor itself or object libraries, into one larger object program which may still contain external references and outputs it onto a suitable medium.
If all external references are satisfied the program may be loaded by the program loader and executed; if not, it may be stored in an object module library or be input to a further linkage process.

Link-load mode
In this mode the Linkage Editor loads a number of object modules into memory where all external references will be matched. The user must ensure that all external references can be satisfied.
If there is insufficient space in memory for all the object modules to be loaded the Linkage Editor will stop processing and print an error message.

Due to his configuration the user must take the following restrictions into account.
The memory size determines the maximum size of a program which has to be processed in memory and its maximum number of entry points in a linkage operation. Moreover it determines the maximum number of unsatisfied external references and the maximum number of entry points during a link-edit process.

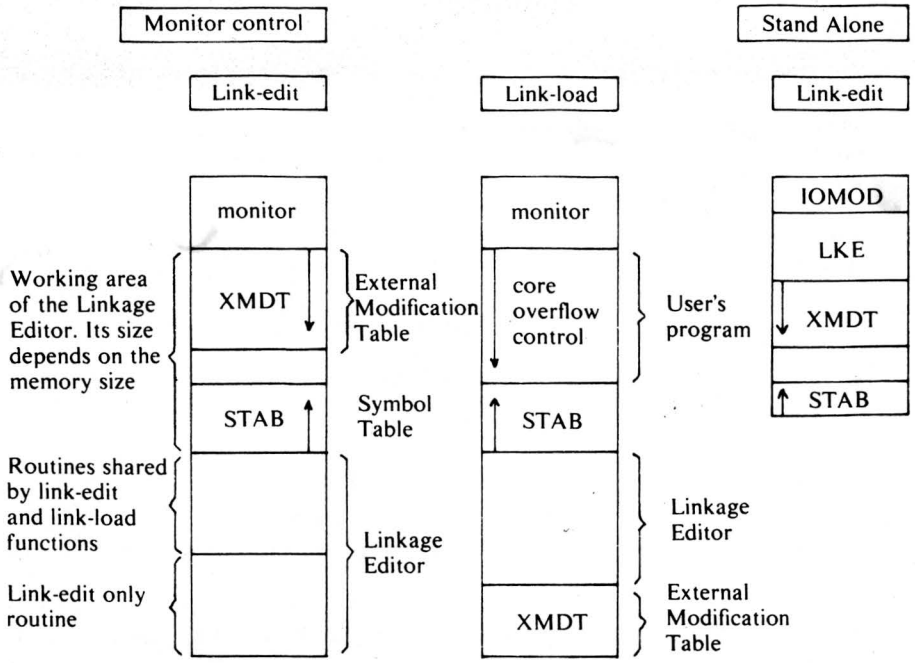
Linkage Editor Generation

The user may generate his own Linkage Editor which has to run under monitor control.

A description of how such a Linkage Editor is generated is given in Appendix A, System Generation, in the section CASLOD.

Memory layout

The figures below show the memory layout for the link-edit function of the Stand Alone Linkage Editor and the link-edit and link-load functions of the Linkage Editor running under Monitor supervision.



Once the processor has received control it types out onto the operator's typewriter:

```
L:
```

and waits for the operator to input the options as described in the option control message or for the default options.
If an erroneous parameter was given in the option message the Linkage Editor types out:

```
L?
```

and the operator must type in the correct message.

The processor will now start a link-edit or link-load operation. The linkage process is controlled by the messages typed in by the operator.

Link-edit operation
In case of a link-edit operation the processor punches, in ASCII, the name of the load module to be generated, as chosen in the option message and types out onto the operator's typewriter:

```
L:
```

The operator may now introduce a control message or a series of control messages.
The processor starts reading the input file each time a P, S or L control message is given and processes the records of the input stream according to their type.
All entry points encountered in the modules are placed in an entry points table. Also the names and lengths of labeled commons are placed in this table when their names have not yet been encountered in the input file. When a name of a labeled common is met the common's length is compared with the length previously defined and the new size will be smaller than or equal to the old size.
Relative addresses of relocatable code clusters are changed in absolute addresses when an A control message has been introduced, or in a partly relocated relative address after an R control message.
The relocation is calculated by adding the relocation value specified in the A or R message to the sum of the lengths of all object modules processed from the A or R message on.
The length is zero for entirely absolute programs.
The code words are relocated by addition of the relocation value.

When externals are specified the code words are changed according to those externals when the latter are matched. Otherwise all information is kept in an external modification table until the externals are satisfied.
The externals listed in an X control message remain unsatisfied but are taken into account when the external reference cluster is modified.
When the external modification table is filled the processor tries to decrease the number of external modifications by satisfying external references each time an external name is defined as an entry point. Whenever this is possible an entry is made in an internal modification cluster and, if necessary, this cluster is output.

Absolute code clusters are processed in the same way as relocatable code clusters except for the address of code clusters which is never modified.

When all records of an object input module have been read, and if the next record is not an End Of File mark the Linkage Editor processes the next object module on the input file (according to the previous command given by the operator). If it is an End Of File mark, the Linkage Editor will type out:

```
L:
```

and waits for an operator control message. When he types in T the Linkage Editor will build the END cluster and prints the map.
Processing is then terminated.

Link-load operation
The processing of modules in link-load mode is almost similar to processing in link-edit mode.
Relocatable code clusters are processed by loading sequentially into memory all code words of the cluster, starting at an absolute address calculated by adding the relative address specified in the cluster to the absolute base address of the currently processed module.
The latter is computed by adding to the absolute address specified in the previous A control message, the sum of the lengths of all object modules processed since the last A control message (or since the beginning of the link-load process when no A message has been introduced).
The absolute code clusters are processed in the same way as the relocatable code clusters, except for the address of the code cluster which is never changed.

The input to the Linkage Editor may consist of:

— object modules
— object module libraries
— control messages.

Object modules
Object modules have the format as described in Appendix C: Object code. Each batch of modules is terminated by an: EOF (End Of File) mark.

Object Module Libraries
The difference between the handling of separate object modules and of libraries is that the input modules are all read and processed and that of a library only those modules are linked which are necessary for satisfaction of external references.

Control messages
The control messages, used to control the operation of the Linkage Editor, may be divided in:

— one option control message
— link-edit control messages
— link-edit and link-load control messages.

Option_Control_Message

The options of the Linkage Editor allow the user to specify:
- the function of the Linkage Editor he wishes to have performed (link-edit or link-load).

- the peripheral units he wishes to use as input and output media.
- the name to be given to the generated object module.
- the type of object output (cassette or 4x4 or 8+8 punched tape).

Link-load

⌈L:<number>⌉
where:
L link-load mode
:<number> a number of three hexadecimal digits indicating the file codes of respectively:
- object output file code: must be 0 (i.e. not used)
- listing output file code
- object input file code

The defaultvalue for this option message is L:24. To get this, type in (LF) (CR)

Link-edit

E⌈:<number>⌉,<ident>⌈,⌊4|8⌋⌉⌈,<start addr>⌊<common addr>|<start addr>,<common addr>⌉⌉
where:
E link-edit mode
number a number of three hexadecimal numbers indicating the file codes of respectively:
object output file code
listing file code
object input file code
If either output file is not required it must be replaced by a zero.
ident name, with a maximum of 6 characters, to be given to the load module.
4 the load module must be punched in 4x4 tape format when the ASR tape punch is used
8 the load module must be punched in 8+8 tape format. This is also the default value.
When either one of the following 2 parameters (common and start address) is used, 4 or 8 must be specified when the output is on punched tape. When the output is on cassette tape, 8 must be specified.
<start addr> start address for the new program
<common addr> start address to be given to a blank common. If bit 15 of this address is 1, it is a relocatable blank common, if it is 0 it is an absolute blank common (see also T command below).

Other control messages
The control messages of the Linkage Editor are used to provide the processor with information when it has partly executed the processing. Two control messages may only be used in link-edit mode. The other messages may be used in both modes.

All control messages are terminated by LF CR

The syntax of each control message is explained in the section on control messages. Since address and symbol appear in almost all messages as a parameter, they are explained here:

<address> :: = string of up to 4 hexadecimal digits. It is always a word address (even number)
<symbol> :: = up to 6 alphanumerical characters.

List of control messages

Message	Meaning
E␣ <entry points name list>	Define entry points (link-edit only)
X␣ <external reference names list>	Define external reference names (link-edit only)
R␣ <address>	Define relative base address
A␣ <address>	Define absolute base address
P	Process the input file up to EOF
S␣ <symbol>	Select a module with the ident. name <symbol>
L	Process library to solve unsatisfied external references
U	List names of all undefined external references.
T	Terminate processing
H	Pause

DEFINE ENTRY POINTS *(link-edit mode only)*

E␣ <entry points name list>

where:

<entry points name list> ::= <symbol>[,<symbol list>]

This control message causes the Linkage Editor to save a series of symbols as entry points in the generated object program. All control messages of this type must precede all others and must be grouped.

DEFINE EXTERNAL REFERENCE NAMES *(link-edit mode only)*

X␣ <external reference names list>

where:

<external reference names list> ::= <symbol>[, <symbol list>]

The symbols specified in this message indicate which external references must be left unsatisfied. The X control message may be used if the generated module is to be input to a further linkage process.
All X control messages, if present, must be grouped and be either the first of the messages or be the first following a (series of) E message(s).

DEFINE RELATIVE BASE ADDRESS FOR RELOCATABLE PROGRAM SECTIONS *(link-edit mode only)*

R␣[< address >]

This control message enables the user to define a relative base address for relocatable program sections. Any absolute program section within a relocatable program section will remain absolute.
The base address progresses from the value specified in this control message. When < address > is not specified the value is assumed to be zero. In either case this value is updated by the addition of the length of the modules input to the Linkage Editor each time such a module has been fully processed.
Relative addresses in code words are relocated in relation to the base address. All relocatable program sections processed following this control message will result in a relocatable program section being output by the Linkage Editor. This does not apply to absolute program sections and this message ceases to operate until an A message is given.

DEFINE ABSOLUTE BASE ADDRESS FOR RELOCATABLE PROGRAM SECTIONS

A␣ < address >

This control message enables the user to specify an absolute base address for relocatable program sections. < address > must be behind the monitor.
Relocatable addresses in code words are updated in accordance with the base address. As with the R message this base address is updated by addition of the length of the modules input to the Linkage Editor, each time such a module has been fully processed. The resulting program, whether output or loaded, will be in absolute format until an R message is given.

PROCESS INPUT FILE UP TO END OF FILE MARK

P

This message causes the processor to process the whole input file up to the EOF mark. All modules are link-edited or link-loaded. After this message IDENT < module name > is printed.

SELECT A SPECIFIED OBJECT MODULE IN THE INPUT FILE

S␣ < symbol >

This message causes the Linkage Editor to process only the object module with the name specified in this message, and which has been given in the IDENT directive previously. The user may specify more than one S control message.

SATISFY EXTERNAL REFERENCES FROM AN OBJECT MODULE LIBRARY

L

This message causes the Linkage Editor to process only those modules of the input file whose entry points match undefined external references. External references listed in a previous X control message, however, will remain unsatisfied. The selected module names are printed.
If a library is not in such an order as to enable all external references to be solved in one run, e.g. when the last module of the library contains an external reference to the first module, it is necessary to present it to the Linkage Editor a second time. The U control message can be used to determine whether this is necessary or not.

LIST THE NAMES OF ALL UNSATISFIED EXTERNAL REFERENCES

U

This message causes the Linkage Editor to list on the operator's typewriter the names of all undefined external references except for any which have been listed in an X control message. One external name is typed out per line.

PAUSE

H

This command forces the Linkage Editor to give a Pause monitor request to put it in pause state. During this pause any operator command may be typed in.
To restart the Linkage Editor give the command
RS⌈␣<input file code>⌉
where the Linkage Editor will then start processing from the input file code specified here. If it is not specified, it will restart on the old input file code.

TERMINATE PROCESSING

T

This message is mandatory. It terminates the processing. Optionally a listing is printed on the listing device.
When this control message has been given the Linkage Editor prints out the following three messages which provide the user with information on the length, the start address and end address of the load module.

L=**** S=**** E=****

where:

L= represents the length in characters of the relocatable program section. The length includes the length of the blank common allocated at the end of the relocatable program section of the generated load module. In link-load mode L=0000.

S= the relative or absolute start address (relative if the address is odd, absolute if even) of the generated load module.

E= in link-edit mode the highest absolute address in the absolute program section of the generated object program. E=0000 if the entire generated program is relocatable.
In link-load mode the first free location after the user's program section, commons included.

The Linkage Editor then outputs the END cluster *(in link-edit mode only)* of the generated program.

The output of the Linkage Editor depends on which option (link-edit or link-load) has been chosen in the option control message.

Link-edit
The output of a link-edit operation consists of:

— a load module or
— an object module.

Load module
This module contains no external references and may serve as input to the Linkage Editor or to the program loader.

Object module
This module still contains external references as during processing an E or X control message has been given. This module may only be input to the Linkage Editor.

Link-load
In a link-load operation no object output is produced as the program to be edited has been loaded directly into memory where an executable program is produced and where all relocations and linkages take place.

Information messages
Information messages are output by the Linkage Editor onto the operator's typewriter when a specific action is required or when the operator needs to be informed on the processing.
It is assumed that the option message has been input.

Message	Meaning
L:	Input a control message.
L?	The control message contained an error. Type in the correct message.
< symbol > ␣ < address >	The processing of an object module is started. < symbol > Ident. name of this module. < address > Load address of the module.
S = < address >	The relative or absolute start address of the generated load module. A relative address is indicated by an odd number of characters. An absolute address by an even number.
L = < hexa value >	This message gives the length of the relocatable program section. < hexa value > is an even number of characters. If it is 0000 the program is entirely absolute.
E = < address >	< address > is the highest absolute address in the generated module. It is 0 when the module is entirely relocatable. In link-load mode the address may include the length of the blank common, if any.

Error messages
All errors which have occurred during the input to the Linkage Editor are signalled by the printing of error messages. The processor does not check for errors in the output.
The error printed may be either a fatal error or a non-fatal error.

Fatal errors
One of the below messages is printed and processing stops.

Message	Meaning
C.ER	Error in a labeled common or error in base address of blank common.
CORE OVERFLOW	Not enough space for the user's program.
D.D.	Double Definition of entry points.
E C TYPE	Erroneous Cluster Type.
I/O ER	I/O error.
T.O.	Table Overflow.
XN	The external named in an X control message must remain unsatisfied.

Non-fatal errors
The error message is printed but processing continues.

Message	Meaning
ER.MOD	Erroneous input module.
IDENT TOO LONG	The IDENT name contains more than 6 characters.
BLK. DATA ER	An error was encountered in a Block-Data.

Map
The map, which is printed after a T control message, consists of a list of all entry points names. Its format is as follows:

< symbol > ␣[U|X|R|A␣ < address > |␣␣ < hexadecimal value >]

where:

< symbol > name of the entry point.
U undefined entry point.
X external name. < symbol > has been declared in an X control message.
R relocatable. In this case < address > is a relative address in the relocatable program section of the generated object program.
A absolute. In that case < hexadecimal value > is the absolute value equivalent to < symbol >.

Note: During the processing of the P, S and L commands the contents of the IDENT record, including comments, is output on the print file for each selected module.
The IDENT string itself is replaced by a hexadecimal value specifying the module displacement inside the generated program.

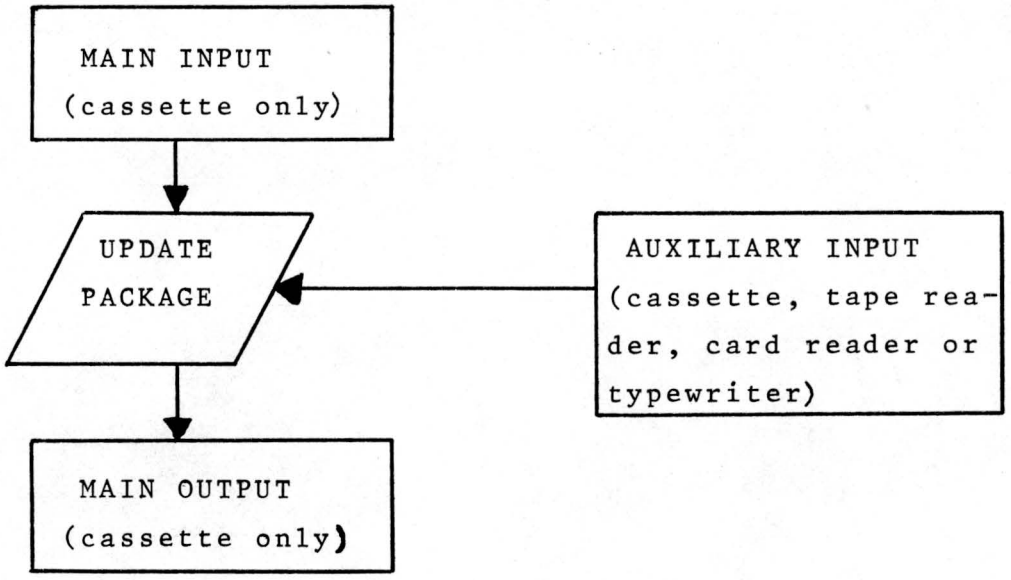
PART 5 UPDATE PACKAGE

The Cassette Update Package is a processor which allows the user to update his files and libraries.
The Update Package comprises the following functions:

- Copy
- Skip
- Delete
- Insert
- List

All these functions can be performed at different levels: for files, modules and lines.

The input and output devices on which these functions are performed, may be represented as follows:



Updating is done by means of a number of control commands which operate on different levels. These levels (file, module and line level) are structured as follows:

- a library consists of a number of files.
- a file consists of a number of blocks starting with a file header and ending with an end-of-file. It may consist of several modules.
- a module consists of a number of blocks starting with an IDENT block and ending with an end-of-segment. There are two types of modules:
  - source modules, consisting of source lines
  - object modules, consisting of clusters (see App. C).

At each of these levels a certain number of commands pertaining to that level can be used. Some of the commands on one level allow the user to switch to another level. The relationship between the commands and their levels is shown in a table at the beginning of the next chapter which describes the commands.

File Codes

The Update Package allows the use of six file codes:
input, output, auxiliary, print, punch and command input.
Two of these are required: input and output.
The file code values may be any value assigned to a device which is recognized by the Cassette Operating System. For each of the file codes, there are some restrictions as to the type of device to which it may be assigned:

- the input file code must be assigned to a compact-labelled cassette tape. This tape is the main input for the Update Package. If no other file codes are specified during a run (AS command), the Update Package will use file code 01 as standard.
- the output file code must be assigned to a compact-labelled cassette tape, except for some utility commands (see end of chapter 2 of this part)for which

it may be a tape punch. This tape is the main output of the Update Package which uses /06 as the standard file code for it.

- the auxiliary input file code is used for reading from an auxiliary input unit by the commands Insert and Skip on Auxiliary. The Update Package has no standard file code for this device.
- the print file code is used by the Update Package to list files modules and file headers. It will usually be the line printer, the Update Package using /02 as standard file code for it.
- the punch file code may be assigned to either cassette, magnetic or punched tape. It is used by the Update Package for the commands Punch File and Punch Module, with /03 as the standard file code.
- the command input file code is assigned to the device from which the control commands are read and onto which the error messages are output. The Update Package uses /05 as the standard file code for this device.

