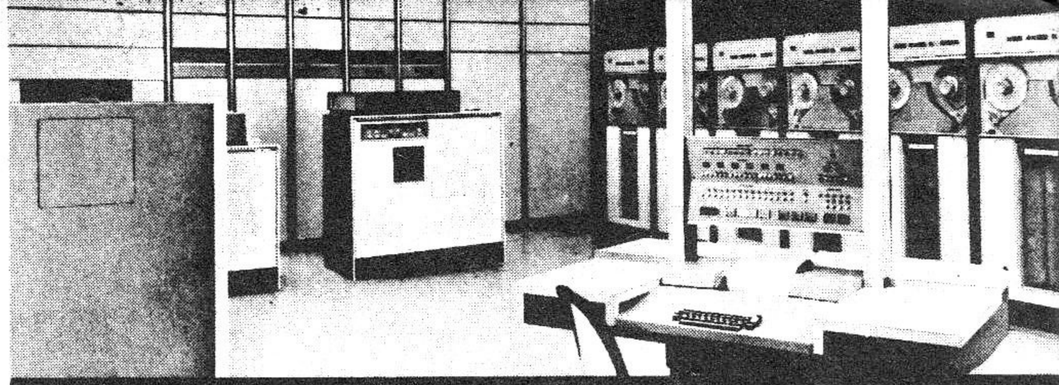


UNIVAC® 1107

GENERAL MANUAL

EXEC II

PROGRAMMER'S GUIDE

U-3671

This manual is published by the UNIVAC® Division in loose leaf format as a rapid and complete means of keeping recipients apprised of UNIVAC Systems developments. The UNIVAC Division will issue updating packages, utilizing primarily a page-for-page or unit replacement technique. Such issuance will provide notification of hardware and/or software changes and refinements. The UNIVAC Division reserves the right to make such additions, corrections, and/or deletions as, in the judgment of the UNIVAC Division, are required by the development of its respective Systems.

PRINTED IN U.S.A.

PREFACE

This manual is intended as a programmer's reference manual on the EXEC II System and its supporting routines. FORTRAN, COBOL, and SLEUTH II for the UNIVAC 1107 are described in other manuals, and only information which is pertinent to the relation between the Monitor and these programs is included herein. A knowledge of the UNIVAC 1107 is assumed.

Because of the dynamic requirement for timely information about the system, interim revisions and additions will be made to the manual from time to time.

UNIVAC 1107 Thin-Film Memory System
EXEC II General Manual

March 4, 1964

UPDATING PACKAGE "A"

The attached 65 pages contain revisions to "EXEC II General Manual," UNIVAC 1107 Thin-Film Memory System, U-3671.

This updating package should be utilized in the following manner:

SECTION	DESTROY FORMER PAGES NUMBERED	REPLACE WITH NEW PAGES NUMBERED*
I	i and ii	i and ii
III	3-1 thru 3-17	3-1 thru 3-26
IV	4-1 and 4-2 4-9 and 4-10	4-1 and 4-2 4-9 and 4-10
V	5-11 and 5-12 5-15 and 5-16 5-21 thru 5-24 5-29 thru 5-36 5-39 and 5-40	5-11 and 5-12 5-15 and 5-16 5-21 thru 5-24 5-29 thru 5-36 5-39 and 5-40
VIII	8-1 and 8-2 8-9 thru 8-16 n.a. (new)	8-1 and 8-2 8-9 thru 8-16 8-19 thru 8-23

* Certain pages have been reprinted without change; they are "backs" or "fronts" of pages containing changes. Such pages do not have a new revision number assigned to them. All revised pages carry the identification of Rev.1.

August 14, 1964

UPDATING PACKAGE "B"

The attached page contains an addition to "EXEC II General Manual," UNIVAC 1107 Thin-Film Memory System, U-3671.

This updating package should be utilized in the following manner:

SECTION	FILE NEW PAGE NUMBERED
V	5-10A

Page 5-10A of Section V is released because the information contained in this page was erroneously omitted during the printing of Updating Package "A" to the manual, U-3671. Since this information logically falls between pages 5-10 and 5-11 of Section V, an exception to the normal page numbering sequence is forced. Hence the letter A is added with this new page 5-10. This does not mean that page 5-10 presently in your manual is to be destroyed. This section will be placed in proper page numbering sequence with the issue of the next updating package.

II. INTRODUCTION

2.1 GENERAL

This manual describes, from the viewpoint of the user, the 1107 EXEC II System. It is necessarily incomplete and subject to minor changes.

2.2 DESIGN CRITERIA

The 1107 EXEC II System has been designed to encompass several differing functions and tasks, as described in subsequent paragraphs.

"JOB SHOP" CAPABILITY

The EXECII must be capable of running efficiently (from both the point of view of the machine itself and of use by the programer) a series of relatively small "scientific" programs. Machine efficiency entails good use of core space and I/O capability, minimal manual intervention either between or within jobs, and little opportunity for disastrous error on the part of the operating staff. User efficiency entails ease of specifying patches and diagnostic requests, effectiveness of listings and diagnostic output, and, above all, minimal turn-around time.

"LARGE PROGRAM" CAPABILITY

The process of construction and checkout of large routines, say 10,000 instructions and up, takes on different aspects than does that of small routines. The difference is characterized by requirements of joining together the results of many relatively independent programing efforts separated by time, distance, or personality. These large-routine problems become particularly acute for 100,000 instructions or more. Checkout facilities for the various pieces by themselves, as well as in conjunction with their fellows, must be provided. In addition, there must be methods for the control of highly segmented routines, for minimizing the propagation of changes at one part of the routine throughout the remainder of the routine, and for minimizing communication problems. Since the various pieces of a large routine may be quite different in their characteristics and in their efficiency requirements (that is, frequency of use), it is important to provide the widest possible range of programing languages; different

languages may then be used for different pieces. The EXEC II System must solve any interference problems when interconnecting the resulting compiled programs.

"ACCOUNTING" CAPABILITY

Since a computer must "work for a living", an accurate and equitable system for keeping records of machine time usage should be available.

"PERIPHERAL" CAPABILITY

Since the 1107 will usually be required to operate without benefit of offline peripheral equipment, the EXEC II System must provide, through a multiprograming facility, the efficient use of card and printing equipment without penalizing the routine being executed.

"SELF-DEFENSE" CAPABILITY

Particularly since peripheral equipment multiprograming is to be a part of the EXEC II System, it must, insofar as possible, protect itself from destruction due to program error.

2.3 GENERAL ORGANIZATION

The 1107 EXEC II System has three relatively distinct parts. The first is a set of input/output routines to be used by either the "user's program" or by the multiprogramed peripheral operations routines (called parasites). These routines are built around an efficient channel dispatching program designed to keep as much I/O activity going as the central computer can tolerate.

The second part is an information storage and retrieval scheme which facilitates manipulation and construction of programs from their component elements. The SLEUTH II assembler, the FORTRAN and COBOL compilers, and various other processors are integrated into this general program manipulation technique. The implementation methods used provide a completely open-ended organization. Features may be added at will to accomplish any ends that a particular 1107 installation might have.

The third part is an integrated system of diagnostic routines designed to return to the programer information of maximum utility and convenience in the checking out of his program. To this end, the programer can be highly selective in what is to be dumped and may receive his dump listings with the symbolics used by the processors intercollated with the contents of the machine memory.

From the foregoing, the reader will have gathered some understanding of the modular approach that has been taken in the design of the EXEC II System. It may be supposed that a continuing effort will be made on the system, thus extending its capabilities and sophistication.

Within the frame of reference described above the EXEC II System is organized along the usual lines with which most programers of large-scale equipment are already familiar.

Jobs to be processed by the EXEC II System are assumed to be input through the card reader. They may be stacked one after another; the system takes care that a job will not interfere with its successor. In the simplest case, the input to the system consists of a RUN control card, a program deck, and source data cards. Various other control cards are used to punctuate this input. In general, a single run can construct programs from one or a combination of calls to source language processors (e.g. FORTRAN), previously compiled subprograms, and library retrievals; execute these programs (with data input cards if required); and produce diagnostic output for debugging purposes.

After completion of one job, the system will commence with the next, normally without operator intervention.

2.4 MACHINE CONFIGURATION

The machine configuration for the 1107 EXEC II System must include a 32,768-word memory, one magnetic drum, a card reader and punch, and a high speed printer.

A system uses a minimum of 4,096 words while the user's program is in core. To this must be added 750-1000 words for each symbiont that is to be active at any one time. The maximum resident system with all appurtenances requires 8,192 words.

The EXEC II System resides on drum and uses drum space in the course of its operation. There are several distinct drum areas, only one of which has a minimum size. A particular installation may vary the size of these drum areas to suit its own needs, habits, and available hardware. The areas are:

a) Basic executive routines, the processors, and the library, occupying about 185,000 words. (If COBOL is omitted, 122,000 will suffice.)

b) Storage for the absolute program to be run, the symbolic information about it provided by the allocator and the processors, and unformatted diagnostic information collected during and after running. The space required depends on the anticipated maximum size of programs.

c) Storage for the program complex file and working space for the processors (not required during the execution of a program).

d) Storage for peripheral backup. Not required if magnetic tape backup is being used. As a rough rule, each half drum available will provide 35 minutes of backup per device used.

When deciding whether tape or drum symbionts should be used at a particular installation, it should be noted that a 300,000-word area on a drum will provide 20 minutes of backup and will replace four tape units.

2.5 SYMBOLISM AND CONVENTIONS

A few comments are required concerning the symbolism used in this manual.

1. All references to machine instructions are in terms of the SLEUTH II mnemonics.
2. Thin-film registers are referred to by either their absolute addresses or by the symbols A0, B11, R3, etc. There is no requirement that the user employ these particular symbols. (Many do not.)
3. When it is necessary to refer to particular bits or characters of a word, they are numbered from left to right across the word, beginning with zero.
4. In the description of calling sequences, all punctuation and capital letters represent themselves. Lower-case and Greek letters are used for fields to be supplied by the user. Such a field is surrounded by " marks when referred to in the text.

The term resident is used to denote a part of the executive which resides in core memory at all times. Examples of resident routines are the dispatcher, the real-time clock routine, and the internal error processing routines. Non-resident routines are taken from the library.

Externally defined symbols, library procedures, and communication cells within the system routine -- resident or not -- are given symbols which contain the symbol "$". (The $ may appear in a label in Sleuth II in the same manner as may a digit.) Symbols referencing the executive are further subdivided by their first character. This first character indicates the portion of the system with which the symbol is associated. A reference to an externally defined symbol or communication cell always has its "$" as the last character; procedure names will have the "$" as their second character. Whenever a procedure is available to generate a linkage to a subroutine (most subroutines are equipped with such procedures), the symbol used for the subroutine and for the procedure will differ only in the placement of the "$".

Various other placements of the "$" are used by the system to denote cross-communications between subroutines. They are not normally of interest to the programer. While it is legal for a programer to employ the "$" in his own symbols, such a practice can lead to trouble and is strongly discouraged. We list here a table of first characters currently assigned. This list will no doubt be extended as further features are added to the system.

A	System Environment Definition	N	FORTRAN communications
B	Block Buffering Package	O	
C	Card routines	P	Printer routines
D	Drum input/output package	Q	Paper tape routines
E	Editing routines	R	
F		S	Sleuth II communications
G		T	Tape input/output package
H		U	
I	Label and item handling routines	V	
J		W	
K	Console system	X	Diagnostic system
L	COBOL communications	Y	
M	Monitor entry points	Z	Symbiont control system

All subroutines discussed in this manual, unless very explicitly stated to the contrary, assume that registers B11, A0 thru A5, and R1 thru R3 need not be restored. Linkage to a subroutine is generally by means of a load modifier and jump referencing register B11. It should be noted here that film registers 28 through 31 may not be restored by FORTRAN-generated subroutines.

The EXEC II System reserves for its own usage the following components of the machine.

a. Film registers 0, 32 through 64, and 80 through 87.
b. A portion of core (variable depending on the features used).
c. A portion of drum (variable depending on the features used).
d. At least one card channel and one printer channel.
e. The console.

Some installations may wish to reserve further input/output equipment for the executive. For example, the paper tape punch may be taken for the output of accounting information.

In addition, the EXEC II System reserves the following commands for its exclusive use:

W	7200	Wait
LL	7211	Load Lockout Register
PAIJ	7213	Prevent All Interrupts and Jump
HKJ	7405	Halt on Keys and Jump
HJ	7405	Halt and Jump
AAIJ	7407	Allow All Interrupts and Jump
LIC	7500	Load Input Channel
LICM	7501	Load Input Channel, Monitored
JIC	7502	Jump Input Channel Busy
DIC	7503	Disconnect Input Channel
LOC	7504	Load Output Channel
LOCM	7505	Load Output Channel, Monitored
JOC	7506	Jump Output Channel Busy
DOC	7507	Disconnect Output Channel
LFC	7510	Load Function Channel
LFCM	7511	Load Function Channel, Monitored
JFC	7512	Jump Function Channel Busy
AFC	7513	Allow Function in Channel
AACI	7514	Allow All Channel Interrupts
PACI	7515	Prevent All Channel Interrupts
ACI	7516	Allow Channel Interrupt
PCI	7517	Prevent Channel Interrupt

III. BASIC SYSTEM FEATURES

3.1 CONTROL CARDS

3.1.1 GENERAL

The primary control exercised over the EXEC II System is by means of control cards. These control cards are used to direct the construction and execution of programs and to provide any necessary communication between the program and its environment (operator messages, magnetic tape assignment and post-mortem dumps). There are 16 control cards which fall into form categories:

a. System Control (RUN, ASG, MSG, PMD, ELT)

b. Processor Control (ASM, FOR, COB, MAP, ALG, PDP, CLP)

c. Allocator Control (XQT, ABS, SCD)

d. Card Input Control (EOF, FIN)

Detailed specification as to the form and content of these control cards is made throughout this manual. This section describes them in general and gives a summary in the form of a table. The RUN and MSG control cards are explicitly described.

3.1.2 FORMAT OF CONTROL CARDS

A control card is identified by the occurrence of a master space character in column one of the card. (The master space character is represented in this manual by the mark "▽".) Some of the control cards allow the specification of options by means of a string of letters. If any of the options are desired, the corresponding letters should be punched beginning in column two of the control card. Their order is irrelevant. The string of option letters is terminated by a blank column. If no options are present column two must be blank. The control card mnemonic appears next on the card. It consists of three letters and must be one of the set of 16 described by this manual. A string of one or more blanks

must precede the mnemonic. In some cases the mnemonic may be followed by a comma and then a single character. If this is so, a blank must follow that character; otherwise, a blank must follow the mnemonic.

The remainder of the control card contains specifications for the system routine involved. They are again preceded by a string of one or more blanks. The syntax of the specifications portion varies considerably from one control card to another and thus is not amenable to the general discussion here.

In summary, a control card has the form

∇options xxx, λ specifications

where "options" represents the option letter string, "xxx" the control card mnemonic, "λ" a single character, and "specifications" the specification portion.

The two control cards used by the card input system are interpreted at interrupt time, rendering the free form described above impractical. These two cards must be of the form:

Column 1	∇
Column 2	blank
Columns 3, 4, 5	EOF or FIN
Column 6	blank

If a malformed control card is encountered the card itself is printed, followed by the message:

ABOVE CONTROL CARD IN ERROR - IGNORED

In the case of premature termination of a run due to an error, any control card other than a PMD (LIBRARY II, U3672) will produce the message:

REMAINING CONTROL CARDS IGNORED

If the system is expecting a control card but encounters one or more cards not containing a "∇" in column one, the situation is signaled by the message:

DATA CARDS ENCOUNTERED BY SYSTEM - IGNORED

3.1.3 THE RUN CONTROL CARD

Each job to be performed by the system must be prefaced by a RUN control card. If the system is expecting a run card and some other control card is encountered, the message

RUN CARD MISSING - DECK NOT ACCEPTED

is printed out. The general form of the RUN control card is

∇ λ RUN identification, account, running time, output

The option letter "λ" serves as a run priority code when the card-drum parasite is being used for input. Those runs with option letters nearest the beginning of the alphabet will be executed first. Selection is made from those runs currently in the system but not yet executed. If the option letter is omitted, a priority code of "D" is assumed. Within a priority letter group the runs are executed in the order that they were introduced into the system. The same rules govern the selection of files of printed output to be sent to the printer(s).

The field "identification" should contain the installation's conventional run identification. Both it and the "account" field may consist of up to six characters taken from the set A ... Z, 1 ... 9, +, =, ·, $. Blanks are illegal. The field "running time" is an estimation in minutes of the running time required for the problem. Should this time be exceeded, the operator will be informed and the run continued. It is the operator's responsibility to terminate the run. Similarly, the operator will be informed of the maximum number of "output" pages. The "running time" field may be omitted in which case the system assumes a maximum time of five minutes; likewise the "output" field may be omitted and a maximum of 50 pages will be assumed.

	Mnemonic	Application	Options	Reference
System Control	RUN	Used to initiate each computer run.	λ(Priority)	2.1.3
	ASG	Used to cause assignment of magnetic tapes	HLOER	3.2.3 ~~2.2.3~~
	MSG	Used to type message to operator	HN	2.1.4
	PMD	Used to cause memory printout after execution	EIDAXC	8.2
	ELT	Used to introduce an element into the program complex file from cards	None	5.6.1
Card Control	EOF	Used to punctuate a data check	None	2.1.1
	FIN	Used to mark the end of a card stream. Not normally required by programmer	None	2.1.1
Processor Control	ASM	Used to call out Assembler	AXLNPSZI	5.5
	FOR	Used to call the FORTRAN Compiler	AXLNPSZI	5.5
	COB	Used to call out the COBOL Compiler	AXLNPSZI	5.5
	MAP	Used to call out the memory allocation processor	AXNPSI	5.5
	PDP	Used to call out the procedure definition processor	AXLIS	7.1.2
	CLP	Used to call out the COBOL library processor	AXLIS	7.1.2
Allocator Control	XQT	Used to execute a program (including allocation, if required)	AXLNZC	5.8.1
	ABS	Used to produce an absolute program	AXLNPZC	5.8.2
	SCD	Used to define a subcomplex	AXLNPC	5.8.3

CONTROL CARD SUMMARY

The "specifications" portion of the RUN control card is always copied to the console printer. The operator may elect to place the system in the "stop between jobs" mode. In this case console action must occur before the run can continue past the RUN card.

3.1.4 THE MSG CONTROL CARD

The MSG control card is used to type a message to the operator. It has the form

▽options MSG message to be typed

Typing will be prefaced by the line "MSG:" and will commence with the first non-blank character of the specifications field. If the "options" field contains an H, the operator will be given the opportunity to execute or scratch the run as described in Chapter IX.

Typing may be suppressed (resulting in the printing of the MSG control card on the printer only) by an N option. In this case the H option is not effective.

3.2 MAGNETIC TAPE ASSIGNMENT

3.2.1 GENERAL

The selection of the tape unit to be used for a particular reel is left to the discretion of the operator. He may mount tapes as far ahead of their required use as ambition and the number of available units will allow. Each file to be mounted is preceded by an unsolicited keyin which will inform the EXEC II System of new tapes now at its disposal. This environment of available tapes is independent of the jobs being run.

Tape files are identified as far as the operator is concerned by means of an operational label. There need be no particular correspondence between the operational label and any label recorded magnetically on the tape itself, although a convention specifying such a correspondence will have obvious advantages for some installations. Others may find the physical reel number a more suitable operational label. An operational label may be any combination of letters and digits to a maximum of six characters. Two special characters are also used consisting of the character "*" and the character "-" to indicate a scratch tape and an empty (or malfunctioning) tape unit, respectively.

The assignment of multireel files is accomplished through the use of an operational label. Multireel files are accommodated by associating more than one tape unit with a particular operational label.* A program will use the tape units in the order indicated by the A keyin, switching between them cyclically for each entrance to the TSWAP$ subroutine described below. Subroutines TINTL$, TLABL$, TASGN$, and TSCRH$ are also available for manipulation of magnetic tape assignments.

Within a program, a tape file (one or more tape units) is referred to by means of its logical unit designation. This designation is the six-bit Field Data code for

---

*The system allows a maximum of eight units to be associated with a particular operational label.

a letter of the alphabet and may be assigned to magnetic tapes by a control card, subroutine linkages, or both. Logical unit designations are associated with operational labels by means of the ASG control card.