Operation

If the Update Package is contained on system tape, it is a very simple matter to load and start it:

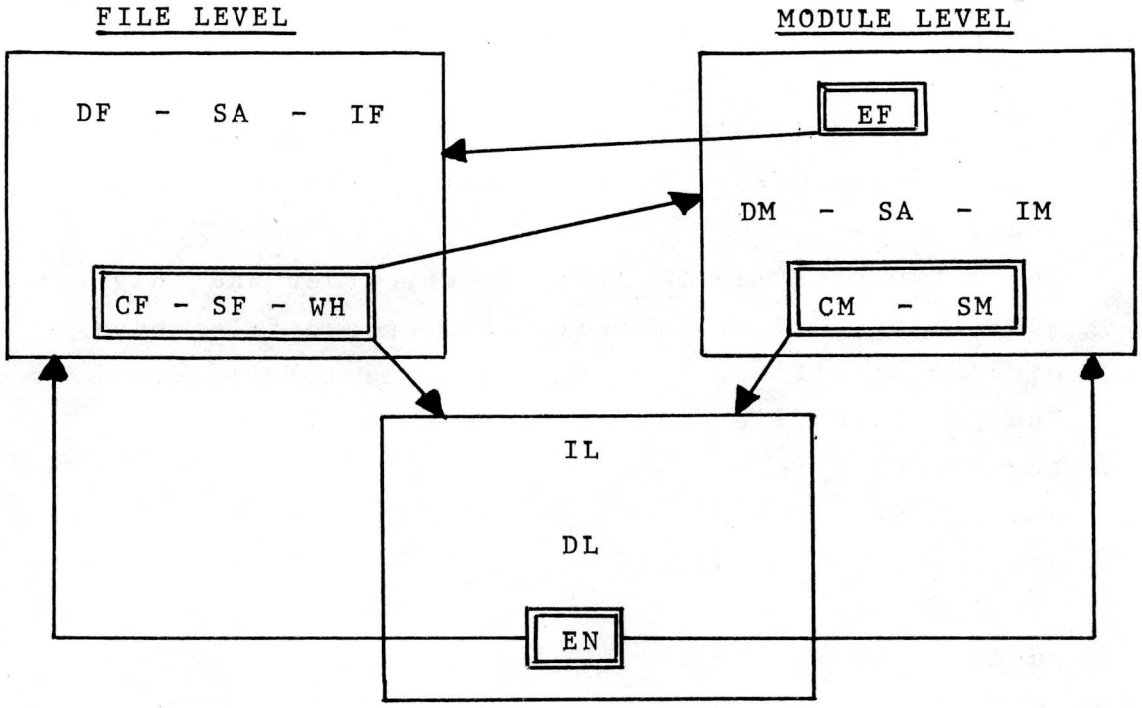
- put the cassette in the drive with file code /04 if has not already been put in
- push the INT button and on output of M: type in
  RN UPD
  and the monitor searches the header and loads and starts the Update Package, which types out
  CE:
  and waits for the user to type in his first command.

(If the update package is contained on an other medium: prepare that medium , load the Update Package by typing LD and start it by typing ST).

The available control commands can be used at different levels as illustrated in the follwing diagram. The arrows show from which set of commands it is possible to switch to another set of commands at another level, i.e. thus:

- from file level to module level: CF, SF, WH
  to line level: CF, SF, WH
- from module level to line level: CM, SM
  to file level: EF
- from line level to file or module level: EN.

Operation always starts at file level.



The control commands are normally entered via the typewriter, i.e. file code /05.

As with the other file codes, however, other assignments can be made by means of the AS command, which can be used at any level. If this command is not given, the standard file codes are used.

The notation for the syntax of the commands is Backus Normal Form, which is explained in Part 1, chapter 7.

Below, first the commands for file, module and line level are described, then the utility commands are given.

AS — Define File Codes — AS

Syntax: AS␣ I=<xx>,O=<xx>,A=<xx>,L=<xx>,P=<xx>,C=<xx>

where

<xx> is a two-digit hexadecimal number specifying a file code for:

I: input file code
O: output file code
A: auxiliary file code
L: listing file code
P: punch file code
C: command input file code

Use: This command is given when a file code different from the standard file code assignments must be used.

The parameters may be given in any order and only assignments different from the standard file codes need be specified. They may be separated by commas or blanks.

The standard file code assignments are:

input: 01
output: 06
auxiliary: no standard
listing: 02
punch: 03
command: 05

CF — Copy up to File — CF

Syntax: CF␣[[<file name>|:EOL]]

where

<file name> is the file header name preceding the file up to which a copy must be made.

Use: When this command is given a copy is made of one or more files (depending on how many precede <file name>) from the main input onto the main output. The copy

is made up to the file named <file name>, if specified, or, if :EOLis specified, of all the files from the current file to the end of library.
If no parameter is specified, only the current file is copied.
If the end of library is encountered before <file name>, an error message is printed, but all files from the current file to the end of library will have been copied.
From this command it is possible to proceed to module or line level.

SF — Skip to File — SF

Syntax: SF␣[<file name>]
where
<file name> is the file header name of the file up to which must be skipped.

Use: This command is given to position the main input tape at the beginning of the file specified (i.e. after its file header label). The positioning movement is always forward, so if the current position is past the file specified, an error message is printed.
If no parameter is specified, the current file is skipped.
If the file specified is not found, an error message is output and the main input tape will be positioned at the end of tape.
From this command it is possible to proceed to module or line level.

DF — Delete File — DF

Syntax: DF␣[<file name>]
where
<file name> is the file header name of the file to be deleted.

Use: This command is used to delete the file specified; this is done by skipping the file on the input tape and thus not copying it onto the output tape. All files from the current file up to <file name> will be copied.
If no parameter is specified, the current file is deleted, i.e. skipped and not copied.
If <file name> is not encountered, an error message will be printed and the main input tape will be positioned at the end of tape.

IF — Insert File — IF

Syntax: IF␣[<file name>|:EOL]

where

<file name> is the file header name of the last file which must be inserted.

Use: When this command is given file(s) are inserted from the auxiliary input device onto the output device.
If no parameter is specified, only the current file is inserted.
If <file name> is specified, all files on the auxiliary input up to and including <file name> are inserted.
If :EOL is specified, all files from the current file to the end of library are inserted.
If <file name> is not encountered an error message will be printed and the auxiliary input will be positioned at the end of library.

WH — Write Header — WH

Syntax: WH␣<name>

Use: When this command is given a file header label is written onto the output tape.
<name> is a string of up to 8 characters (alphanumeric) of which only the first 6 are taken into account by the system.
From this command it is possible to proceed to module or line level.

SA

Search Auxiliary File

SA

Syntax: SA␣⌈<file name>⌉

Use: This command has the same function as the Skip to File (SF) command, except that it is done on the auxiliary input unit and that the move may be forward as well as backward.

EF

End-Of-File

EF

Syntax: EF

Use: When this command is given an End-Of-File block is written onto the output tape.
It is used to terminate the creation of a set of modules (each ending with :EOS).
When this command is given it is possible to proceed from module level to file level.

CM

Copy up to Module

CM

Syntax: CM␣⌈⌈ module name>| :EOF⌉⌉
where
<module name> is the name (IDENT) of the module up to which a copy must be made.

Use: When this command is given, a copy is made of one or more modules (depending on how many precede <module name>) from the main input onto the main output. The copy is made up to the module named <module name>, if specified.
If :EOF is specified, all modules from the current module up to EOF will be copied.
If no parameter is specified, the current module is copied onto the output tape.
If an EOF block is found at the end of a module before an EOS block, Update will write EOF on the output tape. If an EOS is found before an EOF, Update will write an EOS and writing the EOF must always be done by giving the command EF.
If the output file had not been opened, the file header of the input file is copied onto the output file.

If <module name> is not encountered, an error message is printed, but all modules from the current module up to the end of file will be copied.
From this command it is possible to proceed to line level.

SM — Skip to Module — SM

Syntax: SM␣⌈<module name>⌉
where
<module name> is the name (IDENT) of the module up to which must be skipped.

Use: This command is given to position the main input tape at the beginning of a module. The positioning movement is always forward, so if the current position is past the module specified, the tape is read up to end of file and an error message is printed. This is also the case when <module name> does not exist.
If no parameter is specified, the current module is skipped.
From this command it is possible to proceed to line level.

DM — Delete Module — DM

Syntax: DM␣⌈<module name>⌉
where
<module name> is the name (IDENT) of the module to be deleted.

Use: This command is used to delete the module specified; if no parameter is specified, the current module is deleted.
Deletion is done by skipping the module on the input tape and thus not copying it onto the output tape.
All modules from the current module up to <module

name will be copied onto the output tape if a parameter is specified.
If module name is not found in the current file, an error message is printed and thr main input will be positioned at the end of the file.
If the output file had not been opened, the file header of the input file is copied onto the output file.

IM — Insert Module — IM

Syntax: IM␣[<module name>|:EOF]
where
<module name> is the name of the last module which must be inserted.

Use: When this command is given, module(s) are inserted from the auxiliary input device onto the output device.
If no parameter is specified, only the current module is inserted
If <module name> is specified, all modules on the auxiliary input up to and including <module name> are inserted.
If :EOF is specified, all modules from the current module up to the end of file are inserted.
If <module name> is not encountered an error message will be printed and the auxiliary input will be positioned at the end-of-file.
If an EOF block is found at the end of a module before an EOS block, Update will write EOF on the output tape. If an EOS is found before an EOF, Update will write an EOS and writing the EOF must always be done by givong the command EF.
If the output file had not been opened, the file header of the input file is copied onto the output file.

DL — Delete Line — DL

Syntax: !!DL␣<number1>[,<number2>]
where
<number1> and <number2> are line numbers in a program.

Use: This command is used to delete one or more lines.
If <number1> only is specified, the line with that number will be deleted.
If <number1>,<number2> is specified, the lines from <number1> to <number2> inclusive will be deleted.
<number2> must be larger than <number1>.
Any lines following this command on the input command file and not starting with !! will be inserted after the last deleted line.

IL — Insert Line — IL

Syntax: !!IL␣[<number>]
where
<number> is a line number in a program.

Use: This command is used to insert records.
If no parameter is specified, the inserion will be after the current line.
If <number> is specified, the insertion will be after the line specified.
Any records following this command will be inserted also, up to the next update control command starting with !!

!!EN — End of Line Modification — !!EN

Syntax: !!EN

Use: This command is used to terminate line updating.
Any remaining records on the input tape are copied onto the output tape.
!!EN must always be the last command on the line level.
From this command it is possible to return to either module or line level.

EN

Exit

E N

Syntax:: EN

Use: This command is used to return control to the monitor.

SH — Search Header — SH

Syntax: SH␣<header name>

Use: When this command is given, the file header specified is searched on the main input tape. The search may be forward or backward.
If the header is not found, an error message is printed.

LH — List Headers — LH

Syntax: LH␣[<header name>]

Use: When this command is given without parameter, all the file headers on the main input tape will be listed.
If a <header name> is specified, all the idents under this file header will be listed.
If the header is not found, an error message is printed.

LF — List File — LF

Syntax: LF␣[<file name>][, /IDENT]

Use: When this command is given without parameters, the current file on the main input tape is listed. If a <file name> is given, that file will be listed. If the specified <file name> is not found, an error message will be printed and the tape will be positioned at the end of the library.
If /IDENT is specified, all IDENT records are listed. This is used for listing an object library.

LM — List Module — LM

Syntax: LM␣[<module name>]

Use: When this command is given without parameter, the current module on the main input tape is listed.
If a <module name> is specified, that module will

If a <module name> is specified, that module will be listed.
If the specified <module name> is not found, an error message will be printed and the tape will be positioned at the end of file.

PF — Punch File — PF

Syntax: PF␣[<file name>]

Use: When this command is given without parameter, the current file is copied from the main input unit and punched on the punch unit.
If <file name> is specified, that file is punched.
The file header is never punched.
If <file name> is not found, an error message is printed and the tape will be positioned at the end of library.

PM — Punch Module — PM

Syntax: PM␣[<module name>]

Use: When this command is given without parameter, the current module is copied from the main input unit and punched on the punch unit.
If <module name> is specified, that module is punched.
If <module name> is not found, an error message is printed and the tape will be positioned at the end of file.

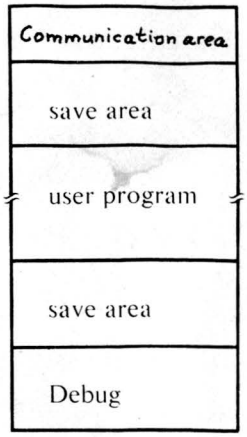
ERROR MESSAGES

In case of an error, one of the following error messages is printed, after which CE: is output and Update waits for the user to give a correct command:

UNKNOWN COMMAND
PARAM ERROR
TOO MANY PARAM
INVALID LEVEL
INPUT I/O ERROR
OUTPUT I/O ERROR
UNKNOWN PARAM
UNAVAILABLE CMD (CMD = command)
END OF FILE SET
NO AUXILIARY F.C. (F.C. = file code)
PARAM MISSING
LABEL. TYPE ERROR (LABEL. = labelling)
INVALID IDENT
INPUT OVERFLOW
I/O ERROR

PART 6 DEBUGGING PACKAGE

The Debugging Package is used to check and test programs written in Assembly Language.
After the user's source program has been assembled the resulting object program must be processed by the Linkage Editor. The Load module must be loaded prior to the Debugging Package. The latter must be loaded with a displacement value equal to the user program length + /20 (= save area).



Once the Debugging Package is loaded and started (LD and ST commands) the user program's absolute load address is printed and the Debug message D: to allow commands to be input from the operator's typewriter. Once the Package is called the user may set breakpoints where the Package will suspend execution of the user program to be tested and await further commands. This allows the user to check his programs section by section. He may examine the contents of any memory location and modify those lying within the program's boundaries or he may test the contents of register A1 thru A14 before starting the execution or during breakpoints halts. The Debugging Package is not reentrant, therefore it is not possible to test a main program and its scheduled labels simultaneously. It is allowed to define breakpoints in scheduled labels.

User program memory locations may be addressed in the following two ways:
– absolute
– relative

A location is addressed **absolute** as follows:
1) take the absolute load address printed on the typewriter log after having called the Debugging Package.
2) add the relative address of the location as printed on the Linkage Edit map

The **relative** address of a location is found by taking the relative address of the symbol as printed on the map.

On-line/Off-line

The Debugging Package allows on-line as well as off-line operation:

On-line: a Debug command is executed immediately after having terminated with LF CR. All commands in chapter 2, except the IF command, are accepted in this mode which begins immediately after having called the Package or when a breakpoint terminated by a RT command has been executed.

Off-line: mode which is entered when a breakpoint is defined. The user may now type one command or a number of commands which will be executed when the user program reaches the specified breakpoint. The string of commands pertaining to a breakpoint must be terminated by eiter a GO or a RT command. The user program's execution is suspended during processing of the commands.

Input/Output

Input
Input to the Debugging Package consists of either commands or of data. Commands are usually input from the operator's typewriter when the Package requests input after having typed D: on the typewriter log.
One command, CI, permits to read commands from either the card reader or the punched tape reader.
The command RE allows to read data from a specified device. This is particularly useful when the contents of buffers have to be changed or filled. The file code of the respective devices must have been assigned before calling the Package.
If commands are read from a device other than the typewriter and the input command is erroneous, this command with error indication and error message is printed on the typewriter log. The correct command may be typed in from the operator's typewriter after D: has been printed on the log.
The following commands are read again from the assigned input device.

Output
The output is normally directed to and printed on the typewriter log.
However, the output may be directed to another output devic by means of the CØ command provided the file code was correctly assigned before calling the Debugging Package.
Commands typed in from the typewriter are copied on the line printer.

Note: If the program to be debugged is aborted the Debugging Package keeps control. The Program Status Word is printed on the typewriter log and the line printer, followed by the relative abort address and the print-out of the contents of 14 registers. On the line printer, or on the typewriter log if the output was directed to TY, a memory dump takes place of the relating area.

A command consists of two ASCII characters which may be followed by a blank and one or more parameters. Parameters are separated by a comma. If a command is terminated by a full stop another command (or commands) may follow on the same line but may not continue on the next line. Each command, or the last command of a line if more commands are specified on one line, must be terminated with LF CR.

Parameter syntax:

<memory reference> ::=	<absolute address> <relative address>

where:

<absolute address> ::=	/ <up to 4 hexa digits> In IF command: M␣ <hexa number>
<relative address>	@ <up to 4 hexa digits>
<register>	R <two digit decimal number>
<constant>	/ <up to 4 digit hexa number>

BREAKPOINT DEFINITION

syntax: AT␣ <memory reference>

The breakpoint facility provides a means of suspending the execution of a user program at any point. To set a breakpoint the user types AT followed by the absolute, or relative address of the word where he wants the program to stop.
Once the breakpoint is set it switches the Package to the off-line mode thus permitting to define a command or string of commands which are to be executed when the running user program reaches the location specified in the breakpoint.
The breakpoint's absolute address is printed when it is executed: BP: absolute address.
Breakpoints are kept in a table. The maximum number of breakpoints allowed in this table is 8. Once a breakpoint is executed or when the user does not consider a breakpoint necessary anymore it may be deleted from this table by the command DB.
This permits to use more than 8 breakpoints in a debug run though only 8 breakpoints can be present in the table at the same time.
Addresses specified in breakpoint definitions need not to be in an ascending order, so the user may first define an address at the end of the program and then one at the beginning if he wishes to do so.

During a debug run a breakpoint may be defined only once, unless the breakpoint is deleted.
When a breakpoint is reached the instruction to which it points is executed.
The last command of a string of commands pertaining to a breakpoint must be either GO or RT, concluding this breakpoint definition.

Restrictions

The breakpoints defined may not:
- be modified by the program
- refer to DATA defined text (the BP is not executed)
- refer to a LKM (or MLK) instruction
- refer to an address defined in an EX, EXK or EXR instruction

DELETE A BREAKPOINT

syntax: DB␣ < memory reference >

This command is used to delete a breakpoint and the commands defined to be executed at this breakpoint.
A breakpoint can be deleted from the breakpoint definition table at any time except when it is being executed.

```
D:AT /89A8

D:DR R1

D:GO

D:AT /8A4C

D:DR R2

D:GO

D:DB /89A8

D://
1210

1283
BP: 8A4C     This is 2nd breakpoint. The 1st
A2 =3833     one is deleted
```

Example 6.1

DUMP MEMORY

syntax: DM␣ < memory ref 1 >, < memory ref 2 >

Through this command the user may examine the content of a memory area from and including < memory ref 1 > thru < memory ref 2 >.
The dump takes place on the operator's typewriter unless otherwise specified by the CO command beforehand.
The dump is presented as 8 words per line. Each line is preceded by an absolute address (multiple of /10). The last line of the dump is filled up to the eighth word of that line, with words immediately following < memory ref 2 >.

```
D:DM /6511,/6528
6510  7D84  F4A5  8B22  F6DE  2020  5245  5345  5256      "    RESERV
6520  4544  2020  4E55  4D42  4552  2055  4E4B  4E4F  ED  NUMBER UNKNO
```

Example 6.2

WRITE MEMORY

syntax: WM␣ <memory ref>,<constant 1>[,<constant 2>,.....,
<constant n>]

This command permits the user to substitute the content of a memory location by an other value or, if more constants are specified, as many memory locations, from <memory ref> on, as there are constants specified.
The locations' contents must be within the program boundaries.

```
D:DM /6D08,/6D12.WM /6D08,/1234,/5678.DM /6D08,/6D0A
6D00  444F  4320  5245  5620  5041  4745  2033  3520  DOC REV PAGE 35
6D10  0D0A  3131  3632  3320  5038  3535  4D20  5359    11623 P855M SY
6D00  444F  4320  5245  5620  1234  5678  2033  3520  DOC REV  4V  35
```

Example 6.3

DUMP REGISTER

syntax: DR␣ <register>[,<register n>]

This command dumps, in hexadecimal format, the contents from 1 to n registers. <register> may be any of the user registers A1 thru A14 (see parameter syntax).
If no parameters are specified the contents of all registers, except for the P and A15 registers, are dumped.
If only one parameter is specified the contents of that register is dumped.
If both parameters are present the contents of those registers, both specified included, are dumped.
The dump is given on the typewriter log unless otherwise specified in the CO command.

```
D:DR R1
A1 =0000

D:DR R1,R5
A1 =0000  A2 =0000  A3 =0000  A4 =0000  A5 =0000

D:DR R15
        ↑
PARAMETER ERROR
```

Example 6.4

WRITE REGISTER

syntax: WR␣<register>,<constant 1>[,<constant 2>.....,
<constant n>]

The content of the specified register will be changed to the value of the first constant or, if more constants are specified, a number of consecutive registers is loaded with the values of the same number of consecutive constants. The first register of the range is the register specified in this command.

```
D:DR R1,R4
A1 =0000  A2 =0000  A3 =0000  A4 =0000

D:WR R1,/0011,/0022

D:DR R1,R3
A1 =0011  A2 =0022  A3 =0000

D:WR R2,/0033,/0044

D:DR R1,R4
A1 =0011  A2 =0033  A3 =0044  A4 =0000
```

Example 6.5

```
D:AT /88D6

D:DR R1,R4

D:WR R1,/0066,/0077,/0012

D:DR R1,R4

D:RT

D://
1210

BP: 88D6
A1 =3132  A2 =3130  A3 =3132  A4 =3130
A1 =0066  A2 =0077  A3 =0012  A4 =3130
```

Example 6.6

CHANGE INPUT DEVICE

syntax: CI␣/ <file code>

When this command is given all further commands are read from the device with the specified file code.
Input devices other than the teletype may be the card reader or the punched tape reader. Their file codes must be assigned before the Package is called. In order to return the input of commands to the operator's typewriter the command CI␣/5 must be given.

```
D:AT /89FE

D:CI /1

D:RT

D://

1383

BP: 89FE
A1 =3133  A2 =3833  A3 =3133  A4 =3130  A5 =0000
A1 =3133  A2 =F1F1  A3 =1234  A4 =2345  A5 =6161
```

```
DR R1,R5
WR R2,/F1F1,/1234,/2345,/6161
DR R1,R5
CI /5

This was punched on tape
```

Example 6.7

CHANGE OUTPUT DEVICE

syntax: CO␣/ <file code>

This command directs the output, from the time the command is read, to the output device with the specified file code.
This file code must have been assigned before the Package is called.
If output is to be returned to the operator's typewriter the command CO must be given with the typewriter file code.
Whether the command is given on-line or off-line does not affect its operation i.e. the output device remains the specified output device until a new CO command is given.