The device of operational labels and logical unit designations, while appearing clumsy at first, has several distinct advantages. First, interchangeability of programs between 1107 installations is enhanced since nowhere within a program is a specific channel or unit referenced. Second, there is at least some flexibility as to the number of tape units necessary for programs employing multireel files -- the very ones usually requiring a large number of tape units. Thus, protection against lost time due to malfunctioning tape units is provided. Third, operators are given a wide choice as to the location of a given tape. This allows sensible premounting of tapes to ensure efficient job-to-job transition. Furthermore, the operator is freed from making switch settings or plugins at the tape synchronizer. This feature is particularly important for Uniservo IIIC units since no tape activity can occur during these rearrangements.

3.2.2 THE TAPE ASSIGNMENT KEYIN

An unsolicited keyin of

A c/u, c/u, ... xxxxxx

will associate each of the tape units c/u" (channel and unit) with the operational label "xxxxxx". At least one space must follow the A and the last of the unit numbers. Spaces may follow (but not precede) the commas if desired.

The keyin will be rejected if any of the following should be true:

a. One or more of the physical tape units is assigned to the program currently operating.

b. The operational label given is associated with a unit(s) assigned to the program currently operating.

c. Meaningless channel number or unit is given.

d. The routine is currently processing an ASG control card.

Rule b. does not apply if the operational label specified is a "-" or "*".

If one unit of a group associated with some old operational label has its operational label replaced by a keyin, all others of that group which are not mentioned in the current list will also be typed. These tape units will automatically be given an operational label of "-".

Any unit receiving a new operational label (either "xxxxxx" or "-") will be rewound with interlock. When the interlock is removed, it is assumed that the new tape is there.

3.2.3 THE ASG CONTROL CARD

The ASG control card will associate a logical unit designation with a scratch tape or with an operational label. It is of the form

∇options ASG assignment, assignment, ...

where each "assignment" may be either "λ", which will associate a scratch tape with logical designation "λ", or "λ = xxxxxx", which will associate those tape units having operational label "xxxxxx" with logical designation "λ". The "options" field may contain the following letters (in any order) with the indicated results:

H - density set high

L - density set low

If no density setting is indicated it will be assumed to be high.

O - parity set odd

E - parity set even (for Uniservo IIIC units only)

If no parity setting is indicated it will be assumed to be odd.

R - rewind

Any option letter appearing on an ASG control card will apply to all assignments made on that card.

An attempt to assign the same logical designation more than once will produce a message on the printer but will accept the last assignment made. No harm results from assigning more than one logical designation to the same operational label.

Any number of ASG control cards may appear in a program; their effect is cumulative.

3.2.4 ASSOCIATED SUBROUTINES

The library subroutines concerned with magnetic tape assignments are TSWAP$, TINTL$, THRU$, TLABL$, TASGN$, and TSCRH$.

The linkage

```
LMJ     11, TSWAP$
+ r, 'λ'
```

will advance in a cyclical manner through the tape units associated with a logical unit. The field "'λ'" is the letter representing the logical unit designation. The field "r" may be used for rewind control. It may take on the values

r = 0	Rewind the old tape unit
r = 1	Suppress rewind of the old tape unit
r = 2	Rewind the old unit with interlock

This linkage may be generated with the procedure call

```
T$SWAP  'λ'
```

which assumes r = 0, or by the calls

```
T$SWAP 'λ', 'RWND'
T$SWAP 'λ', 'NORWND'
T$SWAP 'λ', 'INTLK'
```

for r = 0, 1, or 2, respectively.

After returning from the TSWAP$ subroutine, the next reel number is left in register 12.

The linkage

```
LMJ     11, TINTL$
        + 0, 'λ'
```

is used to reinitialize the cycle of tape swapping. It is not required unless the program is to make more than one pass over the same multireel file. This linkage may be generated by the procedure call

```
T$INTL  'λ'
```

The linkage

```
LMJ     11, THRU$
        + r, 'λ'
```

may be employed to cause the deassignment of a logical unit designation. It will rewind (according to the value of "r" as discussed above) all of the tape units associated with logical unit "'λ'". Output to the typewriter will be made, informing the operator that the units involved are now free. This linkage is made between runs with r = 0 for each logical unit remaining assigned at the termination of the previous run; hence, there is little point in employing the subroutine unless the tape may be released well prior to run termination. Tapes may be deassigned by the procedure call

```
T$HRU   'λ', 'rewind signal'
```

A program may recover the operational label associated with a logical unit designation by means of the linkage

```
LMJ     11, TLABL$
        + 0, 'λ'
```

The operational label for tape "'λ'" will be left in register 12.* With the exception of this and the next two subroutines, a run will be terminated if a reference is made to a logical unit designation which has not been assigned to the run. If no assignment for "'λ'" is available, register 12 will be left cleared. This linkage may be generated by the procedure call

```
T$LABL   'λ'
```

Tapes may be assigned by means of library routines as well as by the ASG control card. To assign a logical unit designation to a given operational label, place the desired label * in register 12 and execute the linkage

```
LMJ     11, TASGN$
        + 0, 'λ'
        (abnormal return)
        (normal return)
```

The abnormal return will be made if the operational label is not available. No assignment will have been made in this case. The normal return indicates that the assignment has been made.

Similarly, a scratch tape may be assigned by the linkage

```
LMJ     11, TSCRH$
        + 0, 'λ'
        (abnormal return)
        (normal return)
```

The abnormal return in this case indicates that no scratch tapes are available.

These linkages may be generated by the procedure calls

```
T$ASGN   'λ', abnorm
```

and

```
T$SCRH   'λ', abnorm
```

---

*The labels treated by TLABL$ and TASGN$ should be right-justified and zero-filled.

The command

J abnorm

will be generated for the abnormal return.

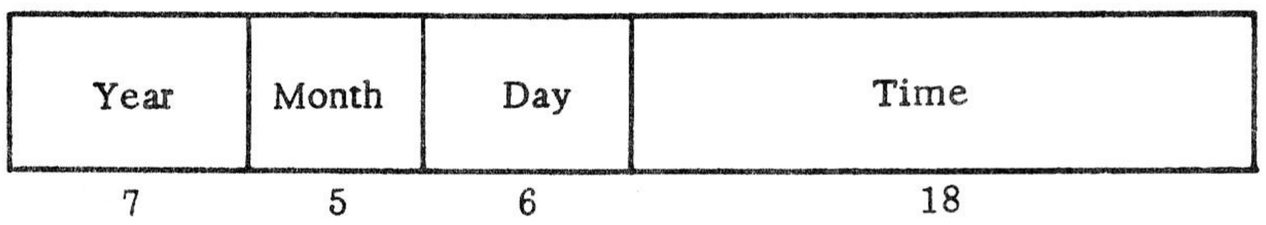
3.3 MISCELLANEOUS FEATURES

3.3.1 REAL-TIME CLOCK

The EXEC II System maintains control over the real-time clock facility in the 1107. This clock is actually an interval timer which resides in register 64 (R0) of film memory. Should this cell be overstored by a program bug, the system will lose track of the actual time of day, with obvious undesirable results.

In some cases the system can detect overstoring of the real-time clock register. The operator is given the opportunity to reset the clock after the job has terminated.

A cell called MTOD$ is updated by the system every second so that it reflects the current time of day. It has the format



in which "year" is the calendar year modulo 100; "month" is the current month as 1 through 12; "day" is the current day of the month as 1 through 31; and "time" is the time of day in seconds from midnight.

A simple editing subroutine is available in the library which will edit the time field. The linkage

```
LMJ     11, ETOD$
F       0, m, loc
```

where

```
F     FORM   6, 12, 18
```

will cause the eight characters

hh: mm: ss

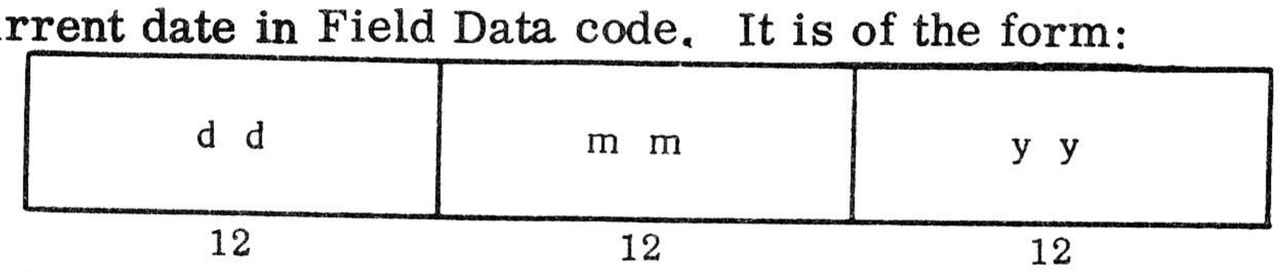
representing the hours, minutes, and seconds of the current time (on the 24-hour clock) to be stored in Field Data code with the least significant character

in character position "m" of the cell "loc". For this purpose the characters are numbered as 0 through 5 from left to right across the word. The procedure call

```
E$TOD      col, image
```

will generate the linkage which stores the data with its least significant character in column "col" of the image area beginning at "image". The leftmost character of the image is considered as column 1. The field "col" may range from 1 to 128 (See also Section 3.9).

A second cell called MDATE$, also in the resident system, is used to hold the current date in Field Data code. It is of the form:



in which "dd", "mm", and "yy" are the characters for the day, month, and year, respectively.

A library subroutine is available for editing the date. It has the linkage

```
LMJ        11, EDATE$
F          0, m, loc
```

which stores nine characters of the form

ddΔxxxΔyy

where

"dd" is the day of the month,

"xxx" is the usual three-letter abbreviation for the month, and

"yy" is the last two digits of the year.

The fields "m" and "loc" are as above. This subroutine linkage also may be generated by the procedure call

```
E$DATE     col, image
```

where the above definitions hold.

The real-time clock routine is responsible not only for maintaining the cell MTOD$ but also for determining the time limit for a run.

3.3.2 SYSTEM RETURN POINTS

The 1107 EXEC II System provides three return points to which a program may jump when it has nothing further to do. These are:

MEXIT$ - the normal return point;

MERR$ - the error return point; and

MXXX$ - the abort return point.

These returns differ with respect to the subsequent behavior of the system.

In the case of MEXIT$ the executive will continue executing the run as directed by any control cards present. A PMD card with an E option will be bypassed. Non-control cards and EOF cards will also be bypassed with a notation to the effect that such were encountered. Any dynamic dumps taken by the program will be recalled from drum, edited, and printed whenever control is returned to the system via MEXIT$.

The return point MERR$ will also result in the editing of dynamic dumps after which the system begins reading cards with the following behavior:

a. Data cards and the EOF control cards will merely be bypassed;

b. PMD control cards with or without the E option letter will be honored and the dump performed; and,

c. Any other control card will be ignored and result in all subsequent control cards also being ignored until a RUN or FIN control card is encountered.

A transfer to the error return point will always produce the message:

ROUTINE TERMINATED AT MERR$

The following messages may also be printed at this time.

SYSTEM IN INTERRUPT CODING FOR CHANNEL xx

The return to system occurred during interrupt coding. This probably

indicates difficulty in an end-action routine or in a dispatcher level input/output routine (see Sections 3.3.2 and 3.1).

ILLEGAL OPERATION xxxxxx xxxxxxxxxxxx

An illegal operation (function) interrupt occurred. The location and its contents accompany the message. If this location is 000177 or less, the contents field is not meaningful (see Section 2.4).

STORE ERROR AT xxxxxx OR xxxxxx

An attempt to store into a locked out portion of core occurred. Since the captured address has an ambiguity of one, both possibilities are printed. If these addresses are in the system's area, the condition probably arose by jumping into the system at an inappropriate point.

OPERATOR KILLED RUN AT xxxxxx

The error termination was forced by the operator by an unsolicited key-in of "E" (see Section 9.4.1).

xxxxxx - CALL ERROR FROM xxxxxx

This message is produced from a variety of routines. The first variable field indicates the name of the routine detecting an improper calling sequence and the second the contents of B11 when the error was detected.

xxxxxx FAULT nn/cc sssssssssss

This is the same fault message that is put on the typewriter when an input/output package detects an unrecoverable error condition (see Section 9.5). In I/O, the fault does not of itself cause an error termination, but in the event of error termination the last message since the beginning of the run is printed. Thus, error returns from the "check routines" (see Section 3.4.1 and 3.5.1) may be filled as MERR$ and the reason for error termination automatically noted.

Any return to MERR$ is always accompanied by a printout of the B, A,

and R registers.

The abort return point, MXXX$, will immediately terminate the run with the message

RUN ABORTED

Cards will be read by until a RUN or FIN card is encountered. Dynamic dumps will not be edited and post-mortem dumps will not be honored.

3.4 INTERNAL INTERRUPT CONTROL

The 1107 provides six internal interrupts which are associated with various conditions in a program. These hardware-defined interrupts and the standard system treatment of them are described below. An additional "interrupt" associated with MERR$ is provided by the system.

3.4.1 STANDARD SYSTEM BEHAVIOR

a. Illegal Function Interrupt: A function code has been detected which is not in the repertoire of the 1107. The executive routine will terminate the current run through MERR$ with an appropriate message.

b. Trace Mode Interrupt: A jump command was executed while the system is in the trace mode. The executive routine will reset the trace mode and continue with the run. The upper half of film register 67 (R3) will have been destroyed.

c. Memory Lockout: An attempt was made to store into a core area not available to the user. The executive routine will terminate the current run through MERR$ with an appropriate message.

d. Characteristic Underflow: A floating-point instruction was executed which resulted in a negative biased characteristic. Provided that the floating-point instruction was not reached by means of an EX instruction, both arithmetic registers containing the result are reset to zero. The address of the offending command is stored into cell MCUF$ (film register 124).

e. Characteristic Overflow: A floating-point instruction was executed which resulted in a biased characteristic greater than the eight bits allowed. The address of the offending command is stored into cell MCOF$ (film register 125). The arithmetic registers are not altered.

f. Divide Overflow: A divide command occurred which produced a quotient exceeding 35 bits in magnitude. The cell MDOF$ (film register 126) is set to the address of the offending command. The arithmetic register normally receiving the quotient will be cleared. The remainder register is not disturbed.

g. Transfer to MERR$: A jump to MERR$ has occurred somewhere within the program or system. It is not possible, in general, to determine the point from which the jump occurred. For a description of standard system behavior see paragraph 3.3.2

3.4.2 INTERNAL INTERRUPT CONTROL SUBROUTINES

Two subroutines, MSEA$ and MREA$, are provided in the resident executive routine for the control of the internal error interrupts. They serve to replace the standard system behavior described in the previous paragraph with a routine the user supplies, and to reset it to the standard.

The set error action linkage

```
LMJ     11, MSEA$
        + 0, ident
        instruction
```

causes the instruction "instruction" to be executed in the event of an internal interrupt of the type specified by "ident". Possible values of "ident" are

MIFCT$	Illegal Function Code
MITRC$	Trace Mode
MILOK$	Memory Lockout
MICUF$	Characteristic Underflow
MICOF$	Characteristic Overflow
MIDOF$	Divide Overflow
MIERR$	Jump to MERR$

The set error action linkage is generated by the procedure call

M$SEA ident, + (instruction)

The reset error action linkage replaces the user's instructions with one normally used by the system. It has the form

LMJ 11, MREA$

+ 0, ident

where "ident" may be any of the above list or may be MIALL$, which will result in resetting all of the error actions. Error actions may be reset by the procedure call

M$REA ident

Whenever the user receives control as a result of a jump to MERR$, the registers and various error messages have been printed. Further, the linkage

LMJ 11, MREA$

+ 0, MIALL$

has been executed. It is suggested that the capturing of a jump to MERR$ be used for instituting special diagnostic processes and not as a part of the normal logic of the program.

3.5 INITIAL CHECKOUT SYSTEM

As the EXEX II System was being developed a simple control routine called the Initial Checkout System (ICS) was used for debugging and utility purposes. While its functions have, in general, been replaced by the more sophisticated counterparts described in this manual, ICS is still part of EXEC II and is available for use when appropriate.

ICS is used with absolute programs. These absolute programs differ from the absolute elements described in Section 5.

In order to initiate the ICS routine, a card of the form

∇ XQT ICS

is given.

ICS reads cards and executes various operations depending on the format of the information found there. These cards are of fixed field organization — their interpretation depends solely on the positions of blank columns. Comments may be written on ICS cards beginning at column 33.

ICS does not check addresses when loading information into core or onto drum. The system itself is easily violated by inappropriate use of ICS.

3.5.1 BASIC ICS OPERATIONS

The basic ICS operations include memory clearing, jumping, loading instructions or octal words, and printing out selected portions of core.

Memory Clear Card

A card of the form

```
1          9
FFFFFFΔΔTTTTTT
```

will clear memory between octal* addresses F and T. The numbers written

*All symbol concatenations used in the allowing pages represent octal numbers.

above the card image represent column numbers at which the corresponding field should be punched.

Jump Card

A card of the form

```
1       9
JJJJJJ
```

will result in a transfer of control to address J. Memory lockout is imposed unconditionally when the jump occurs. The values of all registers and the carry and overflow toggles are restored to their condition when ICS was entered (see also Sections 3.5.5 and 8.2.2).

Instruction Load Card

A card of the form

```
1           9     14  17  20 22
LLLLLLΔΔFFJJΔAAΔBBΔHΔMMMMMM
```

loads an instruction into core at location L. The fields F, J, A, B, H, and M correspond to the various portions of the 1107 instruction word. Note that no space appears between the F and J fields and that the field H contains both the incrementation and indirect address bits (see also Sections 5.8.4 and 8.2.2).

Single Word Octal Load

A card of the form

```
1           9
LLLLLLΔΔWWWWWWWWWWWW
```

serves to load the twelve octal digit field W into core at location L (see also Sections 5.8.4 and 8.2.2).

Multiple Word Octal Load Card

A card punched with a location in columns 1 through 6, zero in column 7, and a count N in column 8 will load N words ($1 \leq N \leq 6$) into core beginning with the specified location. The listing normally produced by ICS is suppressed for the multiple word octal load card. Successive words to be loaded are taken from the card beginning with columns 9, 21, 33, 45, 57, and 69 (see Section 8.2.2).

Memory Print Card

A card of the form

```
1        9  11        18
RΔFFFFFFΔTTTTTT
```

will print the current contents of core and the primary film registers. The field R is used to indicate which of the registers is to be printed according to the values

R =
- 0 - No registers
- 1 - R registers
- 2 - A registers
- 3 - A and R registers
- 4 - B registers
- 5 - B and R registers
- 6 - B and A registers
- 7 - B, A, and R registers

Core, between the limits F and T, is printed out. The amount of core printed by this specification must not exceed 32, 767 words.

3.5.2 THE SNAPSHOT OPERATION

A snapshot request may be inserted into a (previously loaded) program by means of the card form

```
1        9
LLLLLLΔΔRΔFFFFFFΔTTTTTT
```

which will replace the instruction at location L with a call to a dump routine. When L is reached during the execution of the program, registers and core are printed as indicated for the memory print card. The replaced instruction is executed after the dump is taken. At most sixteen snapshots may be specified.

The following restrictions must be observed in choosing the location of snapshots:

a. The replaced instruction must not be altered during the course of the program, nor may it be referenced as data, or by indirect addressing.

b. The replaced instruction must not be an SLJ which specifies indirect addressing and/or indexing.

c. The replaced instruction must not be an LMJ which specifies indexing.

d. The replaced instruction must not be an EX which (ultimately) references an LMJ or SLJ instruction.

3.5.3 DRUM AND TAPE OPERATIONS

ICS contains facilities for transferring blocks of information between core and drum and between core and magnetic tape.

Drum Write Card

A card of the form

```
1        9                 18          25
         DDDDDDDDΔFFFFFFΔTTTTTT
```

will write first an access word constructed as

```
+ T-F+1, F
```

onto drum at address D and then follow it with T-F+1 words taken from core address F at address D+1.

Drum Read Card

A card of the form

```
1        9
         DDDDDDDD
```

will read an access word from drum address D and will then read according to that access word from drum address D+1.