RETURN TO ON-LINE MODE

syntax: RT

This command concludes a breakpoint definition.
When the user program runs it will stop at the address defined by the breakpoint of which RT is the termination. The Debugging Package resumes control and types out D: on the typewriter log thus switching to on-line mode.
The user may now type in new commands. In this way it is possible to react immediately on the results of a breakpoint execution (see example 6.8).

CONTINUE EXECUTION

syntax: GO␣[< memory reference >]

This command concludes, as RT, a breakpoint definition. The difference between both commands is clear when the user program is executed and the breakpoint belonging to GO is reached.
The breakpoint's absolute address is printed or punched and commands belonging to the breakpoint are executed. When the GO command is read control is not returned to the operator's typewriter, as with RT. The execution of the user program continues until a new breakpoint is encountered.
In this case the user has not the possibility to react immediately on a breakpoint's execution results.
If < memory reference > is specified the user program will continue at the specified address which must be within the program's boundaries.
If < memory reference > is not specified the execution resumes at the address following the breakpoint.
In example 6.8 two breakpoints are specified. One terminated by GO and the other one by RT. The first breakpoint is printed on the typewriter log as the CO command was not yet read. All other output is printed on the line printer log. After RT the user may input another command.

```
D:AT /6CE4

D:CO /2

D:DM /651A,/6526

D:GO

D:AT /6DF1

D:DM /651A,/6526

D:RT

D://
1688

BP: 6CE4

NUMBER UNKNOWN
1835

D:RX

S:
```

Example 6.8

CONDITIONAL EXECUTION

syntax: IF␣[<memory ref>|<register]>|=|<[<memory ref>|<register>|<constant>]

This command may only be used after an AT command. It allows a conditional execution of the command string attached to this breakpoint.
The content of an address or register is compared with the content of another address, register or constant.
If the condition specified is true the command string is executed. If not the user program is restarted (implicit GO command).
If <memory reference> is an absolute address then <memory reference> must be specified as M␣<hexa number>.

TRACE

syntax: TR␣<2 ASCII char>

This command can be used to check a condition after a branch instruction which may cause the user program to enter a loop.
The two ASCII characters specified are printed out each time the program reads the breakpoint to which this command is attached.
Restriction:
The ASCII characters may not be
— a space
— a full stop

```
D:AT /6CA8

D:TR LO

D:GO
D://
BP 6CA8
LO
BP: 6CA8
LO
BP: 6CA8
LO
```

Example 6.9

READ FROM EXTERNAL DEVICE

syntax: RE␣/<file code>,<memory reference>,/<no of characters>

Through this command a number of characters may be read in a buffer whose first address is <memory reference>. The data is read from the device with the specified file code.
When this command is given a standard read is sent to the monitor with the specified address and number of characters.
The number of characters mus be specified hexadecimally.
The message READ is printed on the typewriter log when the user requests a record to be written from the typewriter.

START USER PROGRAM

syntax: //

This command must be used to start the execution of the user program after having defined one or more breakpoints. The user program runs until a breakpoint is encountered and commands defined at that breakpoint are then executed by the Debugging Package.
This command has the same function as GO <start address> when the latter command is used on-line.

EXIT

syntax: RX

This command causes an exit from the Debugging Package and switches control back to the Monitor.

The following messages are output when an illegal condition occurs.
The messages are always printed on the typewriter log.

MESSAGE	MEANING
UNKNOWN BP	— the Package is asked to delete a breakpoint which has already been deleted
	— the DB command contains an incorrect address.
BP DOUBLE DEFINED	— the breakpoint with the specified address already exists in the BP table.
REFUSED IN ON-LINE MODE	— the IF command is given not immediately following an AT command.
REFUSED IN OFF-LINE MODE	— an attempt is made to define a new breakpoint without having terminated the previous one by a GO or RT command.
BP CANNOT BE DELETED	— the current breakpoint cannot be deleted
BP TABLE OVERFLOW	— the BP table may not contain more than 8 breakpoints. Delete from the table some breakpoints already executed or considered no longer necessary.
NO BP ON LKM/MLK	— the breakpoint may not point to a LKM or MLK instruction.
PARAMETER ERROR	— this message is printed when an illegal parameter is specified
SYNTAX ERROR	— the syntax in the command is erroneous
FILE CODE NOT ASSIGNED	— the file code specified was not assigned before the Debugging Package was called.
COMMAND UNKNOWN	— the command given does not belong to the list of commands discussed in the previous chapter.
SYMBOLIC REF. ERROR	— symbolic references are not accepted in the Basic Debugging Package
NO START ADDRESS	— start address missing in the module to be debugged
COMMAND TABLE OVERFLOW	— not enough room to record the command string in the command table
INVALID ADDRESS	— relative address not within program limits

PART 7 CASSETTE FULL FORTRAN COMPILER

The Cassette Full FORTRAN compiler accepts Full FORTRAN source modules as input, and produces object modules to be processed by the Linkage Editor with the Full FORTRAN system library. The result is a self contained executable FORTRAN program which can run under control of the Cassette Operating Monitor. The compiler is self-initializing, and does not require re-loading between succesive compilations.

Source language

The source language for the Cassette Full FORTRAN compiler is identical to the one described in the P800M Full FORTRAN Reference Manual (publication number 5122 991 1140X), except that direct access I/O statements are not accepted.

Operating procedures

The compiler is loaded into memory and started by means of the cassette control command RN (Run a program).
Once the compiler is started, it outputs

F:

which requests the I/O options to be specified.
If the standard I/O options are to be used the operator replies by typing

LF CR

(standard I/O depends upon which devices have been assigned the standard file codes 1,2 and 3 which specify the source input device, the listing output device and the object code output device respectively).

If other options are selected the operator must specify the file codes for the source input file, the listing output file and the object file, plus, if required, the options Q and/or N, followed by LF CR. Q specifies object code output in 4x4 format, N indicates that the output listing is to be suppressed; this applies to all source modules being processed

and has absolute priority over any OPTIONS statement in the source program.Error messages will always be listed.

If object code output is not required, the code to suppress it is 0, If the options are incorrectly typed, the compiler outputs a further

F:

and the options can be retyped, ending with LF CR.

The compiler then begins reading and processing the source program, the first line of which must be an IDENT control statement. If the IDENT is missing the compiler will continue reading the source file, without processing, until the IDENT is found.

When an END statement is processed which is not followed by :EOS -indicating further modules are to be compiled- the compiler writes :EOS on the object code output file, and if no errors have been found in the END statement, outputs the message

F/MESSAGE

on the listing device and automatically continues reading a new source module (which must begin with an IDENT statement) without any operator action being necessary.
If there is not another module, the next statement read must be :EOF. When an :EOF is read in the input stream, the compiler checks that the last statement was an END statement, if it was not

F/MESSAGE EM

is printed on the listing device and an END statement is automatically generated and compiled. The compiler then types out

F:7F00

indicating succesfull compilation.

A fresh compilation can then be started, or if no further processing is required, the operator types

```
X
```

upon which the compiler will exit, or

```
:EOF
```

to make the compiler write :EOF on the object code output file and exit.

PART 8 CASSETTE FULL FORTRAN TRANSCODER

The Cassette Full FORTRAN Transcoder is a one-pass processor that translates modules compiled by the Cassette Full FORTRAN Compiler from interpretive code, which is only executable through an object code interpreter, into direct executable machine code. The modules produced by the transcoder are ready for processing by the Linkage Editor.
Transcoded and compiled modules can be used together in one FORTRAN program, if necessary the object code interpreter will be added by the Linkage Editor.

Operating procedures

Input for the transcoder is the object code produced by the Full FORTRAN compiler. The transcoder is loaded and started by means of the cassette control command RN. Once started it outputs

T:

and waits for a control command.
The possible control commands are:

C	copy the current module untranscoded onto the output file
D	delete the current module
E	end the transcoding session
:EOF	write :EOF on the output file and end the transcoding session
T [,NL] [,NP]	transcode the current module NL and NP are optional parameters. If NL is specified, no listing of the transcoded module will be produced (control messages and error messages are always listed on the operator's typewriter), if NP is specified, no object code will be produced.

Each command must be terminated by CR LF.

Error messages

IDT.MIS.	IDENT statement missing or invalid
END.MIS.	end/start cluster missing
CLS.ERR.	erroneous cluster type
COM.MOD.	the current module is not a compiled one
BLK.DAT.	block data modules cannot be transcoded
TBL.OVF.	table overflow
MOD.ERR.	unsuccessfully compiled module
DYN.ALL.	dynamic allocation forbidden
IO ERROR xxxx yyyy	unrecoverable I/O error,xxxx=file code, yyyy= returned status
OBJECT CODE NOT USABLE	because of a fatal error the transcoded object code is not usable

If an error is detected, the transcoder ends the current session. The transcoder can be restarted by depressing the INT-button on the control panel, and typing ST after the monitor has requested a command by the message M:.

Control messages

IDENT <name>	idnetification of the current module
:EOS	end of segment mark read
:EOF	end of file mark read
T:	a control command is requested
COMPILED LENGTH=xxxx	TRANSCODED LENGTH =yyyy length of the compiled and the transcoded module, expressed as a hexadecimal number of characters

Listing

The object code listing consists of:

- the module identification
- the names and numbers of the external references
- the produced instructions

Each transcoded statement is delimited by two lines of asterisks. Flag R indicates 'relocatable', flag X indicates 'external reference'. Two consecutive lines of asterisks indicate an empty area, or an area that is to be filled later on.
An example of a listing of a simple FORTRAN program and the instructions produced by the transcoder is given on the next page.

```
    IDENT PROG
*****************************
$PROG       0002
%PROG       0004
/PROG       0006
F:ER11      0008
F:ER10      000A
F:ER04      000C
F:ER03      000E
F:ST        0010
:EOS
*****************************
*****************************
    0008  2020
    000A  2020
    000C  2F45
    000E  4E44
*****************************
    0010  0001
    0012  0004
*****************************
    0014  86A0
    0016  0002    X  0002
*****************************
    0018  B8C0
    001A  0010    R
    001C  B8C1
    001E  0000    R
*****************************
    0020  B8C0
    0022  0012    R
    0024  B8C1
    0026  0002    R
*****************************
    0028  B8C0
    002A  0000    R
    002C  9140
    002E  0002    R
    0030  8320
    0032  0000    X  000A
    0034  B8C1
    0036  0004    R
*****************************
    0038  8420
    003A  0000    X  0004
    003C  F6A1
    003E  0000    X  0010
*****************************
    0040  8420
    0042  0002    X  0004
    0044  F6A1
    0046  0000    X  0010
*****************************
*****************************
    0048  0008    R
    004A  000C    R
*****************************

COMPILED LENGTH = 0078     TRANSCODED LENGTH = 0052
```

```
IDENT PROG
K=1
L=4
M=K+L
STOP
END
```

PART 9 UTILITY PROGRAMS

The P852M computer permits the user to load and run an Initial Program Loader by simply pressing the IPL button on the front panel.
When doing so the contents of a 64-word Read Only Memory (ROM), which has been preprogrammed with a standard bootstrap, is read into the CPU's memory where it is executed.
Next the IPL, which may be located in front of a program or which may be an independent tape or may be written on cassette tape, magnetic tape or on disc, is automatically loaded and run from one of the following devices:
- operator's typewriter (4×4)
- punched tape reader
- fixed head disc
- moving head disc
- magnetic tape
- cassette tape.

However, before the button is pressed the user must set the following parameters on the data switches:

bit 0 = 1 IPL is to be loaded from ASR (4×4)
0 IPL is to be loaded from other device

bit 1 = 1 IPL from disc
0 IPL from other device

bit 2 = 1 moving head disc
0 fixed head disc

bit 3 = 1 programmed channel transfer
0 I/O Processor transfer

bit 4-7 control information for control unit during execution of CIO Start sent by the bootstrap. Specify the type of device:

TY =	0001	X1215	= 0011 (physical sector address)
TK =	0111	FHD	= 0011 (physical sector address)
MT =	0010		

bit 8 = 1 multiple device control unit
0 single device control unit

bit 9 = 1 X1215 disc (P824-001 CDD disc)

bit 10-15 device address of device from which the IPL is loaded

When the IPL has been read it prints on the operator's typewriter:

OBJCT TAPE ON READER. THINK OF BASE!

The user has now the opportunity to modify the program's base address. This address is displayed in register A9. If the address is not modified the base address is assumed to be 0.
The CPU may be restarted by pressing the RUN button.
If a start address has been defined for the loaded program at assembly time control is given to this address by the IPL.
When the loading of the IPL and program is completed register A9 contains the first free location behind the loaded program.
The loading of an absolute program section does not change the contents of A9.
The bootstrap area may be overwritten by the loaded program.

Loading procedure

- prepare device from which IPL is to be read and place IPL on this device.
- set the relevant information on the data switches.
- push IPL button. The IPL is now read.
- if the IPL is not located in front of the program to be loaded and on the same medium, place the program to be loaded on the input device.
- push RUN button to read the program.

Note: By means of the IPLGEN program, which is part of the system generation procedure and is contained on one of the generation cassettes, the user may generate IPL programs. The Initial Program Loader thus generated can be used for any device.

The ASCII dump program allows to have an ASCII memory dump in hexadecimal format on the line printer or on the operator's typewriter. The program is delivered on paper tape in 8+8 or 4×4 format. A separate IPL must be used to load the dump program. (IPL generation: see App.A: SysGen.)
The IPL is loaded by pushing the IPL button on the control panel. The user may now specify in register A9 the loading address of the dump program. Default value = 0000.
Next push the RUN button to load the dump program. When the reading stops the user must load register A8, A9 and A10 with the following information:

A8 to be loaded with the address of the device onto which the dump will take place, e.g. /10 for the operator's typewriter or /07 for the line printer.
If the device is connected via the programmed channel bit 0 of register A8 must be set to 1.

A9 to be loaded with the first address of the area to be dumped.

A10 to be loaded with the last address of this area.

Press the RUN button to activate the dumping.

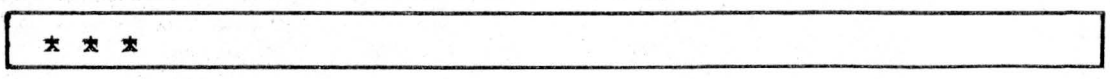
The cassette premark program PREMTK must be used for cassette tapes which are going to be used under the Cassette File Management Package. PREMTK is a program which writes information at the beginning of the tape, so as to make it recognizable to the CFM package as a tape using one of the three ECMA standards of labelling: Basic, Compact or Extended. For details on these labelling systems, see Part 1, Chapter6. The program is loaded from cassette by giving an LD command followed by an ST command and uses the following file codes:

- file code 0A is the command input file. This must be the operator's typewriter, through which the PREMTK prints out the questions to which the user must type in the answers. If this file code is assigned to another device, PREMTK does not output the questions, but only reads the answers. These must then be input in the correct order. See the procedure description below.
- file code 03 is the output file. This must be a TK cassette drive. On this file code PREMTK outputs the premark block.
- file code 02 is the list file. On this file PREMTK lists questions and answers during the program run. Each time am answer is accepted, PREMTK outputs question and answer on the listing.

Basic Labelling

For this type of labelling, a cassette is premarked as follows:

- On side A PREMTK writes one tape mark for intermediate start of track and two tape marks for End of Data, as follows:



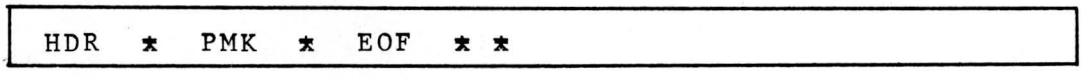
- Side B is premarked in exactly the same manner.

Compact Labelling

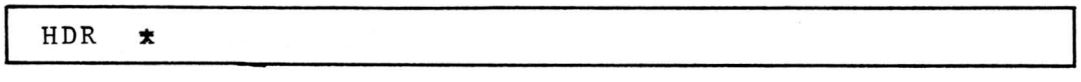
For this type of labelling, a cassette is premarked as follows:

- Side A:
  First PREMTK writes a File Header Label (HDR) at the beginning of the tape, with the following information:
  - label identifier
  - volume identifier
  - file identifier
  - creation date

  All the other fields are filled with zeros (= default value). The File Header Label is followed by one tape mark, followed by a PMK block, which has the same layout as the File Header Label, except that the label identifier has the value /3F (= ?) . By means of this block the CFM package recognizes the tape as premarked.
  The PMK block is followed by one tape mark, an EOF label (for compact labelling) and two tape marks, so that side A of a premarked tape will look as follows:



- Side B:
  PREMTK writes a File Header label with the same layout and data as for the HDR on side A, followed by a tape mark. The CFM package requires this File Header label on side B to recognize the same volume in a multi-track write process. Side B will thus look as follows:



Extended Labelling

For this type of labelling, a cassette will be premarked as follows:

- Side A:
  First PREMTK writes a Volume Header Label (VOL) at the beginning of the tape, containing the following information:

- label identifier
- label number
- volume identifier

All the other fields are filled with zeros (= default value).
The Volume Header Label is followed by a File Header Label containing the following information:

- label identifier
- label number
- file identifier
- creation date

All the other fields are filled with zeros (= default value).
The File Header Label is followed by one tape mark, a PMK block (with the same layout as for Compact Labelling), one tape mark, an EOF label and two tape marks, so that a tape premarked for extended labelling will look as follows:

```
VOL   HDR * PMK * EOF **
```

- Side B:
  PREMTK writes a Start of Track Label (STR, see Part 1, Chapter 6), followed by the same File Header Label as on side A, followed by a tape mark, so that side B of the cassette will look as follows:

```
STR  HDR *
```

Operation

When PREMTK has been loaded and started, it prints a number of questions on the typewriter, to which the user must type the answers. Each answer must be followed by (LF) (CR).
After the first question, the procedure is different for Basic and for Compact/Extended Labelling.

The first question printed by PREMTK is:

TYPE OF LABELLING

Reply: ⌊B|C|E⌋
for Basic, Compact or Extended respectively.

When the reply for this question was B, PREMTK continues:

CHANGE TRACK AND #RS

The user must take out the cassette and put it back in with side B up. When the tape has been rewound, he must restart PREMTK by pushing the INT button and, on output of M:, typing in RS. PREMTK then prints:

RUN AGAIN?

Reply: ⌊NO | OK⌋
If the reply was NO, PREMTK exits.
If the reply was OK, the procedure is repeated.

When the reply for the first question was C or E, PREMTK continues:

VOLUME IDENTIFIER

Reply: <volume>
where <volume> is a string of 4 characters for compact labelling or a string of 6 characters for extended labelling (maximum).
When the character string is SYST, this means that this tape is being premarked to receive IPL + monitor and become the system tape. In this way IPL + monitor can be recorded on this tape with a Copy File command of the Update Package. When SYST is specified, the two following questions are skipped.