Tape Write Card

A card of the form

```
1        9          16
λ        FFFFFFΔTTTTTT
```

where "λ" is a letter representing a tape unit which will write onto tape "λ" block consisting of the information between F and T.

Tape Read Card

A card of the form

```
1       9
λ       FFFFFF
```

will read one block from tape "λ" into core beginning at address F.

Tape Rewind Card

A card of the form

```
1
λ
```

will rewind tape "λ".

3.5.4 INTERNAL CALLS TO ICS

The programer may put ICS into operation from a point within his program by executing the instruction

```
J       MILDR$
```

Cards will then be read and the indicated operations performed. No return is made unless made by a jump card.

The programer may also avail himself of the dump routine in ICS with calling sequence

```
SLJ     MIDMP$
F       registers, length, address
```

where

```
F       FORM    3, 15, 18
```

and "registers" indicate the registers to be printed by a value from 0 to 7 as indicated above, "length" is the number of core locations to be printed and "address" indicates the location of the first word to be printed.

3.5.5 Q OPTIONS

The control cards ASM and PMD may be given with an option letter of "Q". In this case of the ASM, the assembler functions in its usual fashion except that cards punched by an accompanying P option letter will be in the "multiple word octal load card" format described. If an expression appears

on the final END line of the assembly, a "jump card" will be punched also (see Section 5.5).

A PMD card punch in the form

```
1          9  11        18
ΔQΔΔPMDΔRΔFFFFFFΔTTTTTT
```

or

```
1          9  11        18
ΔQEΔPMDΔRΔFFFFFFΔTTTTTT
```

will result in a memory print card. If both Q option and normal PMD cards are to be used for the same run, the Q option cards should precede the others. The last 4000 (octal) words of the user's area of core may not be printed in this manner. If the Q option is present on a PMD card, the only other option letter available is E which results in a dump only in the event of error termination .

ICS OPERATION CARD SUMMARY

	Col. 1	Col. 9
Memory Clear	FFFFFF	TTTTTT
Jump		JJJJJJ
Instruction Load	LLLLLL	FFJJ AA BB H MMMMMM
Single Octal Load	LLLLLL	WWWWWWWWWWWW
Memory Print		R FFFFFF TTTTTT
Snapshot Request	LLLLLL	R FFFFFF TTTTTT
Drum Write		DDDDDDDD FFFFFF TTTTTT
Drum Read		DDDDDDDD
Tape Write	λ	FFFFFF TTTTTT
Tape Read	λ	FFFFFF
Tape Rewind	λ	

IV. INPUT/OUTPUT FACILITIES

4.1 THE DISPATCHER

A resident routine called the dispatcher is at the heart of all communication between the computer and its input/output devices. The dispatcher maintains a queue of channel requests for each channel and will honor each in turn as the channel becomes available. In addition, the dispatcher controls the operation of the multiprogramed symbiont routines, interrupting the user's program temporarily to give control to a symbiont and returning control to the user when the symbiont has released. The dispatcher maintains a pool of buffer areas for the symbionts and a similar pool of drum areas when drum symbionts are employed.

The EXEC II System provides subroutines for communication with each of the input/output devices. The subroutines in turn make all the required calls on the dispatcher. As a result, the user will not normally have reason to link directly to the dispatcher unless he desires input/output operations for which the available subroutines are not suitable.

4.1.1 CHANNEL REQUEST

An input/output channel must complete all previously requested operations before it can be used. Requests for a channel which have not yet been honored are queued by the dispatcher. To make a request for a channel, use the linkage

```
LMJ     11, MRQC$
        + loc initial coding, channel #
        + parameter word
```

in which the "channel #" field contains a value from 0 to 14. The console channel, number 15, is treated separately and should not be requested through the dispatcher. The dispatcher will place the request on the queue associated with the specified channel, and return control to the requesting program following the calling sequence.

When the channel becomes available, the dispatcher will transfer control to the address specified as "loc of initial coding". If the channel is free at the time the request is made, this transfer occurs immediately and control returns to the requesting program later. The "parameter word" is a 36-bit quantity which will be left in A0 when the transfer to initial coding is made. This number is not examined by the dispatcher and may be of any format desired.

4.1.2 INITIAL CODING

The initial coding which is entered by the dispatcher is responsible for starting the I/O operation. Four "initiate mode" routines are available within the dispatcher for this purpose. The calling sequences are

Initiate Function Mode

```
LMJ     11, MIFM$
        + monitor point, loc of access word
```

Initiate Input Mode

```
LMJ     11, MIIM$
        + monitor point, loc of access word
```

Initiate Output Mode

```
LMJ     11, MIOM$
        + monitor point, loc of access word
```

Initiate Input and Function Modes

```
LMJ     11, MIIFM$
        + input monitor point, loc of input
          access word
        + fctn monitor point, loc of fctn
          access word
```

in which "monitor point" is the address to which control is transferred on the occurrence of an internal interrupt. If this address is specified as zero, the mode will be initiated without monitor. The parameter "loc of access word"

is the address of an access word to be used for the information transfer. For the initiation of an input mode, the access word given is checked to prevent the system from being destroyed by the input operation.

The initiate mode routines do not return following the calling sequence. Instead, control is returned to the interrupted program. The I/O routine regains control at the occurrence of an internal or external interrupt.

If an external interrupt is to accompany the input/output operation, then the linkage

```
LMJ     11, MSXI$
        + address
```

should occur in the initial coding. The parameter "address" specifies the point to which control should be transferred when an external interrupt occurs. This linkage causes the desired external monitor address to be set within the system and returns following the calling sequence. Initial coding is always executed with interrupts prevented. The dispatcher receives the interrupt and saves the registers

B11, A0, A1, A2, A3, A4, A5, R1, R2, R3

as well as the carry and overflow toggles. The dispatcher in turn transfers control to the indicated interrupt point.

Within any interrupt coding, these registers may be used freely. Any others must be saved and restored by the programer. All interrupt coding must be terminated by an initiate mode linkage or by a release channel linkage (see below). This linkage will restore the registers and the carry and overflow toggles, and return control to the program interrupted. Under no conditions may interrupts be enabled by the user.

4.1.3 INTERRUPT PROCESSING

When a requested interrupt is received by an input/output routine, the same rules of deportment must be observed as described in the preceding

paragraph. The routine may initiate another mode on the channel and so wait for still another interrupt, or may release the channel. The channel release linkage is

```
LMJ     11, MRLC$
```

No parameters are required and no return is made.

4.1.4 DISPATCHER PROCEDURES

Library procedures are available for generating linkages to the dispatcher:

Request Channel

```
M$RQC   channel #, loc initial coding, parameter word
```

Release Channel

```
M$RLC
```

Set External Interrupt

```
M$SXI   address
```

Initiate Input Mode

```
M$IIM   loc. access word, monitor point
```

Initiate Output Mode

```
M$IOM   loc. access word, monitor point
```

Initiate Function Mode

```
M$IFM   loc. access word, monitor point
```

Initiate Input and Function Modes

```
M$IIFM  loc. input access word, input monitor pointΔ;
        loc. fctn access word, fctn monitor point
```

For the initiate mode procedures, omission of the monitor point specification will initiate the mode without monitor.

4.2 I/O ROUTINES COOPERATING WITH SYMBIONTS

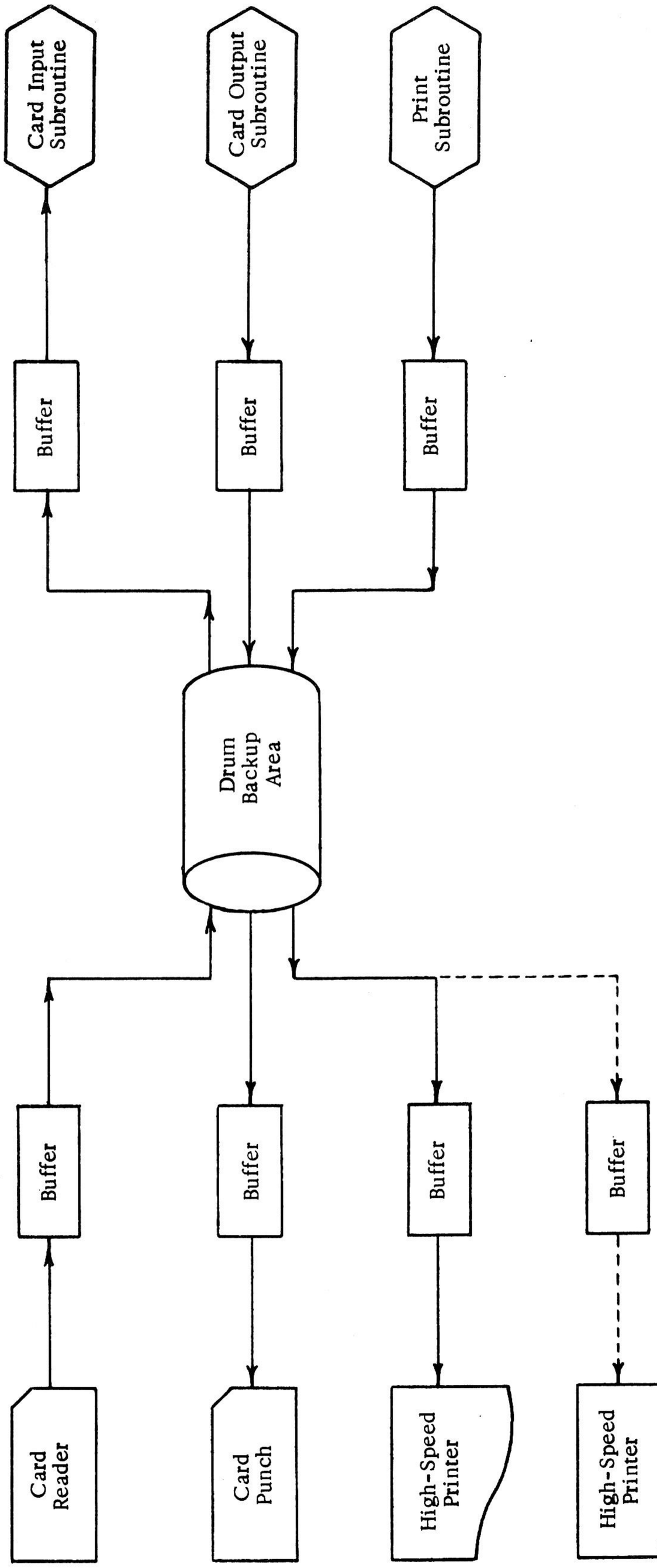
All peripheral operations can be performed on-line, and no satellite equipment will be required for normal operations. In order to maintain efficient use of the computer, a drum-buffering facility is provided which will include a large drum backup for card input and output and for printer output. When appropriate to a particular installation, tape backup may be used instead.

The routines to perform the peripheral operations will be multiprogramed with the job being run. A great deal of effort is expended within the system to provide protection to these Symbionts when they are operating with a program being debugged.

A peripheral system consists of two interconnecting programs -- a subroutine which is entered by the user, and the multiprogramed symbiont. The subroutine communicates with the drum and acts as an input or output operation. The symbiont transfers information between the drum and the actual external device.

Resident routines are used to maintain control of core buffer allocation and drum space allocation for the symbiont system. Space is provided for the symbiont programs themselves from the same pool of core areas that is used to supply buffers.

The size of the memory area set aside for the symbiont programs and their buffers is an installation-selectable parameter, although it is not dynamically alterable.



Organization of the Drum Symbiont System

4.2.1 CARD INPUT

The card input symbiont transfers card images from the reader to the drum. These card images on the drum are picked up by the card input subroutine and transmitted to the user when requested.

The input operation provides for normal mode reading only. That is, all cards are assumed to be punched in the card character set and the images transmitted are 14-word Field Data images.

It should be noted that since all 64 six-bit combinations produce legal card character codes, this mode of operation can be used as a 36-bit binary word input, with an exception on the fourteenth word. This word would contain 24 bits of zeros.

The following linkage is used to obtain a card image:

```
LMJ     11, CREAD$
+ address
(abnormal return)
(normal return)
```

Usually, the card image of the "next card" is transmitted to the area beginning at the location specified in the "address" field of the calling sequence. Control is returned to the second address following the calling sequence (the normal return).

If "address" specifies any of the index registers 1-10, the register designated is loaded with the location of the image as it resides in a core buffer, and the image itself is not transmitted. This image may be destroyed following the next entry to the subroutine.

A control card is identified by a master space in Column 1. Any control card encountered will result in the "abnormal return" from the subroutine. No image is transmitted.

Two situations can exist on an abnormal return. They are identified to the user by the value in A0.

An EOF control card is punched with

∇bEOFb

in the first six columns of the card. Column 7 may contain any character, and Columns 8 thru 80 are ignored. On encountering an EOF card, the subroutine exits to the abnormal return with the character in Column 7 located in A0. Thus A0 ≥ 0. A subsequent request to CREAD$ will cause the next card to be transmitted.

Any other control card is considered readable only by the system. In this case the abnormal return is made with A0 negative and subsequent calls to CREAD$ will continue to result in this same abnormal exit.

Linkage to the card input subroutine can be generated by the procedure

C$READ image address

If a second parameter is supplied to the procedure, a jump to the address specified by that parameter will follow the linkage and serve as the abnormal return.

4.2.2 CARD OUTPUT

A linkage to the card output subroutine will cause the image designated to be transmitted to drum for subsequent punching.

The required linkage is

```
LMJ     11, CPNCH$
        + n, address
```

in which "address" is the location of the first word of the image and "n" is the number of words in the image (n ≤ 14).

If fewer than 14 words are specified, the subroutine will provide the necessary blank words.

The image is moved before returning from the subroutine.

Calls on the card output subroutine can be generated by the procedure

C$PNCH address, n

If "n" is omitted it is assumed to be 14.

4.2.3 PRINTER OUTPUT

A subroutine- symbiont combination is provided for printer operation. To transmit a line for printing, the following linkage is used:

```
        LMJ     11, PRINT$
        P       s, n, address
```

in which

```
P       FORM    12, 6, 18
```

and

"s" is the number of lines to be spaced before printing;

"n" is the number of words in the image ($n \leq 22$); and

"address" is the location of the first word of the image.

Allowance for margins is automatically supplied by the subroutine. If $n \leq 22$, the subroutine will supply a stop code (77) following the last word of the image.

If s = 0, overprinting will occur. If "s" is larger than the number of lines contained within the margins (e.g., s < 54 for normal margin setting), the indicated image will be printed on the top margin line of the next page.

If n = 0, a blank line is printed.

The image is moved before returning from the subroutine.

The above linkage is generated by the procedure

P$RINT address, n, s

If "s" is omitted it is set to 1; if "n" is omitted it is set to 22.

A control subroutine is provided for printer formating:

```
LMJ     11, PLINE$
        + n
```

This linkage causes spacing to occur so that the page is positioned to logical line n - 1. Thus a subsequent call on PRINT$, with s = 1, will cause the image to be printed on line n. (Logical line is defined as the line number relative to the margin settings. Thus line 1 is the top margin line.)

The PLINE$ linkage is generated by the procedure

```
P$LINE    n
```

A subroutine is provided to specify a change in the type of paper form used on the printer. This subroutine is called with the linkage

```
LMJ     11, PAPER$
        + count, address
```

where the field "count" specifies the number of characters in the message and "address" is the memory location of the message that specifies the desired paper form. The subroutine will cause this message to be typed out on the operator's console at the appropriate time.

A subroutine for margin readjustment is provided with the linkage

```
LMJ     11, PMARG$
        + length, top, bottom
```

The field "length" indicates the number of lines to be printer per page, and "top" and "bottom" indicates the last line of the area to be printed. Standard conditions are 66, 6, 60, given 54 printable

If a program requests a paper change or alters the margin settings, the system will restore them to normal conditions prior to the next run.

4.3 THE RESIDENT EDITING ROUTINES

Three simple output editing routines are maintained in core at all times. These routines are used by the EXEC II System for the construction of type-writer and printer messages produced by the system. The programer may avail himself of these routines without interference with the system. They may be used by normal coding, by end-action routines, and by symbionts.

4.3.1 BINARY TO OCTAL

A linkage of

```
        LMJ     11, EBO$
        FF      n, m, addrs
```

in which

```
FF  FORM    6, 12, 18
```

will edit 3n bits from the low-order part of Register 12 to form "n" octal digits which will be stored with the least significant digit at character "m" of the word "addrs". For this purpose m should have the values zero through five for the six characters from left to right across the word. Leading zeros are not suppressed.

This linkage may be generated by the procedure call

```
E$BO    col, image, count
```

which forms "count" characters to be stored with the least significant digit in column "col" of the image area beginning at "image". The left character of the image is considered as Column 1. Column numbers may be as large as desired. Thus print or punch images may be referenced by column number.

4.3.2 BINARY TO DECIMAL

A linkage of

```
LMJ     11, EBD$
FF      n, m, addrs
```

will edit the contents of Register 12 as decimal integer with leading zeros suppressed and prefixed with a "-" if negative. The least significant digit is stored in character position "m" of the word at "addrs". A field of "n" characters is stored with leading blanks as required.

This linkage may be generated by the procedure call

```
E$BD    col, image, count
```

in which the previous definitions hold.

4.3.3 BINARY TO FIELD DATA

A linkage of the form

```
LMJ     11, EBF$
FF      n, m, addrs
```

will edit the 6n low-order bits of Register 12 as "n" characters in Field Data code, storing the least significant character in character position "m" of the word at "addrs". If "n" is greater than six, the characters will be repeated cyclically a sufficient number of times to satisfy the count.

This linkage may be generated by the procedure call

```
E$BF    col, image, count
```

in which the previous definitions hold.

V. THE PROGRAM COMPLEX FILE

5.1 MOTIVATION AND DEFINITIONS

The 1107 EXEC II System has been designed to take advantage of the fast random access drum units available. The availability of this type of storage device has allowed a considerable departure from the "classical" design of executive systems in that a program may be constructed not only from a serial file (card or tape input) of subprograms and from subroutines gleaned by a relatively inefficient search of a library tape, but from a random access file of subprograms. Rather than having the structure of a program depend on the simple existence of an order of subprograms supplied from the outside, it is now feasible to apply selection rules which determine the building blocks and a separate input (the "map") providing a structural framework upon which the program can be built.

A further and important refinement is also possible as a result of the random access storage. Instead of confining program manipulative techniques merely to relocatable subprograms, they may be extended to the treatment of symbolic programs (or their equivalents in a compressed form), to libraries used at "compile time" (e.g., assembly procedures and COBOL library entries), to descriptive tabulations (e.g, maps and possibly others), and to absolute programs ready for execution.

The area of programing perhaps most strongly affected by use of the methods outlined here is the checkout of large programs; the time taken for this is reduced by such a sufficient factor that a number of advanced programs will newly become economically feasible to develop.

The collection of program building block thus manipulated is called the program complex file (PCF); the various entries in the PCF are called elements.

The user has at his disposal two PCF's: his own and the library PCF (normally merely called the library). He may manipulate his own complex, altering its contents as desired and building programs from the elements contained therein; the library may not be altered by the user, but library elements may contribute to the construction of a program.

The PCF, while being manipulated from the system, resides on magnetic drum; however, the user may direct that the entire PCF or any of its elements be transcribed to or from either magnetic tape or punched cards.

5.2 TYPES OF ELEMENTS

At this writing the following types of elements are recognized by the system:

Type 1: Symbolic Programs

Symbolic programs may be held in the PCF for any of the languages available to the system. They may be introduced by the action of a processor or directly by the ELT control card.

Type 2: Absolute Programs

Absolute programs are placed in the PCF by the allocator using the ABS control card.

Type 3: Relocatable Programs

Relocatable programs may be produced by SLEUTH II, FORTRAN, or COBOL. In addition, a relocatable program may be constructed by the allocator using the SCD control card (see below).

Type 4: Processed Maps

A processed map is the output of the memory allocation processor. It is used to control the structure of a program.

Type 5: Compressed Symbolic

Compressed symbolic programs are equivalent to ordinary symbolic programs in nearly all respects. The only observable difference to the user will be the size of the card deck produced if the element is punched into cards.

Type 6: COBOL Library Entries

COBOL library entries are available to a COBOL compilation through the COPY verb. They are entered into the PCF by means of the COBOL library processor (CLP).