FILE IDENTIFIER

Reply: <file ident>

where <file ident> is a string of max. 8 characters for compact labelling or a string of max. 17 characters for extended labelling. A space is not allowed as the first character.
For compact labelling, if an asterisk is typed (*), the cassette will be declared as a working cassette (see Part 1, chapter 10). In this case, the following question is skipped.

CREATION DATE

Reply: yyddd
where yy is the year, and ddd is the day of the year.

CHANGE TRACK AND #RS

The user must take out the cassette and put it back in with side B up. When the tape has been rewound, he must restart PREMTK by pushing the INT button and, on output of M:, typing in RS. PREMTK then prints:

RUN AGAIN?

Reply: [NO | OK]
If the reply was NO, PREMTK exits.
If the reply was OK, the procedure is repeated.

Error Messages

If an erroneous answer is given, PREMTK repeats the question.

PREMTK may output one error message:
OUTPUT FILE 03 NOT ASSIGNED TO "TK" - PAUSE

Control Command

Syntax: #

Use: This command can be used at any time during a premark run to exit. When this command is given,

PREMTK prints out the message OPERA. ABORT
EXIT
and exits. To restart, the user must press the INT button and type in the command ST.

EXAMPLE

```
M:LD
  IDEN
PREMTK              5111 100 23571 COS BASIC PACK. CAS. 2/2  44CC
:EOS  4F8E
:EOF
M:ST

-  *** PREMARK CASETTE ***
   TYPE OF LABELING:  C

   VOLUME IDENTIFIER:WORK

   FILE IDENTIFIER:*
   CHANGE TRACK AND #RS

   M:RS

   RUN AGAIN?  OK

-  *** PREMARK CASETTE ***
   TYPE OF LABELING:  C

   VOLUME IDENTIFIER:MWSC

   FILE IDENTIFIER:AAP

   CREATION DATE:76055
   CHANGE TRACK AND #RS

   M:RS


   RUN AGAIN?  NO

   EXIT
```

APPENDICES

The system generation process, which consists of three main steps, i.e. monitor tables generation, monitor body generation and system medium generation are handled by a number of generation processors which are loaded from cassette and started successively.
The system generation procedure for the Cassette Operating System provides great flexibility in the number of possibilities allowed to the user and extensive logging of the process.

The following processors are used to generate a Cassette Operating System:

- GENMON, a generation monitor used only during the system generation process to run the generation processors.
- COMGEN, which generates the monitor tables from the answers it receives from the user in a conversational process with a standard list of questions.
- IPLGEN, which generates the Initial Program Loader (IPL), the first module to be stored on the system medium. IPLGEN runs under any monitor.
- GENLKE, which runs under GENMON and scans the library of system modules, to select the ones requested during the COMGEN phase and link them with the monitor tables generated during COMGEN.
- CASLOD, which, running under GENMON, is used to record the system processors on cassette tape in the Compact system of Labelling.

In the following paragraphs, the whole set of operations necessary for generating a Cassette Operating System is described in a number of sections corresponding to the sysgen processors listed above. At the end of each section a number of notes and remarks may be given, which the user must carefully read before starting the operation.
At the end of this Appendix, a sysgen example is given, followed by error recovery procedures for errors made during the GENLKE or CASLOD phases.

OPERATION

The minimum configuration required for generating a Cassette Operating System is:
- CPU with 16k memory
- typewriter with address 10
- two magnetic tape cassette drives

The user receives two so-called generation cassettes, containing:
- generation cassette 1 (G1):
  - side A: IPL
    GENMON
    COMGEN
    IPLGEN
    GENLKE
    (these are all programs in load module format)
  - side B: COMLIB (object library)
    CASLOD (a program in load module format)
- generation cassette 2 (G2):
  - side A: PREMTK (cassette premark)
    UPD (cassette update package)
    ASM (assembler)
    LKEGEN (linkage editor)
    LKEGEN
    CASLOD
    DEBUG
    (these are in load module format)
  - side B: empty

The user needs two cassettes of his own:
- one system cassette, referred to in the rest of this chapter as cassette S
- one working cassette, for temporary storage, referred to

as cassette W.

It may be that the user wants to generate his system on a medium other than cassette magnetic tape. Definition of the system medium is established during the GENMON phase. In that case the CASLOD processor is not required, because this processor is used exclusively to record on cassette tape in Compact Labelling.

The system generation process is based, however on generation from cassette tape onto cassette tape and therefore a number of standards have been incorporated in the process. If these are not acceptable to the user, he must alter them during the GENMON phase.

To start the generation process:

- switch on the CPU
- load the generation cassette G1, with side A up, in cassette drive TK05
- load cassette W (working cassette) in cassette drive TK15; if cassette tape is not going to be the system medium, prepare the paper tape punch or magnetic tape drive, whichever medium is chosen under GENMON
- set the data switches on the CPU control panel to allow the bootstrap to load from TK05:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	1

(hexa 0785)

(for the significance of the bits, see Part 1, Chapter 10 ; suffice it to mention here that bit 3 is 0 if the cassette drives are connected to the I/O processor and 1 if they are connected to the programmed channel, and bits 10 to 15 contain the device address).

- press the MC button
- press the IPL button

Now the bootstrap is loaded which loads the first processor from the cassette in TK05 into memory:

GENMON

When GENMON is loaded its identification is output on the typewriter:
GENMON
When loading is terminated,
:EOS
:EOF
is output on the typewriter, followed by the question
STANDARD CONFIGURATION?
The reply to this question can be Y[ES] or N[O].
If the user replies Y followed by (LF) (CR), GENMON assumes the following standard configuration definition:

- typewriter : TY10 at level /6
- tape reader : PR20 at level /4
- tape punch : PP30 at level /5
- line printer : LP07 at level /17
- card reader : CR06 at level /15
- cassette tape : TK05 at level /14
  TK15 at level /14
  TK25 at level /14
- magnetic tape : MT04 at level /13
  MT14 at level /13
  MT24 at level /13

If the user replies N , i.e. if one or more of the device addresses or levels is different from the standards above, GENMON outputs the following list on the typewriter, thereby allowing the user to define the configuration himself:

TY:
PP:
PR:
LP:
TK:
MT:
CR:

DK: (answer NO, for no discs used in configuration)
LKM LEVEL: (standard = 1)
RTC LEVEL: (standard = 2)
PANEL INTERRUPT LEVEL: (standard = 7)

For each of the devices listed, the user can reply as follows:
- (CR) if he wants the standard address and level (see above);

- <address>,<level> if one of these is different from the standards, followed by (CR)
- N or NO followed by (CR) if he wants the device excluded from the system.

When the user has terminated his reply to the first question, GENMON types out:
STANDARD FILE CODE ASSIGNMENT?

The procedure here is the same as for the first question: the reply may be either Y[ES] or N[O].
If the user replies Y or YES followed by (LF) (CR) GENMON assumes the following standard file code assignments:
- file code 1: TY10 (system keyboard)
- file code 2: LP07 (listing output)
- file code 3: TK15 (object output)
- file code 4: TK05 (object input)
- file code 5: TY10 (system typewriter)
- file code 6: TK05 (object input)
- file code 7: TK25 (object input)
- file code 8: PR20 (object input)
- file code A: TY10 (sysgen source input)
- file code B: TK15 (sysgen object output)
- file code C: TK05 (casload object input)

If the user replies N or NO to this second GENMON question, i.e. if one or more of his file code assignments is going to be different from the standard list above, GENMON outputs the following list on the typewriter, thereby allowing the user to give his own file code assignments (since the GENMON

processor is the same for all monitors, some of the file codes given in this list are irrelevant to the generation of a COM' and the user must type in NO after those):

LOAD INPUT DEV. AND MAIN LKE INPUT DEV.	F.C./4:	(standard = TK05)
SYSGEN INPUT DEV.	F.C./A:	(standard = TY10)
SYSGEN OUTPUT DEV.	F.C./B:	(standard = TK15)
AUX. LKE INPUT DEV 1	F.C./6:	(standard = TK05)
AUX. LKE INPUT DEV 2	F.C./7:	(standard = TK25)
AUX. LKE INPUT DEV 3	F.C./8:	(standard = PR20)
AUX. LKE INPUT DEV 4	F.C./9:	(type in NO)
IPLGEN/LKE/CASLOAD OUTPUT DEV.	F.C./3:	(standard = TK15)
LISTING OUTPUT DEV	F.C./2:	(standard = LP07)
CASLOAD INPUT DEV.	F.C./C:	(standard = TK05)
DISLOAD INPUT DEV.	F.C./E2:	(type in NO)

For each of the file codes listed, the user can reply as follows:

- (CR) if he wants the standard assignment (see above)

- <dev.name><dev. address> if one of these is different from the standards, followed by (LF) (CR) (or: <dev.name> (LF) (CR))
- N or NO followed by (CR) if he does not want this file code taken into account.

When the user has terminated his reply to this question, GENMON types out:
END OF GENMON INITIALIZATION
READY TO LOAD PROGRAMS

Now the user can proceed to the next phase: COMGEN

Notes on GENMON:
For the question STANDARD CONFIGURATION:

- From the time the MC button has been pushed up to the end of GENMON initialization, no Ready interrupts (from cassette or magnetic tape especially) should occur.

- If the user has answered N or NO to this question, in the list typed out by GENMON specification of a level is mandatory for LKM LEVEL. For RTC LEVEL it is mandatory if the CPU key is in ON/RTC or LOCK position. For PANEL LEVEL it is also mandatory.

For the question STANDARD FILE CODE ASSIGNMENTS:

- The standards imply sysgen from cassette tape. Therefore, for BOM system generation the answer to this question must be NO and the standards must be redefined.
- file code /4: all programs will be loaded from this file code. During the syslink phase it is used as GENLKE object input, so it must be punched tape or cassette.
- file code /A: from this file code the parameters for the COM table generation (COMGEN) will be read. This may be done in interactive mode (e.g. TY) or not (e.g. punched tape reader, cassette tape, magnetic tape or card reader).
- file code /B: this may be punched tape, cassette tape or magnetic tape.
- AUX. INPUT DEV.: these may be assigned in advance, especially for syslink, if libraries are to be scanned on various devices.
- file code /3: this is the main output device (sequential), i.e. punched tape, cassette tape or magnetic tape.
- file code /2: for logging of the sysgen operation: LP. If no line printer is available, type in NO.
- File codes /4, /A, /B, /3 are mandatory; file codes /6 up to /9 and /2 are optional.

- When answers to GENMON questions are given on an ASR typewriter, they must not be typed in before the bell signal because of the low speed of the GENMON I/O module. This is not necessary for devices on V-24 interface, such as the matrix typewriter P842, because they work in echo mode.

COMGEN

When GENMON initialization is terminated and the message READY TO LOAD PROGRAMS has been output the user can start the second phase, COMGEN, the building of the COM tables. This is done by typing in replies to questions output on the typewriter by COMGEN. From these replies COMGEN builds the tables and records them on the medium with file code /B, i.e. in standard cases cassette drive TK15.
When the questions and answers are handled via the typewriter, this phase is done in conversational mode. It is also possible however, to do it in non-conversational mode, for example by having the questions and answers pre-recorded on punched tape. In such a case file code /A must have been assigned to tape reader during the GENMON phase under the question STANDARD FILE CODE ASSIGNMENTS?

In any case, the COMGEN phase is started as follows:
- push the INT button on the CPU control panel
- on output of M: type in LD
- now the following sysgen processor, i.e. COMGEN is loaded from the cassette tape and its identification is output:
  IDENT COMGEN
- when loading is terminated,
  :EOS
  :EOF
  is output on the typewriter.

(At this point, if file codes /A and/or /B have been redefined under GENMON, i.e. if they are not assigned to magnetic tape cassette, prepare the relevant device. Normally, cassette drive TK15 should contain a working cassette now and be ready to receive the tables output by COMGEN).

- push the INT button
- on output of M: type in ST
- Now COMGEN is started and outputs the following messages on the typewriter:

SYSGEN P852/P856/P857 COS
TABGEN INITIALIZATION
IDENT SYSTEM DEFINITION

- then, if the user works in conversational mode, a series of questions is output, to which the user must type in the replies:

IDENT

Reply: Specify a character string of up to 6 alphanumeric characters, to be punched at the beginning of the module. This may be followed by a character string of up to 73 characters, containing comments. The comment field must be separated from the ident field by a blank.

Error Message: TG03: the first character is not alphabetic.

MEMORY SIZE

Reply: Specify, with two hexadecimal characters, the size of the machine memory in 1k words. The number must be a multiple of 8k and may not exceed /20.

Example: 10: 16k memory.

Error Messages: TG03: the specified number is not hexadecimal.
TG04: the size is not a multiple of 8k
TG05: the size exceeds 32k words.

STACK SIZE

Reply: Note, that each device requires up to 10 words in the stack for each interrupt managed for that device. Specify up to 4 hexadecimal characters, giving the size in words, of the stack which is used by the system to save registers when an interrupt occurs.

Error Message: TG03: the number specified is not hexadecimal.

USER INTERRUPT ROUTINES

Reply: Specify all user interrupt routines (usually drivers for special devices) which must be link-

edited with the monitor for inclusion in the system, as follows:

```
<L>,<ENTRYi>
    :
   END
```

L,ENTRYi specifies the name (up to 6 characters), i.e. the start address of the interrupt routine connected to level L.

END indicates that all routines have been declared or is used if none are entered.

Error Messages: TG03: syntax error
TG06: the first character of the name (ENTRYi/j) is not alphabetic
TG09: error in the level declaration.

POWER FAILURE

Reply: Specify the level of power failure option as follows:
<L> giving the level number in 1 or 2 hexadecimal digits.

Specify N if this feature is not available in the system.

Note: The standard system power failure routine performs only a HALT.

Error Message: TG03: parameter error.

REAL TIME CLOCK LEVEL

Reply: Specify the level for the real time clock, in one or two hexadecimal digits.

LKM LEVEL

Reply: First specify the level to which the LKM interrupt is connected in the same manner as for

power failure and real time clock. Then, define the standard monitor requests necessary for your system. COMGEN prints the names of these monitor requests one by one, and after each one the user can answer:
Y if the monitor request is required
N if it is not wanted.
All monitor requests are optional.
The following list of standard monitor requests is printed by COMGEN:

IORM	(Input/Output Requests)	necessary for standard system
WAIT	(Wait for an Event)	
EXIT	(Exit)	
GTBUF	(Get Buffer)	required with FORTRAN
FRBUF	Release Buffer)	
PSE	(Pause) (recommended for cassette and magnetic tape handling)	
ABRT	(Control Abort) (required for Debug)	
CFM	(Cassette File Management (required for COM)	

Error Message: TG06: the reply was neither Y nor N. The user must type the correct reply.
TG08: syntax error.

USER LKM

Reply: The user may specify his own set of monitor requests for inclusion in the monitor, as follows:

⟨N1⟩,⟨ENTRY1⟩
.
.
.
⟨Ni⟩,⟨ENTRYi⟩
.
.
.
⟨Nn⟩,⟨ENTRYn⟩
END

where Ni consits of two hexadecimal digits defining the DATA ⟨number⟩ which follows the LKM instruction, and ENTRYi is the symbolic entry point of the routine processing the LKM DATA Ni

function.
The system will extend the monitor request table in which word i contains the address of ENTRYi. Therefore, during GENLKE, the user must provide the Linkage Editor with the module containing all entry points ENTRYi specified here.
END indicates that all user LKM functions have been declared or that none are wanted.

Note: Ni must not be equal to any of the standard LKM DATA numbers.

Error Message: TG03: the first parameter is not hexadecimal
TG07: the first character of the second parameter is not alphabetic
TG08: syntax error.

PANEL LEVEL

Reply: When operator communication (OCOM) modules are selected, the user must specify the level to which the control panel interrupt is connected.
The level is specified as:
<L> a level number of 1 or 2 hexadecimal digits.
Specify N if no operator communication feature is used.

Error Message: TG03: parameter error.

OCOM OPTIONS

Reply: Specify if the operator communication package containing the operator commands must be included, by typing:
Y if it must be included, or
N if it must not be included.
If the reply was Y, COMGEN will print, one by one, the operator commands available. By typing Y or N after each one, the user indicates whether he wants it to be included or not.
The following list is output:
DM (Dump Memory)
MC (Manual Control) (required for magnetic tape and cassette handling)

CF (Clear File)
WH (Write Header)
SH (Search Header)
WF (Write First Header)
RN (Run Program)
} required for Cassette File Management

HD (Halt Dump) (recommended if DM has been selected)

WM (Write into Memory)

LD (Load a Program)
ST (Start a Program)
} (required for Debug)

RY (Retry Device)
RD (Release Device)
} (see Note below)

AS (Assign File Code)

AB (Abort a Program)

PS (Pause)

RS (Restart) (required if PS or PSE (LKM) selected)

Note: When the commands RY and RD are not selected, the system will stop when an I/O error occurs which requires operator's intervention. In a configuration with only a typewriter these two commands are not needed, but if more devices are used, they are highly recommended.

USER COMMANDS

Reply: Specify any user-made operator commands which must be included in the system, as follows:

<CM1>,<ENTRY1>
|
<CMn>,<ENTRYn>
END

where CMi consists of 2 characters defining the operator command and ENTRYi specifies the entry point of the module which processes command CMi.
END indicates that all commands have been declared or that none are required.

Note: CMi must not be equal to any of the standard operator command mnemonics.

Error Message: TG03: syntax error.

DEVICES ON PROGRAMMED CHANNEL

(Note: devices on channels must be declared according to Release Bulletin)

Reply: Specify, which devices are connected to the programmed channel, as follows:

⟨DNDA⟩,⟨L⟩
:
END

where:
DN is the device name, in two ASCII characters.
DA is the device address, in two hexadecimal digits.

⟨L⟩ specifies the level to which the device is connected.

END indicates that all devices have been declared.

Note: For the opertor's typewriter, three devices (TY, TP, TR) must be declared with the same address, if they will all be used.
No check is made on device declaration.

Error Messages: TG01: the device specified is not supported by the system
TG02: device address error
TG09: level error
TG10: device not declared.
TG06: missing parameter.
TG03: parameter error.

DEVICES ON MULTIPLEX

Reply: Specify, which devices are connected to the multiplex in the same manner as for the devices on the programmed channel.

Note: For line printer there are additional parameters:

- Line printer:
  LPDA,L,LG,⟨number⟩
  where LG=⌊S⌋L⌋ : S = 80-column printer
  L = 132-column printer.
  ⟨number⟩: a hexadecimal number specifying the number of lines per page wanted.

No check is made on the device declaration.

Device names used:

TR : ASR tape reader
TP : ASR tape punch
PR : high-speed reader
PP : high-speed punch
TY : operator's typewriter
CR : card reader
LP : line printer
TK : cassette tape unit
MT : magnetic tape unit

HIGHEST FILE CODE NUMBER

Reply: Specify the highest file code number (from /01 to /FF) used in any of the programs. This will indicate how many words the system must reserve for the File Code Table.

Error Message: TG03: syntax error.

CFM. UNITS (appears only if Y was answered for CFM under the question LKM LEVEL)

Reply: Specify the cassette tape drives making use of Cassette File Management, as follows:

```
TL⟨xx⟩
 |  |
 |  |
TL⟨xx⟩
EN
```

where <xx> is the device address of the cassette tape unit.
If the user does not want to select CFM he must reply EN. The following question will then be skipped.

Example:

```
TL05
TL15
TL25
EN
```

NBER OF CASSETTE FILE NAMES (this question is asked only if Y was answered to CFM under LKM LEVEL)

Reply: The user must type in a number equal to or larger than 2. The number is hexadecimal. This number is used by COMGEN to reserve space for the dynamic catalogue (which keeps the file headers used during a run).

SPECIFY FILE CODES

Reply: Specify all file codes used by the programs. If system processors are used, their standard file codes must be declared. The standard file codes are: 01:source input, 02:listing output, 03:punch output, 04:object input, 05:operator's typewriter. Declaration is done as follows:

```
<FC>,<DNDA>
      :
<FC>,<DNDA>
      :
    END
```

where FC is a file code (2 hexadecimal digits) assigned to the device indicated by DN with address DA.

At this point, a file code can be assigned to the cassette units specified under CFM UNITS.

Note: File code 05 is used by the system for input/output to/from the operator's typewriter. It cannot be assigned or reassigned by AS operator commands.
So, if used, it is necessary to declare at least this file code. All other file codes may be assigned at execution time by means of the operator command AS.

Error Message: TG08: syntax error
TG10: device has not been declared.

FILE CODES ASG TO USER DEVICES

Reply: The user may declare all file codes assigned to his non-standard devices. The related I/O drivers (including Device Work Tables) and the interrupt routines for these devices must be written by the user and be incorporated in the system during the GENLKE phase. (The entry points for the interrupt routines must be declared under USER INTERRUPT ROUTINES). These file codes must be declared here, as they cannot be assigned by AS operator message. The reply must be as follows:

/FC/,<DWTi/

END

where:
FC is a two-digit hexadecimal file code which will be generated in the File Code Table.
DWTi is the entry point (a name of up to six characters) of the Device Work Table (DWT) associated with this device.
END indicates that all file codes have been declared.
Any number of file codes can be assigned to the same DWTi. The system checks only if the file code is a two-digit hexadecimal number, but not if

it has already been used or is one of the standard file codes used by the system processors.

Error Messages: TG03: syntax error, e.g. the file code is not a hexadecimal number.
TG07: the first character of the DWT is not an alphabetical character.

ABORT MODULE (This question is printed only if N was replied for AB under OCOM OPTIONS and ABRT under LKM LEVEL)

Reply: If neither the monitor request 'Keep Control on Abort' nor the operator command AB have been selected, the user may include the system abort module by typing Y. If the reply is N, the abort module will not be included. Abort of the user program will, if the module has been selected, result in an output message and the user may enter a new command to run the next job. If the module has not been selected, an abort will stop the machine, through a Halt instruction.

Error Message: TG06: invalid reply.

SIMULATED INSTRUCTIONS

Reply: Y or N, depending on whether the simulation package must be included or not. If the reply was Y (only for P85 the following list is output:

MULTIPLY:
DIVIDE:
D ADD: (= double add)
D SUB: (= double subtract)
D SHIFT:(= double shift)
MLR - MSR: (= multiple load/stor register)

After each item, the user must type in Y or N to indicate whether he wants that instruction simulation routine included or not.

SIMULATED ROUTINES SAVING AREAS NB

Reply: This question is output only if the reply to the previous question was Y.
The user must type in the number of save areas required by the simulation package.

MAX NUMBER OF SCHEDULED LABELS

Reply: Type in a two-digit hexadecimal number, specifying the maximum number of scheduled labels which may be in queue at the same time. This will be the length of the FILLAB table described in the paragraph on Scheduled Labels.

Note: This is not the maximum number of scheduled labels used in the program, which may be a higher number.

TABGEN ENDED

Take the working cassette out of cassette drive TK15 (file code /3).

IPLGEN

During this phase an Initial Program Loader for the user's system will be generated. This implies, that at this point he must prepare his syetem medium. In standard cases, the output from IPLGEN will be onto cassette in drive TK15 (file code /3 under question STANDARD FILE CODE ASSIGNMENTS of GENMON phase), so the user must put his system cassette S into drive TK15 with side A up and wait until it has been rewound.

Now IPLGEN can be loaded and started:

- push the INT button on the CPU control panel
- on output of M: type in LD
- now the next sysgen processor is loaded from the generation cassette in TK05 and its identification typed out:
  IDENT IPLGEN
  when loading is terminated,
  :EOS
  :EOF
  is output
- push the INT button
- on output of M: type in ST
- Now the IPLGEN processor is started and an Initial Program Loader (IPL) is written at the beginning of the user's system medium.

Note:

- When the IPL is generated onto cassette or magnetic tape, the positioning of this tape must remain stable until the user's monitor which must follow it is recorded onto it. Remember that a cassette is rewound when taken out of the drive and put back in again!

GENLKE

During this phase the final user monitor is obtained by linking the tables generated under COMGEN with the monitor modules required from the COM Library on side B of generation cassette G1 and possibly, with modules from user and/or extension libraries.

This implies some manual actions on the casettes but first GENLKE must be loaded into memory:

- push the INT button on the CPU control panel
- on output of M: type in LD
- now GENLKE is loaded from the cassette tape and its identification is output:
  IDENT GENLKE
  when loading is terminated,
  :EOS
  :EOF
  is output on the typewriter.

On the basis of the availability of only two cassette drives, the cassettes are now handled as follows:

- take generation cassette G1 from TK05 after GENLKE has been loaded.
- put the working cassette W with the tables generated under COMGEN in drive TK05 and wait for it to rewind.
- start GENLKEas follows:
  push the INT button
  on output of M: type in ST
- now GENLKE outputs
  L:
  and the user must type in the link-edit command as follows:
  E⌈:<decimal number>⌉,<module name>⌈,⌊8|4⌋⌉
  where <decimal number> consists of 3 digits:
  - the first is the file code for the object output device
  - the second is the file code for the listing device

- the third is the file code for the object input device.

<module name> is the name of the user's COM;
8 or 4 is used if the monitor is punched on paper tape to indicate whether it must be punched in 4-track or 8-track format.

If the standard file codes are used (see under GENMON, i.e. /3, /2 and /4 respectively), they need not be specified and the command can be given as:

E,<module name>

- then GENLKE outputs

L:

to which the user must reply with:

P

The tables generated during COMGEN are now recorded from the cassette W in TK05 onto the user's system cassette in TK15.

- when this is finished take the working cassette W from TK05
- put the generation cassette G1 with side B up into drive TK05 and wait for it to rewind. It is now positioned correctly for the scanning of the COM Library.
- in response to the

L:

output by GENLKE, now type in

L

upon which GENLKE will start scanning the COM LIbrary, select the required modules and record them onto the user's system cassette in TK15. The names of the selected modules are output on the listing device, together with their base addresses and any comments included in the identifiers.

- when this is finished, GENLKE again types out

L:

The user must now type in

U

to check if there are any unsatisfied references. The last module to be included must be INIMON, so if GENLKE types

INIMON

after the user has typed his first U, it is correct. The

GENLKE processor then types out

L:

and the user can type

L

to solve this last unsatisfied reference.

(If there were more unsatisfied references, the user must repeat this L:L process until INIMON is the last unsolved reference and then give his last L:L command. Otherwise, reload the COM Library cassette, rewind and try again from L:L.)

- Now, after all modules have been included, GENLKE again types

L:

The user once more types

U

to make sure that all references have been solved. Then, on the next

L:

the user types

T

to indicate the end of the GENLKE phase.

GENLKE then outputs the symbol table of the generated COM on the listing device and on the typewriter it outputs monitor length (L=XXXX), the monitor start address (S=XXXX) and the first free location after the monitor in memory (E=XXXX).

Note:

If the user has 3 cassette drives and wants to use them all during this phase, the procedure is as follows:

- take the generation cassette from drive TK05 and put it back in with side B up (COM Library). Wait for it to rewind.
- put the working cassette W, containing the tables generated under COMGEN in drive TK25 and wait for it to rewind.
- start the GENLKE processor:

  INT button

  M: ST

  and on output of

  L:

  type in the option message as follows:

E:327,<module name>,8
where 3 and 2 are the normal object output and listing file codes, but 7 is assigned to input file code which normally is /4 (TK05), but now must be /7 (aux. LKE input dev. TK25), for TK25 contains the working cassette with the tables. See also GENMON for the assignments.

- when this is accepted, GENLKE outputs

L:
and the user must type
P
Then GENLKE processes the input file (/7), i.e. the working cassette, up to EOF. On output of
L:
the user must type in
H
which puts GENLKE into Pause state.

- it must then be restarted with an RS command with a new input file code, switching it back from /7 to /4, the normal input file code and the one assigned to TK05, which contains the COM library which is now to be scanned. So push the INT button and on output of M: type

RS 04
On output of
L:
type in
L
GENLKE now starts scanning and selecting the required modules from the COM library and the rest of the procedure is the same as described above, starting after the user's first L command.

Note:
If modules from other libraries beside the COMLIB must be link-edited during this phase, they must be scanned in a defined order, which is:
- User Libraries, if any
- Extension Libraries
- COMLIB

When more references than INIMON remain unsatisfied, rescanning must start at the first step in this sequence.

Now the user has generated his Cassette Operating Monitor on the system cassette in cassette drive TK15. He can now proceed to the CASLOD phase, to record the system processors on the same cassette.

CASLOD

This processor is used, under GENMON, to record the system processors on the user's system cassette, in the Compact system of labelling.

CASLOD uses the following file codes:

- input file code /0C.
  This is normally the cassette in drive TK05. It must in any case be assigned to a cassette drive. The input must consist of load modules separated by a tape mark each, i.e. the standard system processors. The end of the set of processors is indicated by two consecutive tape marks.
  The value of the input file code can be changed by means of an RS command when the CASLOD program is in Pause state (see CASLOD control commands below).
- output file code /03.
  This file code must always be assigned to a cassette drive. Normally it is drive TK15. The cassette in this drive must at this point at least contain the IPL and the newly generated user monitor, on side A of the cassette.
- operator file code /0A.
  This is the typewriter (TY10), through which the user may give one of 7 control commands to CASLOD and on which CASLOD will print error and information messages.

CASLOD is loaded and started not more than two times during the recording of the system on the system cassette, once after the monitor has been generated and once after generation of the user's linkage editor.
In both cases CASLOD writes an EOF label after the previously recorded program (monitor, linkage editor). Before each recorded program CASLOD writes a file header, in accordance with the standards for the Compact labelling system. This header (HDR) will get the same name as in the IDENT field of the recorded program. These IDENTs must all be different.
On side B of the generated cassette CASLOD will write a First Header Label (see also the chapter on the Labelling System).

First the user must load the CASLOD program from the cassette in drive TK05:

- push the INT button on the CPU control panel
- on output of M: type in
  LD
- now the CASLOD processor is loaded into memory and its identification is output:
  IDENT CASLOD
- when loading is terminated,
  :EOS
  :EOF
  is output on the typewriter.

- take the cassette G1 out of cassette drive TK05
- put the second generation cassette G2 in drive TK05 with side A up and wait for it to rewind
- to start the CASLOD program, push the INT button and, on output of M: type in
  ST
- CASLOD outputs the message
  CASLOAD PROGRAM
- after writing an EOF and 2 tape marks after the monitor on the user's system cassette (this takes a few minutes, so wait until the orange light over the cassette is out), it starts reading from file code /0C (i.e. cassette G2 in TK05) and prints the Ident of the first program it encounters, i.e.
  PREMTK
  for the Cassette Premark Program, followed by
  CASC:
  and waits for the user to type in a CASLOD Control Command, unless an erroneous condition has been encountered, in which case CASLOD may output one of the following error messages:
  - INCOMPATIBLE ASSIGN - PAUSE (load new input tape)
  - INCORRECT LABELLING (reload CASLOD program)
  - FATAL ERROR ON OUTPUT (reload CASLOD program)
  - FATAL ERROR ON INPUT (reload CASLOD program)

The CASLOD Control Commands which the user may type in after CASLOD has typed out CASC: are the following:

(LF) (CR) : copy current module

NX : go to next module

HT : write header and exit

EN : end of process

Y: premark side B of the generated cassette and exit

N: do not premark but exit

PS : pause.

Copy Current Module

Syntax: (LF) (CR)

Use: When this command is given, CASLOD writes a file header label on the system cassette with volume identifier (system cassette), file identifier (file name as in Ident field) and ASCII zeros in the other fields.
Then it copies the input file onto the system cassette until an EOF is read. CASLOD writes an EOF label on the system cassette after the newly loaded program and prints the message END OF INPUT: XXXXXX
Then it reads the next Ident cluster on the input file, prints the program name found on the typewriter, followed by CASC: and waits for the next command.

Error Messages:

- ERROR ON TAPE OUTPUT (CASLOD rewinds input file and output file and waits for next command)
- ERROR ON TAPE INPUT (same as previous error)
- END OF PROGRAM SET (CASLOD waits for next command)
- UNKNOWN COMMAND (CASLOD waits for next command)
- FATAL ERROR - OUTPUT (CASLOD exits)
- FATAL ERROR - INPUT (CASLOD exits)
- NO IDENT NAME IN THE FIRST BLOCK (CASLOD exits)

Get Next Module

Syntax: NX

Use: This means that the current module must not be copied onto the system cassette. CASLOD scans the input file for the next program, prints its name on the typewriter, followed by CASC: and waits for the next command.

Error Messages:

- ERROR ON TAPE OUTPUT or INPUT (CASLOD rewinds input file and output file and waits for next command)
- END OF PROGRAM SET (CASLOD waits for next command)
- UNKNOWN COMMAND (CASLOD waits for next command)
- FATAL ERROR - INPUT (CASLOD exits)

Write Header and Exit

Syntax: HT

Use: When this command is given, CASLOD writes a file header label with volume identifier (system tape) and file identifier (Ident of current program). The layout of the header must correspond with the specifications of Compact Labelling.
Having written the HDR label, CASLOD exits.
This command is used to write the header label for the user Linkage Editor, before it is generated.
CASLOD rewinds the input file to the first tape mark before it exits.

Error Messages:

- ERROR ON TAPE OUTPUT or INPUT (CASLOD rewinds input file and output file and waits for next command)
- UNKNOWN COMMAND (CASLOD waits for next command)

End Of Process

Syntax: EN

Use: When this command is given, CASLOD exits. Because the generated cassette is not a premarked cassette, the user must ensure that the first block on its side B is an HDR label with the same volume identifier as on side A of the cassette.
When this command is given, CASLOD types out a message asking whether side B must be premarked:
PREMARK FACE B?
followed by CASC:. The user must type in YE⌈S⌉ or NO:
- NO: CASLOD exits
- YES: CASLOD types out CHANGE TRACK AND #RS, and goes into Pause state. The user must reload the cassette with side B up, push the INT button, type in RS after which CASLOD is restarted and writes a header label with the same volume identifier on side

B of the cassette. Then it exits.

Error Messages:

- UNKNOWN COMMAND (CASLOD waits for next command)
- ERROR ON TAPE INPUT or OUTPUT (CASLOD rewinds input and output file and waits for next command)
- PREMARK FACE B

Pause

Syntax: PS

Use: When this command is given, CASLOD goes into Pause state. It can be restarted by an RS operator command, which may be followed by a parameter specifying a new input file code. CASLOD checks the validity of this file code:

- if it is valid, it will start reading the new input file and print the first Ident name encountered on the typewriter, followed by CASC:
- if it is not valid, CASLOD goes into Pause state
- if no new file code was specified in the RS command, CASLOD prints out CASC: and waits for the next command.

Error Messages:

BAD FILE CODE (CASLOD goes into Pause state)

- normally then, after CASLOD has typed out PREMTK (first Ident encountered), followed by CASC: the user types

(LF) (CR)

to indicate that the Cassette Premark program must be recorded on the system cassette
- the CASLOD processor writes the file header label for the Cassette Premark and records it on the system tape. Then it types END OF INPUT: PREMTK and the name of the next program it encounters on the input file, i.e.

UPD

followed by

CASC:

and waits for the next command from the user.
- the user may type

(LF) (CR)

and CASLOD will then record it as the next program on the system tape. (For the other possible commands, see above.)

- then CASLOD types
  END OF INPUT: UPD
  scans the input file and types out the name of the next program,
  ASM
  followed by
  CASC:
  To include it in his system, the user must type
  (LF) (CR)
- then CASLOD types
  END OF INPUT: ASM
  scans the input file and types out the name of the next program,
  LKE
  followed by
  CASC:
- the user must type in
  HT
  after which CASLOD writes the file header label for the LKE and then exits.
- now the user must load the next program from the input file, i.e. the first LKE, by pushing the INT button and on output of M: typing in
  LD
- LKE is loaded into memory. When its :EOS and :EOF have been typed out, push the INT button and on output of M: type in
  ST
- LKE then types out
  L:
  after which the user must type the LKE option message:
  E ⌈:<number>⌉ , < name>⌈,⌊4 | 8⌋⌉ (this will normally be E,LKE)
  as under GENLKE (see section GENLKE)
- when this is accepted, LKE types out
  L:
  after which the user must type in
  R␣<number>
  where <number> is the memory area which must be reserved for the generation of the user's Linkage Editor. This is calculated as follows:

<number> = memory size - (monitor size + LKE size + /80)
where:
- memory size is known
- monitor size = INIMON start address (typed out during GENLKE)
- LKE size = ending address (typed after :EOS when loaded) - start address (typed when loaded)
- /80 = communication area + save area.

<number> must be in hexadecimal.

- after this has been accepted, LKE outputs
  L:
  on the typewriter and the user must type
  P
- the system types out LKE <number - 1> and the second LKE is loaded from cassette. The user's Linkage Editor is generated and recorded on his system tape, <LKE> being the name of the file on which the Linkage Editor is recorded.
- when LKE types out
  L:
  the user must type
  T
  to terminate the process. LKE exits
  and outputs the message
  L = XXXXXX S = XXXXXX E = XXXXXX
  where L= length of the relocatable program section
  S= relative or absolute start address
  E= highest absolute address (0000 if the whole module is relocatable).

- Linkage Editor generation being completed, the user must load the next program from the input file in TK05:
  push the INT button and on output of M: type
  LD
- the CASLOD program is reloaded and its identification output:
  IDENT CASLOD
- when loading is terminated,
  :EOS
  :EOF
  is output on the typewriter and the user must start CASLOD

by pushing the INT button and on output of M: typing in
ST

- CASLOD first writes an EOF label after the tape mark following the newly generated user's Linkage Editor on the system tape. This takes a while; wait till the orange light goes out.
- then CASLOD reads the next program from the input file (normally still /0C) and prints out its identification:
  DEBUG
  for the Debugging Package, followed by
  CASC:
  and waits for the user to type in a CASLOD Control Command.
  If the user types
  (LF) (CR)
  the debugging package is recorded onto his system tape. (For the other possible commands, see above.)
- after this CASLOD types out
  END OF INPUT: DEBUG
  followed by
  END OF PROG. SET
  and
  PREMARK FACE B?
  CASC:
- if the user wants to premark side B of the cassette, the procedure is as under Premark (see Part 9, chapter 3), otherwise he types
  NO
  upon which CASLOD outputs the message
  CASLOD ENDED

This also terminates the system generation process, for the user now has a complete Cassette Operating System on the magnetic tape cassette in file code /03 (normally TK15).

The FORTRAN processor and Library can be recorded on side B of the system cassette by means of the Cassette Update (see Chapter on Cassette Update).