Type 7: Procedure Definition

Procedure definitions in the PCF are available to the assembler. They are entered by the procedure definition processor (PDP).

5.3 IDENTIFICATION AND MANIPULATION OF ELEMENTS

Each element in the complex is given a name. There may, however, be several elements with the same name. In order to distinguish among similarly named elements, each element is supplied with a subname called a version. Given a name and a version, the element must be uniquely identified.

Names and versions consist of one to six characters. The first of these must be a letter and the remaining letters or digits. Certain special characters are also tolerated and treated as digits. These are used by the executive and by COBOL to guarantee uniqueness in special situations. Special characters in names and versions should be avoided by the user.

It is not necessary always to specify a version. The lack of a version is treated either as a unique version or as indicating any suitable version of the named element, whichever the situation demands.

When identifying an element, the user specifies "name" or "name/version".

A third piece of titular information called a flag is also given to each relocatable element. The purpose of the flag is to automatically select the particular version of an element appropriate for the run being made, a valuable asset when checking out large programs. For example, should a correction or addition to an early phase of a large routine produce havoc, the outdated -- but workable -- version may be used to carry out total program checkout. At the same time, checkout may proceed on the latest edition. With checkout proceeding simultaneously on both early and later phases, debugging efficiency is greatly enhanced.

The flag associated with an element contains 26 bits; one for each letter of the alphabet. Associated with a given allocation will be a letter. If the flag in an element contains a one-bit corresponding to that letter, that particular version will be selected to be included in the program constructed for this run. In this manner, the programer responsible for checkout of the later phase may, by having his personal letter and insuring that the running version of the subprogram in his phase has a one-bit in that position in its flag, obtain a useful

checkout run. The process of selecting elements to make up a program will not resort to the flag unless ambiguity between names exists and a particular version has not been specified.

Elements of the program complex file are manipulated by four distinct methods:

1. By processors
2. By the complex utility routine (CUR)
3. By the ELT control card
4. By the allocator

Each of these methods will be discussed in turn in later sections of this chapter. They all have, however, the common feature that they result in the insertion and/or deletion of elements. The complex utility routine (CUR) provides, in addition, the ability to transcribe elements between the PCF and various input-output devices.

Whenever an element is inserted into a program complex file, it is determined whether the name and version of the new element duplicate those of some element already in the complex. If so, then the automatic deletion rule comes into play: If the name and version of the new element is the same as those of a previous element, then the previous element is deleted from the program complex file. In this case, an element not having a version name is treated as distinct from those so equipped.

5.4 THE TABLE OF CONTENTS

The contents of a program complex file are described by its table of contents. The table of contents is actually a collection of tables which serve various purposes in the manipulation of the PCF. We shall briefly describe these tables.

5.4.1 THE ELEMENT TABLE

The element table contains one item for each element in the PCF. It records the name, version, flag, type, and date and time at which the element was created or last altered. An element may be broken into several distinct pieces, each of which occupies a distinct area of drum. For example, a relocatable element consists of a preamble table (containing information about entry points, undefined symbols referenced, the maximum value of control counters used, and the results of processing INFO lines or the equivalent), a text file (containing the instructions and their relocation bits), and perhaps a symbol table. For each distinct piece, the Element Table entry contains a "descriptor" which locates the piece on drum.

5.4.2 THE ENTRY POINT TABLE

The entry point table contains a reference to the element table for each externally defined symbol within an element. All externally defined symbols defined anywhere in the PCF are represented.

5.4.3 THE PROCEDURE NAME TABLE

The procedure name table contains an entry locating the corresponding procedure definition entry for each procedure, function, or form contained in the PCF.

5.4.4 THE COBOL LIBRARY TABLE

The COBOL library table contains an entry for each name which may be referenced by a COPY verb in a COBOL program.

5.4.5 THE BLOCK TABLE

The block table is used to control the initial loading of information into storage areas used in common by several subprograms. This table is used to implement the BLOCK DATA feature of FORTRAN. It may, of course, be used in an analogous manner by SLEUTH II programs.

5.5 PROCESSORS

A processor is a routine which is designed to translate a source language into relocatable code or other form to be used in the construction of a program. Corrections may be applied at the same time the translation is taking place. The source language serving as input to a processor may be obtained from cards, from magnetic tape, or from an element already in the complex. As mentioned above, it may then subsequently be transmitted to another medium.

The processors currently available to the system and the SLEUTH II assembler (ASM), a FORTRAN IV compiler (FOR), a COBOL compiler (COB), the memory allocation processor (MAP) and processors for treating librarys of assembly procedures (PDP) and COBOL library entires (CLP).

The general form of the Processor Call Control Card is:

▽options prcs, loc n1/v1, n2/v2, n3/v3 (flag)

in which:

"options" is a string of alphabetic characters, representing options to be taken for this particular processing, as described below

"prcs" is the name of the processor;

"loc" gives the location of the input: omitted, input from cards; "*", input from the complex on drum; an alphabetic character, input from the corresponding logical tape unit:

"n1/v1" is the name or name/version of the source language element:

"n2/v2" is the name or name/version of the updated source language element, if one exists. (If this field is omitted, no updated source language is placed in the complex.)

"n3/v3" is the name and version to be applied to the relocatable element code resulting from this processing (if omitted, the relocatable element will have the same name as that of the updated source language element, and a version name CODE):

"(flag)" is a string of alphabetic characters enclosed in parentheses, giving the flag to be associated with the newly created relocatable code (if this is omitted, the flag associated with the relocatable code is taken to be all zeros).

The "options" field may contain the following letters (in any order) with the indicated results:

A Accept the results of the processing as correct even through errors were detected.

X Abort the remainder of the run if any errors are detected by the processor.

If neither an A nor an X are specified and errors are detected, the run will continue but any attempt to execute the program in error will be inhibited.

P Punch the resulting element into cards.

S Punch the (updated) symbolic language in compressed form. If both P and S are specified, the compressed symbolic deck will appear first.

L Produce complete printed listing.

N Suppress all printing by the processor.

If neither L nor N are specified, a partial listing will be produced. Its contents will depend on the particular processor.

I Produces single-spaced listing.

Z Suppress the formation of information to be given the diagnostic system.

Not all the processors in the system will have precisely this form for their call card. Where differences occur they will be explicitly noted.

A given run may contain as many processor call control cards as desired, each either inserting new elements into the complex or updating elements already there.

If source language is being taken from drum or tape, correction cards may follow the processor call control card. The updated source language is placed in the program complex file if and only if a name and version for the new source language element is specified by n2/v2.

Corrections are made on the basis of source line serial numbers supplied by each processor. Each group of correction cards is preceded by a card containing a minus sign in column one followed by a decimal integer or by two decimal integers separated by a comma. That is

$$\text{or} \quad \begin{matrix} -n \\ -n,\ m \end{matrix}$$

Spaces may be left following the minus sign or the comma. The first of these cards indicates that the cards following up to the next card with a minus in column one or the next control card are to be inserted into the old source language file immediately following the line with serial number "n". The second form indicates that lines "n" through "m" inclusive are to be replaced by the cards following up to the next card with a minus in column 1 or the next control card.

Corrections must be made in increasing order by serial number.

The processor call card described above applies in its entirety to calls on ASM, FOR, COB, and MAP. For calls to CLP and PDP there is essentially no distinction between the updated source language and the processed output. In these two cases then, the processor call degenerates to the form

$$\nabla \text{ options} \quad \left.\begin{matrix} \text{CLP} \\ \text{PDP} \end{matrix}\right\}, \text{ loc} \quad n1/v1,\ n3/v3$$

Note that the flag field is omitted. The "options" are restricted to A, X, S, I, and N with the above meanings.

When corrections are to be made to procedure definitions or COBOL library entries, the processed language element is addressed by the processor call card.

When the source language is specified as coming from the PCF (from drum) the system will print one of three messages depending on if error conditions are detected.

INPUT SOURCE LANGUAGE ELEMENT NOT AVAILABLE

The system was unable to locate the element "n1/v1" in either the user's or library PCF.

INPUT SOURCE LANGUAGE ELEMENT AMBIGUOUSLY SPECIFIED

More than one element exists in the user's PCF with name "n1" which would serve as suitable input for the processor being called. This message will not occur if a version "v1" is specified.

INPUT SOURCE LANGUAGE ELEMENT MALFORMED

The same language element "n1/v1" is not properly constructed. This condition could arise from a shuffled ELT deck, but probably indicates a system or hardware error.

COMPLEX OVERFLOW

Insufficient room remains in the PCF to process this element. The system remains in the normal mode to allow CUR to save the PCF as thus far produced.

ERRORS NOTED IN ELEMENT PRODUCED

The processor has informed the system that the element produced contains errors. The allocator will include an erroneous element in a program but will itself signal an error.

5.6 CARD AND TAPE FORMATS OF ELEMENTS

As well as being maintained in a program complex file on drum, elements may be held on punched cards or on magnetic tape. This section describes the formats of elements as they exist on external media.

5.6.1 THE ELT CONTROL CARD

The ELT (element) control card introduces an element into the current program complex file from punched cards. The ELT is always followed by the cards containing the element. The general form of the ELT card is

∇ ELT name/version (flag), type, date, time

in which "name/version" is the name or name and version to be associated with the element. The field "type" gives the type number of the element as described under "Types of Elements" (Section 5.2). If omitted, Type 1 (source language) is assumed.

The date and time fields identify when the element was created or last altered. The date is punched as a six-digit decimal number of the form "yymmdd", and the time is punched as the decimal number of seconds from midnight. If these fields are omitted when an ELT card is read, the current date and time will be entered into the element table.

When an element is punched by a processor or by CUR it is always preceded by a suitable ELT control card. Such decks can simply become part of the input to subsequent runs.

The automatic deletion rules apply to insertion of elements by an ELT control card.

The complex utility routine (Section 5.7) is called into play when an ELT card is encountered. The reader is referred to its description for interpretation of any error messages produced.

No option letters are associated with an ELT control card.

5.6.2 DESCRIPTOR CARDS

Each separate piece of an element is preceded by an EOF control card. Column 7 is always punched with the descriptor number in Field Data Code.