SYSGEN EXAMPLE

Below, an example is given of the generation of a COS system, as described in the preceding pages.
The system is generated using two cassette drives, TK05 (file code 4) and TK25 (non-standard drive for file code 4; see GENMON). The generation is started by setting up /0785 or /1785 on the data switches and with generation cassette G1, side A in TK05 and a working cassette W in TK25. The user needs a cassette S for the generated system.
The example shows the typewriter output as well as the line printer listing, which is shown indented to the right.

```
  IDENT GENMON          5111 100 23561 COS BASIC PACK. CAS. 1/2
:EOS
:EOF

**GENMON**05**

STANDARD CONFIGURATION ? N      (underlined = typed by user.)
TY: (LF) (CR)
PP:
PR:
LP:
TK:
MT: N
CR: N
DK: N
LKM LEVEL:
RTC LEVEL:
PANEL INTERRUPT LEVEL:

STANDARD FILE CODE ASSIGNEMENT ? N
LOAD INPUT DEV. AND MAIN LKE INPUT DEV. F.C. /4 :
SYSGEN INPUT DEV.                       F.C. /A :
SYSGEN OUTPUT DEV.                      F.C. /B : TK25
AUX. LKE INPUT DEV *1*                  F.C. /6 :
AUX. LKE INPUT DEV *2*                  F.C. /7 :
AUX. LKE INPUT DEV *3*                  F.C. /8 :
AUX. LKE INPUT DEV *4*                  F.C. /9 :
IPLGEN/LKE/CASLOAD OUTPUT DEV.          F.C. /3 : TK25
LISTING OUTPUT DEV                      F.C. /2 :
CASLOAD INPUT DEV.                      F.C. /C :
DISLOAD INPUT DEV.                      F.C. /E2: N
```

```
**GENMON**05**

STANDARD CONFIGURATION ?  N
TY: 10,/06
PP: 30,/05
PR: 20,/04
LP: 07,/17
TK: 05,/14
TK: 15,/14
TK: 25,/14
LKM LEVEL: /01
RTC LEVEL: /02
PANEL INTERRUPT LEVEL: /07
```

```
STANDARD FILE CODE ASSIGNEMENT ?  N
LOAD INPUT DEV. AND MAIN LKE INPUT DEV.  F.C. /4 : TK05
SYSGEN INPUT DEV.                        F.C. /A : TY10
SYSGEN OUTPUT DEV.                       F.C. /B : TK25
AUX. LKE INPUT DEV *1*                   F.C. /6 : TK05
AUX. LKE INPUT DEV *2*                   F.C. /7 : TK25
AUX. LKE INPUT DEV *3*                   F.C. /8 : PR20
AUX. LKE INPUT DEV *4*                   F.C. /9 : NO
IPLGEN/LKE/CASLOAD OUTPUT DEV.           F.C. /3 : TK25
LISTING OUTPUT DEV                       F.C. /2 : LP07
CASLOAD INPUT DEV.                       F.C. /C : TK05
DISLOAD INPUT DEV.                       F.C. /E2: NO

** END OF GENMON INITIALIZATION **
** READY TO LOAD PROGRAMS **
```

```
** END OF GENMON INITIALIZATION **
** READY TO LOAD PROGRAMS **

M:LD
  IDENT COMGEN              5111 100 23561 COS BASIC PACK. CAS. 1/2  3D3
:EOS  67BE
:EOF
M:SI

SYSGEN P852/P856/P857  COS  #05
TABGEN INITIALISATION
SYSTEM DEFINITION
IDENT
*MANCOS
MEMORY SIZE
*10
STACK SIZE
*A0
USER INTERRUPT ROUTINES
*END
POWER FAILURE
*0
REAL TIME CLOCK LEVEL
*2
LKM LEVEL
*1
*IORM   : Y
*WAIT   : Y
*EXIT   : Y
*GTBUF  : Y
*FRBUF  : Y
*PSE    : Y
*ABRT   : Y
*CEM    : Y
USER LKM :
*END
PANEL LEVEL
*Z
OCOM OPTIONS :Y
*DM: Y
*MC: Y
*CF: Y
*WH: Y
*SH: Y
*WF: Y
*RN: Y
```

```
  IDENT COMGEN              5111 10
:EOS  67BE
:EOF
SYSGEN P852/P856/P857  COS  #05
TABGEN INITIALISATION
SYSTEM DEFINITION
IDENT
*
MANCOS
MEMORY SIZE
*
10
STACK SIZE
*
A0
USER INTERRUPT ROUTINES
*
END
POWER FAILURE
*
0
REAL TIME CLOCK LEVEL
*
2
LKM LEVEL
*
1
*
IORM   :
Y
*
WAIT   :
Y
*
EXIT   :
Y
*
GTBUF  :
Y
```

```
*HD: Y
*WM: Y
*LD: Y
*ST: Y
*RY: Y
*RD: Y
*AS: Y
*AB: Y
*PS: Y
*RS: Y
USER COMMAND
*END
DEVICES ON PROG CHANNEL
*TY10,4
*PR20,4
*PP30,5
*TK05,14
*TK15,14
*TK25,14
*EN
DEVICES ON MULTIPLEX
*LP07,17,L,32
*EN
HIGHEST FILE CODE NUMBER
*20
CFM. UNITS
*TL05
*TL15
*TL25
*EN
NBER OF CASSETTE FILE NAMES
*20
SPECIFY FILE CODES
*1,TK05
*2,LP07
*3,TL15
*4,TL25
*5,TY10
*8,PR20
*9,PP30
*EN
FILE CODE ASG TO USER DEVICES
*END
SIMULATED INSTRUCTIONS :N
MAX NUMBER OF SCHEDULED LABELS
*2
TABGEN ENDED
```

```
*
FRBUF :
Y
*
PSE   :
Y
*
ABRT  :
Y
*
CFM   :
Y

USER LKM
*
END
PANEL LEVEL
*
?
QCOM OPTIONS :
Y
*
DM:
Y
*
MC:
Y
*
CF:
Y
*
WH:
Y
*
SH:
Y
*
WF:
Y
*
RN:
Y
*
HD:
Y
*
WM:
Y
*
LD:
Y
*
ST:
Y
*
RY:
Y
*
RD:
Y
*
AS:
Y
```

```
*
AB:
Y
*
PS:
Y
*
RS:
Y
USER COMMAND
*
END
DEVICES ON PROG CHANNEL
*
TY10,6
*
PR20,4
*
PP30,5
*
TK05,14
*
TK15,14
*
TK25,14
*
EN
DEVICES ON MULTIPLEX
*
LP07,17,L,32
*
EN
HIGHEST FILE CODE NUMBER
*
20
CFM. UNITS
*
TL05
*
TL15
*
TL25
*
EN
NBER OF CASSETTE FILE NAMES
*
20
SPECIFY FILE CODES
*
1,TL05
*
2,LP07
*
3,TL15
*
4,TL25
*
5,TY10
*
8,PR20
*
9,PP30
*
EN
```

```
FILE CODE ASG TO USER DEVICES
*
END
SIMULATED INSTRUCTIONS :
N
MAX NUMBER OF SCHEDULED LABELS
*
2
TABGEN ENDED
```

Take cassette W out of TK25.
Put cassette S in TK25.)

```
M:LD
  IDENT IPLGEN          5111 100 23561 COS BASIC PACK. CAS. 1/2  3D38
:EOS  4366
:EOF
M:ST
```

```
  IDENT IPLGEN          5111 100 23561 COS BASIC PACK.
:EOS  4366
:EOF
```

```
M:LD
  IDENT GENLKE          5111 100 23561 COS BASIC PACK. CAS. 1/2  3D38
:EOS  78FE
:EOF
```

```
  IDENT GENLKE          5111 100 23561 COS BASIC PACK. CAS.
:EOS  78FE
:EOF
```

Take cassette G1 out of TK05.
Put cassette W in TK05.)

```
M:ST
L?E,MANCOS
L?P
 MANCOS   0000
```

Take cassette W out of TK05.
Put cassette G1, side B in TK05.)

```
L:L
```

```
0000  MANCOS
0A24          I:PFAR  5111 100 23561 COS BASIC PACK. CAS. 1/2
0A2A          I:TRAP  5111 100 23561 COS BASIC PACK. CAS. 1/2
0A52          I:LKM   5111 100 23561 COS BASIC PACK. CAS. 1/2
0AC8          I:RTC   5111 100 23561 COS BASIC PACK. CAS. 1/2
0AFA          NSCHLB  5111 100 23561 COS BASIC PACK. CAS. 1/2
0CB6          WAIT    5111 100 23561 COS BASIC PACK. CAS. 1/2
0CCA          EXIT    5111 100 23561 COS BASIC PACK. CAS. 1/2
0D06          GETBUF  5111 100 23561 COS BASIC PACK. CAS. 1/2
0DEA          FRBUFF  5111 100 23561 COS BASIC PACK. CAS. 1/2
0E6A          IORM    5111 100 23561 COS BASIC PACK. CAS. 1/2
10C2          DRIY01  5111 100 23561 COS BASIC PACK. CAS. 1/2
```

```
12B8          DRPO18  5111 100 23561 COS BASIC PACK. CAS. 1
1328          DRPR38  5111 100 23561 COS BASIC PACK. CAS. 1
138C          DRLP    5111 100 23561 COS BASIC PACK. CAS. 1
15CC          D:TK    5111 100 23561 COS BASIC PACK. CAS. 1
1A8E          ENDIO   5111 100 23561 COS BASIC PACK. CAS. 1
1C26          COMIO   5111 100 23561 COS BASIC PACK. CAS. 1
1D6C          INPUT   5111 100 23561 COS BASIC PACK. CAS. 1
1EEC          OUTPUT  5111 100 23561 COS BASIC PACK. CAS. 1
2038          INTCP   5111 100 23561 COS BASIC PACK. CAS. 1
21A4          M:RETR  5111 100 23561 COS BASIC PACK. CAS. 1
2368          MANCT   5111 100 23561 COS BASIC PACK. CAS. 1
23E0          ABORT   5111 100 23561 COS BASIC PACK. CAS. 1
24A2          PAUSE   5111 100 23561 COS BASIC PACK. CAS. 1
24F0          RSTART  5111 100 23561 COS BASIC PACK. CAS. 1
2516          LOADER  5111 100 23561 COS BASIC PACK. CAS. 1
27E2          WM      5111 100 23561 COS BASIC PACK. CAS. 1
280C          DMH     5111 100 23561 COS BASIC PACK. CAS. 1
2918          BH      5111 100 23561 COS BASIC PACK. CAS. 1
2950          START   5111 100 23561 COS BASIC PACK. CAS. 1
2972          M:CMAD  5111 100 23561 COS BASIC PACK. CAS. 1
2986          CHLEV   5111 100 23561 COS BASIC PACK. CAS. 1
2A38          HEBIN   5111 100 23561 COS BASIC PACK. CAS. 1
2A7A          INIT    5111 100 23561 COS BASIC PACK. CAS. 1
2B6C          CFMPAC  5111 100 23561 COS BASIC PACK. CAS. 1
2C7C          C:FMIO  5111 100 23561 COS BASIC PACK. CAS. 1
2CEE          WRHEAD  5111 100 23561 COS BASIC PACK. CAS. 1
2F32          WRITEL  5111 100 23561 COS BASIC PACK. CAS. 1
30EE          WRITE   5111 100 23561 COS BASIC PACK. CAS. 1
31C2          READYI  5111 100 23561 COS BASIC PACK. CAS. 1
349A          UPDECB  5111 100 23561 COS BASIC PACK. CAS. 1
3502          V:COMN  5111 100 23561 COS BASIC PACK. CAS. 1
357E          SKIPBW  5111 100 23561 COS BASIC PACK. CAS. 1
3602          REWIND  5111 100 23561 COS BASIC PACK. CAS. 1
3702          VREAD   5111 100 23561 COS BASIC PACK. CAS. 1
3842          SEARNX  5111 100 23561 COS BASIC PACK. CAS. 1
39F2          SEARCH  5111 100 23561 COS BASIC PACK. CAS. 1
3CC0          WF      5111 100 23561 COS BASIC PACK. CAS. 1
3DA2          CF      5111 100 23561 COS BASIC PACK. CAS. 1
3E1C          SH      5111 100 23561 COS BASIC PACK. CAS. 1
3EB8          M:ASPR  5111 100 23561 COS BASIC PACK. CAS. 1
4008          ANALYS  5111 100 23561 COS BASIC PACK. CAS. 1
41A8          WH      5111 100 23561 COS BASIC PACK. CAS. 1
423C          CATMAN  5111 100 23561 COS BASIC PACK. CAS. 1
4390          E:RMES  5111 100 23561 COS BASIC PACK. CAS. 1
```

```
L:U
  INIMON

444C          INIMON  5111 100 23561 COS BASIC PACK. CAS. 1
```

```
L:L
L:U
L:T

I:TRAP    R   0A2A
INIMON    R   444C
M:DISP    R   0AFA
I:PFAR    R   0A24
I:RTC     R   0AC0
I:LKM     R   0A52
```

M:IORM	R	0ED0
WAIT	R	0CB6
EXIT	R	0CCA
GETBUF	R	0D06
FRBUFF	R	0DEA
PSMAC	R	24B8
ABADR	R	2402
I:ITCP	R	2046
DMH	R	280C
MANCT	R	2368
V:OCF	R	3DA2
V:OWH	R	41A8
V:OSH	R	3E26
V:OWF	R	3CC0
V:ORN	R	3E1E
HT	R	20DC
WM	R	27E2
LOADER	R	2532
START	R	2950
RYPRO	R	225C
RDPRO	R	225C
M:ASPR	R	3EB8
ABORT	R	2488
PAUSE	R	24A2
RSTART	R	24F0
D:RAS1	R	10C2
D:RPTP	R	12B8
D:RPTR	R	1328
D:RLP	R	1390
D:TK	R	15CE
S:TY	R	1200
S:PP	R	12E4
S:PR	R	134E
S:LP	R	14C2
S:TK	R	17F6
V:REDY	R	31C2
CFMIOR	R	2C94
CFMRET	R	2C9A
V:OST	R	2952
CFMPAC	R	2B6C
SYSAB	R	241C
HLT	R	0084
STB	R	0242
CVT	R	0244
CVTMSZ	R	0244
CVTSTB	R	0246
CVTSBA	R	0248
CVTBBA	R	024A
CVTBKA	R	024C
CVTDSP	R	024E
CVTLDR	R	0250
CVTPLS	R	0252
PCT61	P	008A
CPT	R	0090
LKMAL	R	025E
CSTAB1	R	0270
CSTAB2	R	027C
M:ADD	R	009C

```
C:ONTC    R   0436
D:WT      R   02B6
D:WT1     R   02DC
P:DWLC    A   0026
D:WTEN    R   03C0
F:CT      R   045E
SCLFG     R   0098
FILLAB    R   00AA
CASLFT    R   0254
CVTRED    R   0256
RCFMIO    R   0258
CFMRIT    R   025A
V:LDST    R   025C
LFTAB1    R   04A0
DISPAT    R   0AFA
CKDTC     R   0AC8
EXSCH     R   0B74
CHLEV     R   2998
MAINEX    R   2AC0
PWAIT     R   0CB6
E:SECB    R   0EC2
E:FECB    R   0EC8
E:S011    R   1B4C
E:S015    R   1B64
E:S000    R   1B2E
L:VCH     R   1A8E
E:S012    R   1B52
D:RAS2    R   10CA
D:RAS3    R   10D2
O:TPUT    R   1EEC
S:TIO     R   1C6A
O:TRIO    R   1C76
C:WAIT    R   1C26
H:LTIO    R   1C70
S:SST     R   1C82
C:INPT    R   1C34
C:OUT     R   1C4C
R:TUR2    R   1B6A
R:TUR4    R   1AB6
M:RETR    R   21A4
M:TEX     R   1CB6
I:NRIO    R   1C7C
S:MTC     R   1D0A
E:NDIO    R   1A9A
R:TURN    R   1AAE
R:TUR1    R   1AC4
R:TUR5    R   1ACA
E:S013    R   1B58
E:S014    R   1B5E
R:TUR3    R   1B74
I:NPUT    R   1D6C
STOPFG    R   2044
INHCP     R   2A78
CPRTN     R   20E2
CPRTN1    R   20E4
ECBCP     R   2126
BUFCP     R   2132
ERHB      R   2182
HB        R   2A3E
MCAREL    R   2A80
```

```
BH              R   2918
INHST           R   2A7E
U:ORN1          R   2510
LDFLAG          R   2A7C
M:CMAD          R   2972
PFAR            R   2A7A
INIT            R   2A82
U:RID           R   3702
U:WRTE          R   30EE
U:GTYP          R   3502
U:EOF           R   30EE
U:SKBI          R   357E
U:UNLK          R   3544
WRDECR          R   349A
U:SKEW          R   3402
U:WHDR          R   30EE
U:SHDR          R   3004
U:RLP           R   3616
U:REL           R   348C
U:SCHB          R   3602
U:SCHF          R   3616
U:SCHN          R   3842
U:ENB           R   3540
U:DENB          R   3574
C:EMIA          R   3070
S:ARCH          R   428C
ENDTU           R   3074
U:DYCT          R   42AA
U:DYCD          R   4230
WRITEL          R   2F30
E:RMES          R   4280
SEARN           R   2E42
TRFHDR          R   4278
WHDRED          R   2F32
TRANSF          R   3270
REPTR1          R   31B6
COMPAR          R   3184
HDRTRN          R   3874
COMEND          R   3860
SNEXTH          R   3804
SFNAME          R   4208
STAUTP          R   4010
ANALYS          R   4072
CHCNT1          R   4000
CHCNT2          R   400E
FOPD            R   4010
OCMBUF          R   4022
PNT1            R   4008
PNT2            R   4000
ERRIN           R   416A
STCLIP          R   4020
STLOAD          R   401F
STINPT          R   401A
SYNTAX          R   4190
S:ARC1          R   428C

L = 4488     S = 444D     E = 4488
M:LD
  IDENT CASLOD                5111 100 23561 COS BASIC PACK. CAS. 1/2  2D38
:EOS  494C
:EOF
```

```
  IDENT CASLOD                    5111 100 23561 COS BASIC PACK.
:EOS  494C
:EOF
```

(Take cassette G1 out of TK05.
Put cassette G2, side A in TK05.)

```
M:ST
CASLOAD PROG.
```

(At this point CASLOD must write an EOF. This takes a few minutes. Wait for the orange light over TK25 to extinguish.)

```
PREMTK
CASC: (LF) (CR)

*END OF INPUT:  PREMTK

UPD
CASC:

*END OF INPUT:  UPD

ASM
CASC:

*END OF INPUT:  ASM

LKE
CASC: HT
```

```
CASLOAD PROG.
PREMTK
CASC:
*END OF INPUT:  PREMTK
UPD
CASC:
*END OF INPUT:  UPD
ASM
CASC:
*END OF INPUT:  ASM
LKE
CASC: HT
```

```
M:LD
  IDENT LKE                 5111 100 23571 COS BASIC PACK. CAS. 2/2  3D38
:EOS  58FE
:EOF
M:ST
L:E,LKE
L:P,1F6E
L:P
 LKE       1F6E
L:I
L = 2334     S = 32BB     E = 0000
M:LD
  IDENT CASLOD              5111 100 23571 COS BASIC PACK. CAS. 2/2  3D38
:EOS  494C
:EOF
M:ST
CASLOAD PROG.
```

```
  IDENT LKE                       5111 100 2357
:EOS  58FE
:EOF
1F6E  LKE                       5111 100 23571
```

```
  IDENT CASLOD                    5111 100 2357
:EOS  494C
:EOF
CASLOAD PROG.
```

(At this point CASLOD must write an EOF. This takes a few minutes. Wait for the orange light over TK25 to extinguish.)

```
DEBUG
CASC:

*END OF INPUT:  DEBUG

*END OF PROG. SET
*PREMARK FACE B ?

CASC: NO

*CASLOD ENDED
```

```
DEBUG
CASC:
*END OF INPUT:  DEBUG
*PREMARK FACE B ?
CASC: NO
*CASLOD ENDED
```

ERROR RECOVERY PROCEDURES

GENLKE Error Recovery

If the user has made one or more errors during the GENLKE phase, e.g. with the Unsatisfied References, it is not necessary to completely retry the COS generation process. One possible way to recover the error more quickly is as follows:

- reload generation cassette G1, side A and wait until it has been rewound. Load the GENMON monitor into memory and answer its questions as during normal generation.
- reload the user's system cassette S in drive TK15 (file code /3) and wait until it has been rewound.
- give load commands successively as follows:

```
M:LD
        IDENT COMGEN
        :EOS
        :EOF
M:LD
        IDENT IPLGEN
        :EOS
        :EOF
```

- start IPLGEN:

  M:ST

  and wait for the end of IPLGEN execution.
- Carry on as normal from the GENLKE phase.

CASLOD Error Recovery

One possible way of recovering from a fatal error made during the CASLOD phase is as follows:

- if CASLOD has already written the EOF block behind the generated monitor on the user's system cassette S, the user must:
  - reload this cassette into drive TK15 (file code /3) and wait until it has been rewound
  - give the following Manual Control operator command:

    M:MC 03,16

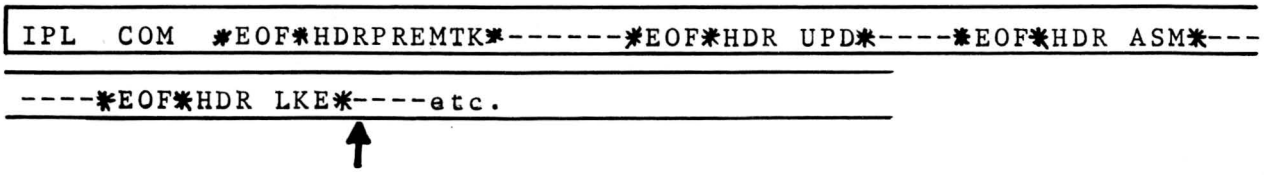
    which will cause the cassette to unwind up to the EOF block, which is the case when the orange light over TK15 goes out.

- then reload generation cassette G1, side A into drive TK05, wait until it has been rewound and go through the GENMON phase as normal.
- remove cassette G1 from drive TK05 and put it back in with side B up.
- type in the Manual Control operator command
  M:MC 04,16,2
  to unwind the generation cassette up to the CASLOD program
- while waiting for the orange light over TK05 to go out, type
  M:LD
  to load CASLOD into memory.
- after loading has been completed and
  :EOS
  :EOF
  has been typed out, the user may carry on generating his system as normal from the CASLOD phase.

LKEGEN Error Recovery

When a fatal error occurs during the generation of the user's Linkage Editor, e.g. when the COS LKEGEN program cannot be executed because the parameter 'memory size reserved for LKE' has been calculated erroneously, a possible way of recovery is the following:

- as the user must know, the part of the system which he has already generated on his cassette is the following:



- therefore the following actions must be taken:
  - put generation cassette G2, side A in drive TK05 and the user system cassette S in drive TK15 and wait until they have been rewound
  - give successive load operator commands (LD) up to LKE
  - position the user's system cassette correctly by means of the Manual Control operator command
    M:MC 03,16,0C
    i.e. make it skip forward over 12 tape marks, so that it will be after the tape mark following the LKE Header
  - start LKE by means of the command ST
  - resume generation as normal from the LKEGEN phase.

This appendix refers to the I/O order which the user must specify in register A7, when he gives an I/O monitor request (LKM1).

BASIC READ (/01) (Standard I/O and CFM)

Operator's Typewriter:
All characters are entered on 8 bits until the requested length is reached.

ASR Tape Reader:
All characters are entered on 8 bits. The reader stops one character after an Xoff code has been read.

High-speed Tape Reader:
All characters are entered on 8 bits, without checking or special features, until the requested length is reached.

Card Reader:
All the words are entered and stored in Hollerith code on 12 bits (4 to 15). In each word the column image is right-justified. The words are stored until the requested length is reached. The length is given in words.

Magnetic Tape Cassette:
All Read/Write operations (Basic, Standard, Object) are the same, with the following characteristics:
- maximum record length: 256 characters.
- required length: block length.

- effective length: block length (without control character).

- all read/write operations are done on the requested length
- incorrect length after read operation: no error, if requested length is greater than block length and the returned status is correct.
- throughput error or data fault: retry is made automatically, up to five times.
  - after read : backspace - read
  - after write : backspace - erase - write.

Magnetic Tape:
Same as for cassette tape, with the following differences:
- maximum record length: 4095 characters; minimum 12 characters.
- required length: block length (2 dummy characters must be reserved behind the buffer).
- physical block length: required length + 2.
- effective length: block length.
- 12 characters are always transferred, in any case.
- incorrect length: see cassette tape above.

STANDARD READ (/02) (Standard I/O and CFM)

Operator's Typewriter:
ASCII characters are entered on 8 bits, with the following special features:

- the special characters, coded from /0 to /1F, are ignored.
- code /7F (Rub-out or Delete character) is ignored.
- code /5F (←) can be used to delete the preceding character. If several ← are used consecutively, an equal number of preceding characters will be deleted.
- code /5E (↑) is used to delete the line preceding it, up to the next carriage return.
- code /0D (carriage return) indicates end of block. It is the last character to be entered. It is not transmitted to the user's buffer.
- code /0A means "line feed". It is ignored and not transmitted to the buffer.
- code /5C (\) is used as a tabulation symbol (see ECB word 5). If the address of the tabulation table is zero, or if the number of tackets is zero, or if the storage address is greater than the last tacket, the code /5C is stored in the buffer. In other cases, /5C is not stored and replaced by spaces, as indicated by the tackets in the tabulation table.

ASR Tape Reader:
For ASCII characters, the same features apply as for the keyboard: the code for carriage return must be preceded by the code for Xoff.
For object code in 4+4+4+4 tape format, the first character identifies the object format. It must be in the range from /18 to /1F and is converted to a number from /0 to /7 and stored on one character. The second character contains the word-count of the input block, excluding the first word and the checksum. Each punched row (4 bits) entered after this identifier is stored on one half-character up to the checksum. When the checksum has been read, input is stopped. The 8+8 tape format cannot be read on the ASR tape reader. To start the reader, an Xon code is sent by the system before entering the characters.

High-speed Tape Reader:
Same as for the ASR tape reader. In addition: for object code in 8+8 format, the first character, identifying the object code format, must have one of the following values: /10, /1 to /4 or /15 to /17. It is converted to a number from /0 to /7. Each punched row (8 bits) entered after this identifier is stored on one character up to the checksum. The second character is the length of the block, in words, excluding the first word and the checksum.

Card Reader:
All words are read in Hollerith code, on 12 bits, converted and stored in ASCII code, on 7 bits, until the requested length is reached. Words which are not in Hollerith are converted into the ASCII code for /20 and a "data fault" status is returned in the software status (ECB word 4: bit 13 is 1). There is no special code. However, EOS and EOF marks are detected (bits 14 and 15 in the software status).

BASIC WRITE (/05) (Standard I/O and CFM)

Operator's Typewriter:
All characters are output without checking or special features. This order can be used to print something and have the answer on the same line.

ASR Tape Punch:
All characters are output without checking or special features.

Line Printer:
All characters are output without checking. There is no control character.

High-speed Tape Punch:
All characters are output without checking or special features.

Cassette and Magnetic Tape:
See under Basic Read (/01).

STANDARD WRITE (/06) (Standard I/O and CFM)

Operator's Typewriter:
All characters, except /0 to /1F (special code characters), are output without checking. At the end of a line, a carriage return and line feed are output. The first word in the buffer contains a control character (second character), as for the line printer (see below). If it equals /30 or /31, it is output as line feed; if it is different, it is not output.

ASR Tape Punch:
Same features as for the keyboard. At the end of a line, the following character sequence is output: LF - Xoff - CR - Rubout

High-speed Tape Punch:
Same as for ASR tape punch.

Line Printer:
All characters are output without checking, except for the control code. It must be stored in the right-hand character of the first word of the buffer. This control code may have one of the following three values:

+ (/2B): print the line without advancing the paper (superposition).
0 (/30): advance two lines before printing.
1 (/31): skip to top of page before printing.

All other control codes are used as normally: advance one line and print. At the end of the buffer, after the requested length, one character must follow to be used by the system for a print code.
If the requested length is more than one line, the system puts a print code after the maximum length and the buffer will be printed on two or more lines.

Cassette and Magnetic Tape:
See under Basic Read (/01).

OBJECT WRITE 8+8 TAPE FORMAT (/08) (Standard I/O and CFM)

High-speed Tape Punch:

The standard object code is output in 8+8 format, where the first character is a format character and is output on one row, converted as follows:

/0 – /10
/1 to /4 – /01 to /04
/5 to /7 – /15 to /17

The second character contains the length in words, excluding the first word and checksum. An 8-bit checksum is performed and punched.

Cassette and Magnetic Tape:
See under Basic Read (/01).

GET TYPE OF LABELLING (/21)

CFM (Compact and Extended) only:

the CFM package returns in ECB3 the information on the type of labelling used on the cassette with the file code specified in ECB0, as follows:

bit 2 = 1: tape premarked
3 = 1: unknown type of labelling
4 = 1: basic labelling
5 = 1: compact labelling
6 = 1: extended labelling
7 = 1: working cassette with compact labelling
8 = 1: system cassette with compact labelling
9== 1: file open for read
10= 1: file open for write.

WRITE TAPE MARK/WRITE EOF (/22)

Standard I/O:

Operator's Typewriter:
An end-of-file mark is output as follows:
:EOF LF Xoff CR Rub-Out

ASR Tape Punch:
An end-of-file mark is output as for the typewriter.

High-Speed Tape Punch:
An end-of-file mark is output as for the typewriter.

Line Printer:
An end-of-file mark is output as :EOF

Cassette Tape:
An end-of-file mark is output as :EOF

CFM Package:

For basic labelling:
a tape mark is written
For compact and extended labelling:
an EOF label is written according to the specifications for these labelling systems.

If order /29 has previously been given, the CFM package will write the EOF label with the field identifiers given in the user's buffer label. The system will write correctly only:

- for compact: label identifier field
  volume identifier field
  file identifier field
- for extended: label identifier field
  label number field
  file identifier field

If the returned status is:
bit 0 = 1 and bit 1 = 1 and bit 8 = 1,
an error has been detected in the calling sequence: the file is not open.

WRITE FILE HEADER LABEL (/23)

CFM (Compact and Extended) only:

For this command the user program must give in the ECB the length of the label and the address of the block containing the layout of the file header label.
For compact labelling:

- length of the label is 32 characters.
- the CFM package writes label identifier field and volume identifier field. Any other fields will be filled according to the fields in the user's label of which the address is given in the ECB.

For extended labelling:

- length of the label is 80 characters
- the CFM package writes label identifier field and label number field. Any other fields will be filled according to the fields in the user's label of which the address is given in the ECB.

This order also opens a file.
After this order all other orders compatible with the labelling system used will be accepted except another order /23. This order will be accpted only after the now opened file has been closed with a 'Write EOF'.

If the returned status is:
bit 0 =1 and bit 1= 1 and bit 8 = 1,
an error has been detected in the calling sequence: the previous file has not been closed with an EOF or the header name has already been catalogued.

WRITE EOV (/24)

Standard I/O only:
End-of-tape management for magnetic and cassette tapes under the standard I/O package is a user program responsibility. When the physical end of a tape is encountered during a write operation, a status is returned in ECB4 with the EOT bit set. The user may then issue a Write EOV request before requesting the operator to mount a new tape. When a new tape is mounted, for magnetic tape the unit must first be switched off by pressing the OFF LINE button, while for cassette tape a Manual Control (MC) operator command 'Unlock' must be given to enable the operator to remove the cassette.
Then the operator can mount a new reel or cassette and restart the program. To ensure that all records will be retrieved when this file is read, the EOT status also returned in ECB4 should be ignored and only the EOV status must be taken into account.
Note: In case the EOT is detected when reading an EOV, only the EOV status is returned.

WRITE EOS (/26)

Standard I/O:
Operator's Typewriter:
An end-of-segment mark is output as follows:
:EOS LF Xoff CR Rub-Out

ASR Tape Punch:
An end-of-segment mark is output as for the typewriter.

High-Speed Tape Punch:
An end-of-segment mark is output as for the typewriter.

Line Printer:
An end-of-segment mark is output as :EOS

Magnetic Tape:
An end-of-segment mark is written as :EOS + 8 blank characters.

Cassette Tape:
An end-of-segment mark is written as :EOS

CFM Package:
An end-of-segment mark is written on the cassette tape as :EOS.

SEARCH FILE HEADER LABEL (/27)

CFM (Compact and Extended) only:
the CFM package searches the file header with the name specified in the user's buffer, of which the first 6 characters are accepted as the name.
If previously the order /29 has been given, the file header label will be written into the user's label buffer specified under that order.

After this order the system is ready for read operations on the file. Write operations are rejected, except for a last file for which an incorrect labelling status was returned, in which case the user may give a 'Write EOF' order to close it.

If the returned status in the ECB is:
bit 0 = 0 and bit 6 = 1:
an unknown file header label name has been specified.

ENABLE ACCESS FOR LABELS (/29)

CFM (Compact and Extended) only:
with this order the user program gives the address of a 'label buffer' into and from which the labels will be written and read. If the program operates in this mode, the CFM package

will fetch the data to be written in the different fields of the label from this buffer label. The layout of these fields must be according to the specifications for the labelling systems.
The buffer address must be given in ECB1, the length is 32 characters for compact labelling and 80 for extended.
The order is always accepted.
Into the label buffer are written the file header label and the EOF label.

INHIBIT ACCESS FOR LABELS (/2A)

CFM (Compact and Extended) only:

when this order is given, the CFM package will stop reading and writing labels from and into the label buffer.
The order is always accepted.

RETURN INFORMATION ABOUT A FILE CODE (/30):

Standard I/O Package only:

By means of this order it is possible to find out the assignment of a file code. The information will be returned in the Event Control Block:

ECB0: File Code

ECB1: Device Name (2 ASCII characters):

- TY = operator's typewriter
- TR = ASR tape reader
- TP = ASR tape punch
- PR = tape reader
- PP = tape punch
- LP = line printer
- CR = card reader
- MT = magnetic tape
- TK = cassette tape

ECB2: maximum record size

ECB3: left character: unused
right character: device address

ECB4: status = 0. For line printer, this word contains the number of lines specified for this printer at system generation time.

If the file code in ECB0 is set to zero, the other words of the ECB will also contain zeros. This means no device has been assigned to this file code.

REWIND TO LOAD POINT / REWIND FILE (31)

Standard I/O:

When this order is given , a tape is positioned at the beginning.

CFM Package:

- basic labelling:
  the tape is positioned after the first tape mark on the tape, at the very beginning.
- compact and extended labelling:
  the current file is rewound to the beginning of the file, i.e. after the tape mark which follows its file header. If the program operates in Enable Access for Labels mode, i.e. if previously the order /29 has been given, the file header label of the current file is written into the label buffer.

SEARCH BACKWARD FOR TAPE MARK / SEARCH FOR FIRST FILE (/36)

Standard I/O:

The tape is rewound until a tape mark is encountered. The tape is then positioned after this after this tape mark.

CFM Package:

- basic labelling:
  the tape is rewound until a tape mark is encountered. The tape is then positioned after this tape mark.
- compact and extended labelling:
  the tape is rewound until the first file header on the tape is found. The tape is then positioned after this header, i.e. the same position as when the tape is loaded.
  If an order /29 has previously been given, the first file header is written into the user's label buffer.

SEARCH FOR NEXT FILE HEADER LABEL (/37)

CFM (Compact and Extended) only:

the file header label of the next file on the tape is searched
If order /29 has previously been given, the label is written into the user's label buffer.
This order is always accepted.
When no next file header is found on the tape, the status returned in ECB4 is:
bit 0 = 0 and bit 9 = 1, i.e. end of file set, no data follow.

UNLOCK (/38)

Magnetic Tape:
This order switches the tape drive off-line.

Cassette Tape:
This order unlocks the cassette from the drive unit.

The object format as described below is the output of the Assembler, FORTRAN compiler and the non-disc Linkage Editor.
This type of object may also be the input of the Linkage Editor and of the non-disc loaders.
The object input or output consists of clusters, which are object code records of specific types and formats. Those records vary in length and are separated from each other by a gap (punched tape only).
Object modules are separated by an End Of Segment mark (:EOS LF XOFF CR) while the last module is followed by EOS EOF.

The object code is punched in two formats:

- 8 + 8 format where the information contained in 16-bit words, is punched over two rows of 8 bits. This format can be read from the high speed punched tape reader.
  The first character of each record is a record identification (X'10', X'1' to X'4', X'15' to X'17').
  The second character specifies the number of 16-bit words in the remainder of the record, the checksum excluded. This character is not punched.
  The remainder consists of 16-bit words punched over two rows. The last word of each record is the checksum and is followed by an XOFF character + Tape Off character.

- 4×4 format where the information contained in 16 bits is punched over four rows of four bits. This format can be read from the high speed punched tape reader and from the reader attached to the operator's typewriter.
  The first character of each record is a record identification (X'18' to X'1F')
  The second character specifies the number of 16-bit words in the remainder of the record, the checksum excluded.
  The remainder of the record consists of n 16-bit words punched over four rows.
  The last word of the record is a checksum and is followed by an XOFF character and Tape Off character.

The characters in object code records are modified in the following way:

- add X'10' to zero characters and to binary characters X'5' to X'7'.

Other characters remain unchanged.

Cluster identification

Cluster type	8+8 format	4+4+4+4 format
0	/10	+8 =/18
1	1	+8+/10 =/19
2	2	+8+/10 =/1A
3	3	+8+/10 =/1B
4	4	+8+/10 =/1C
5	/15	+8 =/1D
6	/16	+8 =/1E
7	/17	+8 =/1F

If the first character is ≠ /07 (Bell),/0A (Line Feed), /0D (CR), /11 (Xon), /13 (Tape Off)),/7F (Rub Out),/12 (Xon punch), /14 (Xoff punch) and <20 then it is object code.
Only the last four bits are kept (∧/F). If the value is <8 then the object code is of format 8+8 and if ≥8 then 4+4+4+4. To find the cluster type 8 must be subtracted.

In P852M Object Code there are 8 types of clusters, namely

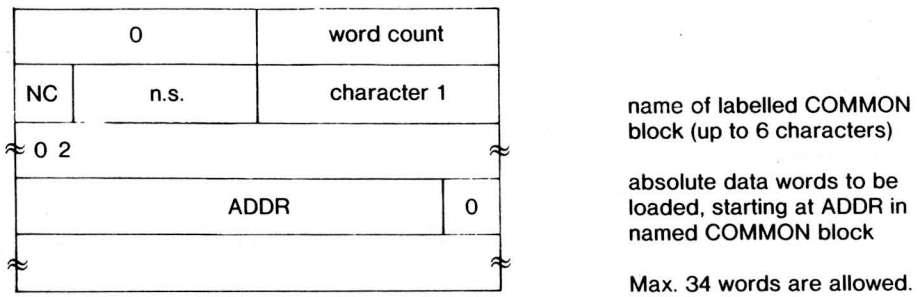
– BLOCK DATA CLUSTERS	type 0
– ENTRY POINT NAMES CLUSTER	type 1
– EXTERNAL REFERENCE NAME CLUSTER	type 2
– CODE CLUSTER	type 3
– INTERNAL MODIFICATION CLUSTER	type 4
– ENTRY POINT DEFINITION CLUSTER	type 5
– COMMON LENGTH DEFINITION CLUSTER	type 6
– END CLUSTER	type 7

The first record of any object module is a program identification record in ASCII format and consists of an IDENT statement which is the same as the source statement from which it originated, followed by LF XOFF CR.

The eight types of clusters do not have the same length.

BLOCK DATA CLUSTER (0)

This cluster is only generated by the FORTRAN compiler when a Block Data subroutine is processed. It permits the initialization of a COMMON block.



where:

– the first character of the first word indicates the type number of the cluster (0).
– the second character of the first word contains the number of words in the remainder of the cluster, excluding the checksum.
– bits 0, 1 and 2 of the second word contain the number of characters (max 6) specifying the name of the labeled COMMON block. Bits 3 to 7 are not relevant.
– ADDR, bit 15 = 0 indicates the relative address of the first data word to be loaded in the named COMMON block.

ENTRY POINT NAMES CLUSTER (1)

This cluster contains a list of entry point names. Entries in this cluster have no fixed length.
All clusters of this type are grouped and follow the IDENT.
Each entry point name is given a value in the ENTRY POINT DEFINITION CLUSTER.

1	word count
NC (=4) n.s.	N
0 2 3 7 A	M
0 E	0 0
NC (=1) n.s.	E
NC (=3) n.s.	E
0 2 3 7 N	T

where:

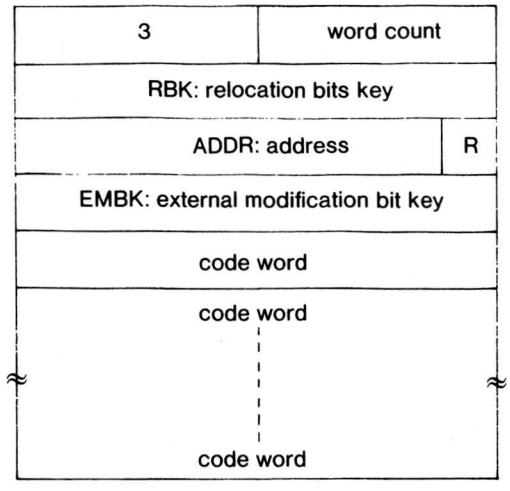
- the first character of the first word indicates the type number of this cluster
- the second character of the first word indicates the number of words in this cluster
- NC gives the number of characters in the name
  A maximum of six characters is allowed
- n.s. = not significant.

EXTERNAL REFERENCE NAME CLUSTER (2)

This cluster contains a list of external reference names. These names are referred to in the code cluster by a number; the first external reference name encountered in the module has received number 2, the second one number 4 etc. In the remainder of the object program the externals may be referred to by their number.
The format of this cluster is the same as for the type 1 cluster.

CODE CLUSTER (3)

This cluster contains the code words (e.g. instructions) generated by the Assembler and Linkage Editor and the addresses from which the code words have to be loaded into memory. Each cluster may contain a maximum of 16 code words.



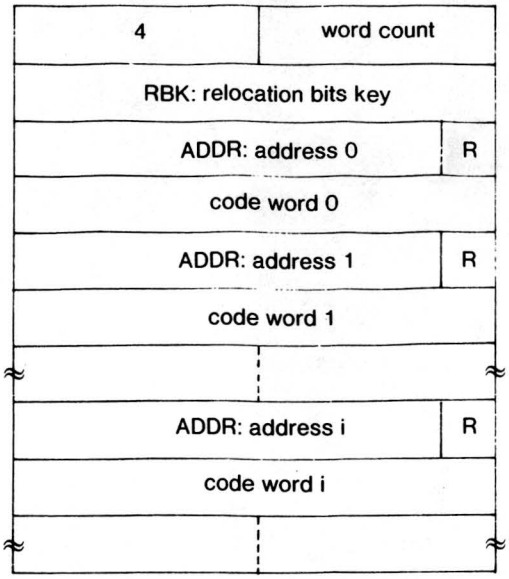
where:

- the first character of the first word indicates the type number of this cluster.
- the second character of the first wordt indicates the number of words in this cluster.
- RBK is a Relocation Bit Key. Bit 0 of this word is related to the first code word in this cluster, bit 1 to the second word etc.
  If a **code word** has its RBK bit set (to 1) this code word is relocatable i.e. at load time or link-load time the absolute load address of the object module will be added to the address of this word. At link edit time the relative address of the object module will be added to the address of this word
- ADDR contains the address from which the list of code words in this cluster must be loaded into memory.
  If bit 15 (R) = 0, ADDR is the absolute address of the first code word of this list (absolute program section). In that case the RBK bit is reset except when there is a reference from an absolute program section to a relocatable program section.
  If bit 15 = 1, ADDR is the relative address, i.e. the address of the first code word of the list relative to the load address of the module (relocatable programm section).
- EMBK is an External Modification Bit Key. As for RBK, bit 0 is related to the first code word, bit 1 to the second etc.
  If a code word has its EMBK bit set the next word in the list is not a code word but a word that contains the number of external reference.

  The purpose hereof is that the previous code word has to be modified by the Linkage Editor by adding the numerical equivalent of the external reference to the code word.

INTERNAL MODIFICATION CLUSTER (4)

Code words generated by the Assembler and Linkage Editor are listed in this cluster each time a forward reference is satisfied.
Each code word in this cluster is associated with a relocation bit of a RBK as described in cluster type 3, and an absolute or relocatable word address.
Bit 0 of RBK is related to code word 0, bit 1 to code word 1 etc.

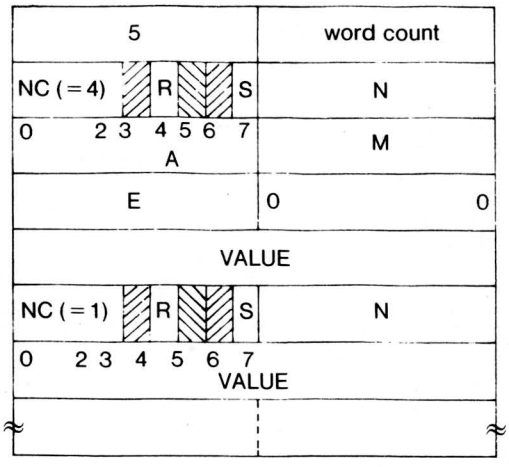


where:

- the first character of the first word indicates the typenumber.
- the second character of the first word indicates the number of words in this cluster, checksum excluded.
- R, if 0 the address is absolute
  if 1 the address is relocatable

ENTRY POINT DEFINITION CLUSTER (5)

This cluster contains a list of entry point names and the number they have been given for reference purposes.



where:

- NC is the number of characters in the entry point
- R if 1, the entry point name is relocatable
  if 0, the name is absolute
- S if 1, the name is the name of an internal symbol table
  if 0, if not
- NAME the name of the entry point
- VALUE value of the entry point

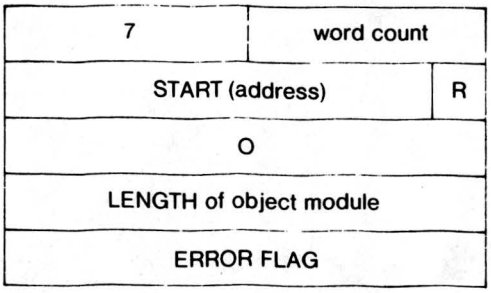
COMMON LENGTH DEFINITION CLUSTER (6)

This cluster is output by the Assembler of 8k and up and by the FORTRAN compiler.
Those two can make use of a common block in which memory locations are shared between subprogram and main program variables.
References to a common area are considered to be external references to that common with a displacement value representing a relative address in the common.
Since the Linkage Editor has to know the length of the common blocks the length of all blocks are given in this cluster.

The format of this cluster is the same as for the Entry Point Definition cluster, except that R and S are not significant, and

NAME = the name of the common block. If it is a blank common the name is one blank.
VALUE = length in characters of the common block.

END CLUSTER (7)



This cluster is always the last cluster of a module. Its format is as follows:

where:

- the first character of the first word indicates the typenumber of this cluster
- the second character indicates the number of words in this cluster, checksum excluded.
- START is the start address of the object module. If R (bit 15) = 0 the address is absolute, if 1 it is the relative starting address. If START and R are 0, there is no start address.
- LENGTH contains the length of the object module given as an even number of characters.
- ERFLAG is a flag indicating whether any errors have occurred during the assembly or link-editing of the module.
  If so, the number of errors is given in binary. If no errors have occurred this word is 0.

Appendix D File codes and device names

File codes enable files to be identified and addressed. The file codes are assigned to devices at SYSGEN. The device addresses are fixed by hardware. The size of the file code table is also established at SYSGEN so that dummy file codes should be included if future expansion is envisaged.

Standard File Codes	**File**
01	Source input
02	Listing output
03	Punch output
04	Object input
05	Operator's typewriter (input and output)

Device Names	**Device**
TR	ASR tape reader
TP	ASR tape punch
PR	High speed tape reader
PP	High speed tape punch
TY	Operator's typewriter
CR	Card reader
LP	Line printer
TK	Magnetic tape cassette
MT	Magnetic tape
NO	No device: an operation on this file will have no effect.

Appendix E Control Unit Status Word Configuration

This is the hardware status as returned in ECB word 4. See I/O monitor request.

Bit	Description	Control Unit						
		ASR	CR	LP	PTP	PTR	TK	MT
0								
1	has become ready						×	×
2	rewinding							×
3	tape mark read						×	×
4	no data						X	
5	Load point beginning of tape						X	X
6	write unable						×	×
7	A or B side (A=1, B=0)						×	
8	device address						×	×
9	device address						×	×
10	EOT tape low				×	×	×	×
11	program error						×	×
12	incorrect length		×				×	×
13	parity error						×	×
14	throughput error	×	×			×	×	×
15	not operable (only significant bit for TST)	×	×	×	×	×	×	×

```
00000                                   IDENT    BEBOOT
00001                           *
00002                           *
00003                           *       DISPLAY THE KEYS AS FOLLOWS:
00004                           *
00005                           *       BITS     MEANING
00006                           *       0=1      IPL LOADED FROM ASR , FORMAT 4*4
00007                           *       1=1      DISK,THEN
00008                           *       2=1      MOVING HEADS
00009                           *       2=0      FIXED HEADS
00010                           *       3=1      PROGRAMMED CHANNEL
00011                           *       3=0      I/O PROCESSOR
00012                           *       4 TO 7   BOU LINES ( 4 RIGHTMOST BITS )
00013                           *       8=1      MULTI DEVICE CONTROLLER
00014                           *       8=0      SINGLE DEVICE CONTROLLER
00015                           *       9=1      X1215 DISK
00016                           *       10 TO 15 DEVICE ADDRESS
00017                           *
00018                           *
00019                           *
00020                           *       DESCRIPTION
00021                           *
00022                           *       BEBOOT LOADS ONE RECORD ONTO LOCATION /80 THEN START AT /84
00023                           *       THE RECORD IS THE SECTOR # 1 IF DISK,OR 254 CHARACTERS OF THE
00024                           *       INPUT DEVICE,LEADING NULL CHARACTERS IGNORED
00025                           *
00026                           *
00027                           *
00028                           *       USED REGISTERS :
00029                           *
00030                           *       A1 BOU LINES,NOT TO BE DESTROYED IF BOOT IS CALLED AGAIN
00031                           *       A2 ADDR OF INR INSTRUCTION
00032                           *       A3 ADDR OF CIO INSTRUCTION(WHICH IS DESTROYED AND NEEDS TO BE
00033                           *                                  RESTORED IF BOOT IS CALLED AGAIN)
00034                           *       A4 ADDR OF SST INSTRUCTION
00035                           *       A5 MULTIPLEX : CONTENTS OF 1ST WORD TO BE SENT TO EXT REGISTER
00036                           *          IO BUS : CHARACTER COUNT,INITIALIZED AT 254 AND DECREMENTED
00037                           *       A6 MULTIPLEX : CONTENTS OF 2ND WORD TO BE SENT TO EXT REGISTER
00038                           *                                  (LOADING ADDR)
00039                           *          IO BUS : ADDR OF NEXT CHAR TO BE LOADED,INIT AT /80 AND
00040                           *                                  INCREMENTED
00041                           *       A7,A8 WORK REGISTERS
00042                           *       A9 WORK REGISTER
00043                           *       A10 TO A14 NOT USED
00044                           *       A15 CONTAINS THE KEYS'VALUE
00045                           *
00046                           *
00047                           *
00048                           *
00049                           *
00050                           *
00051                           *
00052                                   EJECT
00053                                   AORG     0
00054                           *
00055                           *
00056                           BOOT    EQU      *
00057                           *       INITIALIZE REGISTERS
00058  0000  0200  F                    LDK      A2,INR       ADDR OF INR INSTRUCTION
00059  0002  0300  F                    LDK      A3,CIO       ADDR OF CIO INSTRUCTION
00060  0004  0400  F                    LDK      A4,SST       ADDR OF SST INSTRUCTION
00061                           *                             EXTRACT DEVICE ADDR AND INIT I/O COMMANDS
00062  0006  861E                       LDR      A6,A15
00063  0008  263F                       ANK      A6,/3F       DEVICE ADDR
00064  000A  9629                       ADRS     A6,A2        INITIALIZE I/O INSTRUCTIONS
00065  000C  962D                       ADRS     A6,A3
00066  000E  9641                       ADS      A6,HIO
       0010  0000  F
00067                           *                             EXTRACT CONTROLLER ADDR AND INIT WER INST
00068  0012  871E                       LDR      A7,A15
```

```
00069  0014  3FC8          SLC     A7,8        MULTI OR SINGLE DEVICE CONTROLLER
00070  0016  5600  F       RF(6)   INIT20      SINGLE ONE
00071  0018  260F          ANK     A6,/F       MULTIPLE ONE
00072                INIT20 EQU     *           INITIALIZE MULTIPLEX DBLE WORD:
00073  001A  9631          ADRS    A6,A4       SST INSTRUCTION
00074  001C  3E41          SLL     A6,1
00075  001E  9641          ADS     A6,WER1     SET UP WER INSTRUCTIONS
       0020  0000  F
00076  0022  9641          ADS     A6,WER2
       0024  0000  F
00077                      *                   LOAD A1 WITH BOU CONTENTS
00078  0026  811C          LDR     A1,A7
00079  0028  0550          LDK     A5,80       MULTIPLEX DOUBLEWORD: LOAD 80 CHAR INTO
00080                      *                   LOCATION /80
00081  002A  0680          LDK     A6,/80
00082                      *                   CHECK IF DISK
00083  002C  3FE7          SRC     A7,7
00084  002E  5600  F       RF(6)   NODISK      NO
00085                      *                   FIXED HEADS ?
00086  0030  3FC1          SLC     A7,1
00087  0032  5600  F       RF(6)   NOSEEK      YES
00088  0034  0103          LDK     A1,3        SEEK ZERO
00089  0036  41C0    CIO   CIO     A1,1,0      CIO SEEK ZERO
00090                NOSEEK EQU     *
00091  0038  811E          LDR     A1,A15
00092  003A  3966          SRL     A1,6        SECTOR NUMBER
00093  003C  213C          ANK     A1,/3C
00094  003E  8520          LDKL    A5,/80CD    1ST WORD OF MULTIPLEX FOR DISK DEVICE
       0040  80CD
00095                NODISK EQU     *
00096                      *                   EXECUTE WER,WHATEVER THE CHANNEL IS
00097                WER1  EQU     *
00098  0042  7500          WER     A5,0
00099                WER2  EQU     *
00100  0044  7601          WER     A6,1
00101  0046  F031          EXR*    A4          SST
00102  0048  F02D          EXR*    A3
00103  004A  5C06          RB(4)   *-4
00104  004C  871E          LDR     A7,A15
00105  004E  3F43          SLL     A7,3
00106  0050  5600  F       RF(6)   SST         MULTIPLEX
00107                      *
00108                      *                   IO BUS
00109                      *
00110  0052  8194          LDR     A9,A5       IF A9=A5 IGNORE LEADING CHAR,
00111                INR   EQU     *
00112  0054  4F00          INR     A7,0,0      READ ONE CHAR
00113  0056  5C04          RB(4)   *-2
00114  0058  E994          CWR     A9,A5       LEADING CHAR?
00115  005A  5400  F       RF(4)   INR10       NO
00116  005C  27FF          ANK     A7,/FF      CHECK IF NULL
00117  005E  580C          RB(0)   INR         YES,IGNORE
00118                      *                   NO,CHECK IF 4*4
00119                      *
00120                INR10 EQU     *
00121  0060  879E          LDR     A15,A15     4*4
00122  0062  5600  F       RF(6)   STORE       NO 8*8
00123  0064  3F44          SLL     A7,4        YES,4*4
00124  0066  809C          LDR     A8,A7       SAVE LEFT BITS
00125  0068  F029          EXR*    A2          READ NEXT CHARACTER
00126  006A  5C04          RB(4)   *-2
00127  006C  270F          ANK     A7,/F       GET 4 RIGHT MOST BITS
00128  006E  9702          ADR     A7,A8
00129                STORE EQU     *
00130  0070  E739          SCR     A7,A6       STORE CHAR
00131  0072  1601          ADK     A6,1        NEXT CHAR ADDR
00132  0074  1D01          SUK     A5,1        COUNT DONE ?
00133  0076  5924          RB(1)   INR         NO
00134                      *                   YES
00135                      *
00136  0078  4180          HIO     CIO     A1,0,0
00137                      *
00138                STATUS EQU     *
00139  007A  4FC0          SST     SST     A7,0
00140  007C  5C04          RB(4)   *-2
00141  007E  0F84          AB      /84
00142                      *
00143                      *
00144                      *
00145                      END     BOOT
```

```
          SYMBOL TABLE

  BOOT    0000  A   INR     0054  A   CIO     0036  A   SST     007A  A
  HIO     0078  A   INIT20  001A  A   WER1    0042  A   WER2    0044  A
  NODISK  0042  A   NOSEEK  0038  A   INR10   0060  A   STORE   0070  A
  STATUS  007A  A

          ASS,ERR,    00000
;EOF
PROG ELAPSED TIME:  00H-00M-00S-000MS-
BEA IPL52S
DATE     /   /     TIME 24H-60M-60S-
```

char.	ASCII octal	Intern Hexa	char. set punch comb.	char.	ASCII octal	Intern Hexa	char. set punch comb.
space	240	20	on punch	D	304	44	12,4
!	241	21	11,8,2	E	305	45	12,5
"	242	22	8,7	F	306	46	12,6
#	243	23	8,3	G	307	47	12,7
$	244	24	11,8,3	H	310	48	12,8
%	245	25	0,8,4	I	311	49	12,9
&.	246	26	12	J	312	4A	11,1
'	247	27	8,5	K	313	4B	11,2
(	250	28	12,8,5	L	314	4C	11,3
)	251	29	11,8,5	M	315	4D	11,4
*	252	2A	11,8,4	N	316	4E	11,5
+	253	2B	12,8,6	O	317	4F	11,6
,	254	2C	0,8,3	P	320	50	11,7
-	255	2D	11	Q	321	51	11,8
.	256	2E	12,8,3	R	322	52	11,9
/	257	2F	0,1	S	323	53	0,2
0	260	30	0	T	324	54	0,3
1	261	31	1	U	325	55	0,4
2	262	32	2	V	326	56	0,5
3	263	33	3	W	327	57	0,6
4	264	34	4	X	330	58	0,7
5	265	35	5	Y	331	59	0,8
6	266	36	6	Z	332	5A	0,9
7	267	37	7	[	333	5B	
8	270	38	8	\	334	5C	
9	271	39	9	]	335	5D	
:	272	3A	8,2	↑	336	5E	
;	273	3B	11,8,6	←	337	5F	
<	274	3C	12,8,4				
=	275	3D	8,6	Bell	207	07	
>	276	3E	0,8,6	Linefeed	212	0A	
?	277	3F	0,8,7	Car.Ret.	215	0D	
@	300	40	8,4	X on reader	221	11	
A	301	41	12,1	X off reader	223	13	
B	302	42	12,2	Rubout	377	7F	
C	303	43	12,3	X on punch	222	12	
				X off punch	224	14	
				FF		0C	

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X

Comment Sheet

P800M Programmer's Guide 1, Volume VI : COS (5122 991 27361)

Name _____

Company _____

Department _____

Address _____

_____

Telephone Number _____ ext. _____

Comments or Suggestions:

PHILIPS DATA SYSTEMS B.V.

MARKETING GROUP SMALL COMPUTERS
P.O. Box 245, Apeldoorn, The Netherlands
Phone: 055-230123; telex: 49142

For further details contact the above address or:

EUROPE

Sweden
Svenska AB Philips
Data Systems Division
Lindingövägen 50
Fack
10250 Stockholm 27
Tel. 08 635000

Denmark
Philips Data Systems A/S
Prags Boulevard 80
2300 København S
Tel. 0127 2222

Norway
Norsk A/S Philips
Data Systems Division
Nils Hansens vei 2
P.O. Box 5040
Oslo 6
Tel. 02 679380

Finland
OY Philips AB
Department Data Systems
Kaivokatu 8
P.O. Box 10255
Helsinki 10
Tel. 90 17271

Belgium
Philips Data Systems SA
Marketing Group
Small Computers
Anspachlaan 1
1000 Brussel
Tel. 02 2193900

France
Philips Data Systems
Département Mini-ordinateurs
5 Square Max-Hymans
75015 Paris 15
Tel. 01 734 7759

Western Germany
Philips GmbH-Eiserfeld
Bereich Prozessrechner
Münsterstrasse 330
4 Düsseldorf 30
Tel. 0211 631064

Höhenstrasse 17
7012 Fellbach bei Stuttgart
Tel. 0711 523081

Austria
Österreichische Philips GmbH
Industrie Elektronik
Breitenfurterstrasse 219
1230 Wien
Tel. 0222 831501

Italy
Philips s.p.a.
Sezione P.I.T.
Via Elvezia 2
20052 Monza
Tel. 04 361441

Switzerland
Philips AG Data Systems
Binzstrasse 18
8027 Zürich
Tel. 01 442211

Great Britain
Philips Data Systems
Elektra House
2 Bergholt Road
Colchester
C04-5AA Essex
Tel. 206 5115

The Netherlands
Philips Nederland B.V.
Professionele
Produkten en Systemen
Boschdijk 525
Eindhoven
Tel. 040 782953
788325

Spain
Philips Ibérica S.A.E.
Grupo Instrumentación
Martinez Villergas 2
Madrid 27
Tel. 091 4042200

FAR EAST

Japan
Nihon Philips Corporation
P.O. Box 13
World Trade Centre
Hamamatsu-cho, Minato-ku
Tokyo 105
Tel. 03 435 5211

NORTH AMERICA

U.S.A.
North American Philips Corp.
Dept. 007
100 East 42nd Street
New York N.Y. 10017
Tel. 212 697 3600

Canada
Philips Electronics
Industries Ltd.
Telecommunication Division
1001, Ellesmere Road
Scarborough 706
Ontario
Tel. 416 7521980