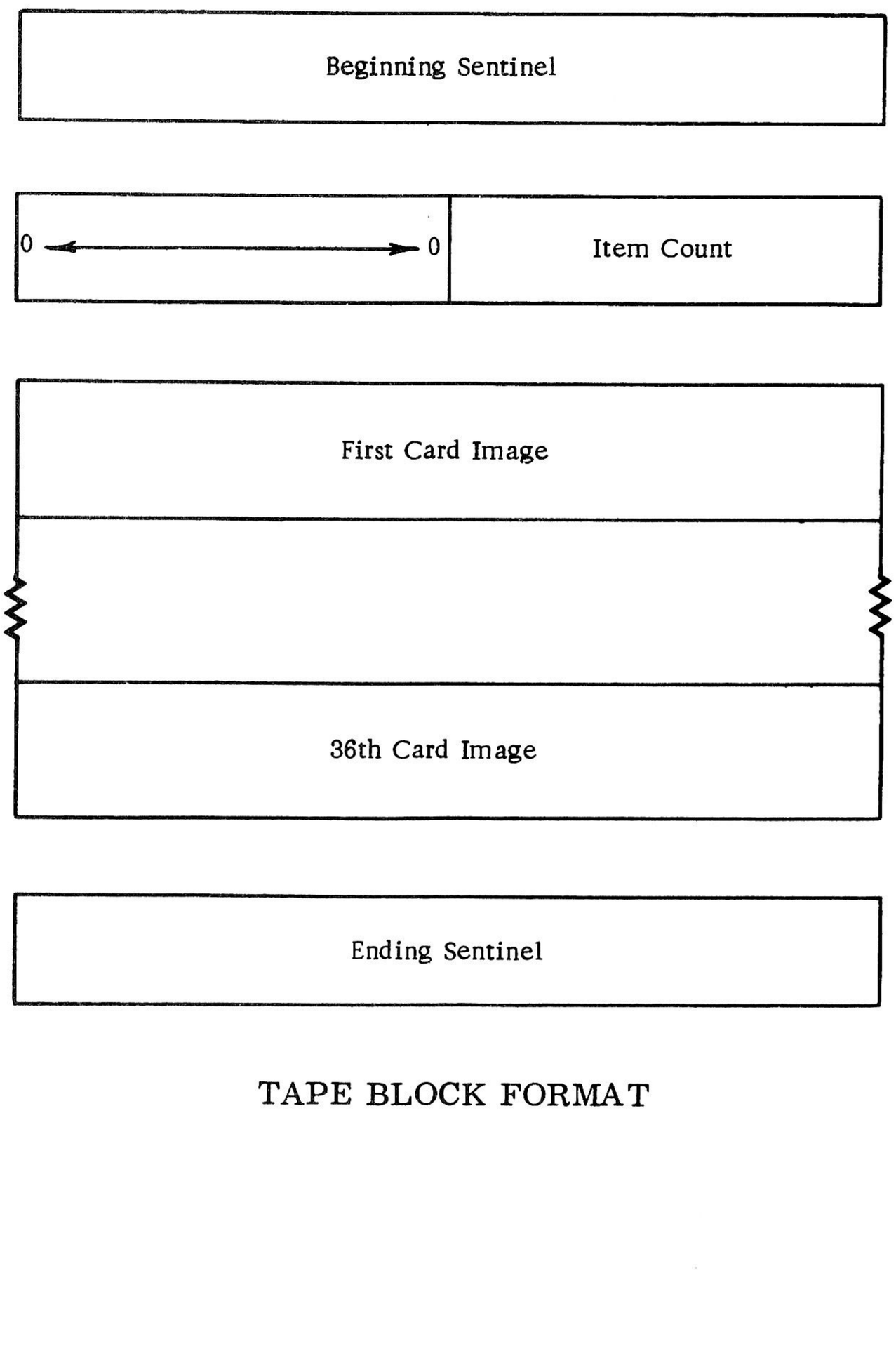
Following an EOF card will be a text file (e.g., symbolic, relocatable code, or compressed symbolic) or a table (e.g., relocatable preamble or procedures name table). Except for symbolic elements and certain other cases, these cards contain essentially binary information. They are none the less punched in the Hollerith equivalent of the Field Data characters represented by the binary. The first word of each card serves for control fields and the remaining 12 full words for the information itself. Columns 79 and 80 contain a two-character sequence field AA, AB, ---, AZ, BA, ---, BZ, ---.(As a handy tip, text and table cards will usually contain a Y punch in Column 1 whereas the control cards do not. This makes separating a deck into its components relatively easy.)

5.6.3 TAPE FORMAT

An element is recorded on magnetic tape by means of the same images that are punched into cards.

The first word of each physical tape block is the "Beginning Sentinel". It is the name of the element for the first block of an element and plus zero for all other blocks. The "Item Count" field indicates (in binary) the number of card images in the block. It may vary from zero to 36. The "Ending Sentinel" is the last word of the physical tape block. It is plus zero for all but the last block of the element and is minus zero for the last block.

When elements are written to tape, the PCF table of contents does not go with them, but is reformed when the elements are re-entered from tape.



TAPE BLOCK FORMAT

5.7 THE COMPLEX UTILITY ROUTINE - CUR

The complex utility routine provides the system with the ability to read, write, list, and otherwise manipulate the program complex or elements thereof. CUR is an interpretive routine, the units of input information being cards which follow the XQT CUR card. These input cards are of free field format; the first field, which may start in Column 1 or any succeeding column, specifies the operation which CUR is to perform. The end of CUR input is signalled by a control card.

CUR will log upon the printer all operation cards which it reads. If it reads an improperly specified input card, or encounters any other irrecoverable error, such as a tape or drum read error, it enters the error mode. In this mode the remaining input cards are simply read and logged, with the exception of TOC, TRW, and TRI which are executed. Each such card is logged with preceding asterisks to clearly identify it. At the end, if the routine is in the error mode, it exits to MXXX$.

5.7.1 THE CUR OPERATIONS

In all the descriptions, "unit" is a logical tape unit (a letter from A to Z) and "name/version" is either one label, or two labels separated by a slash (/), which are construed as the name or the name and version of an element in the complex. Completely blank cards will be ignored.

Input of an Entire Program Complex File - IN

The card

IN unit

will read a file from the designated servo and place the elements it contains in the program complex, and also form element items to be placed in the table

of contents. Should the reading of a tape into the PCF result in duplication of a previous name/version, the previous one will be deleted, and a notation to that effect made in the printed log produced by CUR. The tape is not rewound before reading.

Input of a Single Element - TRD

The card

TRD unit

will read one element from the specified unit and place the element in the program complex. The remarks made above also apply to this operation card.

Output of an Entire Complex on Tape - OUT

The card

OUT unit, type, type, ...

will write out on the designated unit all non-deleted elements, as specified by "type" (see Section 5.2), in the program complex file. If the "type" field is omitted all elements will be written out.

The sequence OUT, ERS, IN will serve to physically remove all deleted elements from the complex.

Output of a Single Element on Tape - TWR

The card

TWR unit, name/version

will write the element with the specified name and version on the specified unit.

Output of a Single Element on Cards - PCH

The card

PCH name/version

will punch an element from the drum PCF on cards. These cards will be suitable for future loading into a PCF. The requisite ELT card is punched.

Output of a Single Element on the Printer - LIST

The card

LIST name/version

will list an element on the printer. All listings are interpretive and descriptive of the type of information. For a relocatable element, for example, the special relocation bits are interpreted. The LIST function is not available for processed maps or absolute programs.

List of the Table of Contents - TOC

The card

TOC $t_1, t_2, t_3 \ldots$

will list various tables of the table of contents upon the printer. t_1, t_2, etc., are permitted to be any of the following:

EL	List Element Table
EP	List Entry Point Table
PN	List Procedure Name Table
CL	List COBOL Library Table
BL	List Block Table

Omission of the list will cause all of the tables to be printed.

Tape Rewind - TRW

The card

TRW unit, unit, ...

will rewind the designated tape unit or units.

Tape Rewind Interlock - TRI

The card

TRI unit, unit, ...

will rewind the designated tape unit or units and interlock them against further operation.

Tape End File - TEF

The card

TEF unit, unit, ...

will write a file mark on the designated unit or units when UNISERVO IIIC tape units are addressed. It is ignored for UNISERVO IIA units.

Tape Position to End of File - PEF

The card

PEF unit, unit, ...

will position the designated tape unit or units forward to an end-of-file mark for UNISERVO IIIC tape units or to the end of recorded information for UNISERVO IIA tape units.

Locating Specific Element on Tape - FIND

The card

FIND unit, name/version

will search the tape unit specified for the name/version specified and position the unit to read that element in a forward direction. If the element to be found on the tape has a version, it must be specified on the FIND card. The FIND operation searches the tape in a forward direction until either the element is located or end-of-file is reached. In the latter case, the tape is rewound and again searched. If an end-of-file again obtains, an error message is printed.

Erasing the Tables of Contents - ERS

The card

ERS

will erase the table of contents of the PCF on drum. The occurrence of a RUN control card will also erase the table of contents of the user's complex so that the ERS operation will not be required unless more than one complex is to be considered during a given run.

Marking an Element of the Complex as Deleted - DEL

The card

DEL name/version

will mark the specified name/version in the element table as deleted; in addition, the deletion will be logged on the printer, and any COBOL library entries or procedure entries defined by this element will also be deleted. The DEL operation does not physically remove the element from the element table.

Altering the Flag of an Element - FLG

The card

FLG, λ name/version (flag)

is used to set or alter the flag of any element. The letter "λ" may be I, E, or R (insert, erase, or replace). The flag in the element with the name and version given is altered by inserting the bits in the given flag, erasing (clearing) the bits in the given flag, or replacing the entire flag by the given flag. The field "flag" is a string of alphanumeric characters enclosed in parentheses which is taken as the given flag.

Altering the Version of an Element - VER

The card

VER name/version$_1$, version$_2$

will result in changing the element from "name/version$_1$" to "name/version$_2$". If the field "version$_2$" is omitted, the effect is to eliminate the version identification from the element. Should this alteration of version cause duplication of

the name and version of an element already in the PCF, that element will be deleted.

5.7.2 ERROR MESSAGES PRODUCED BY CUR

The complex utility routine during the course of its operation performs various checks on the validity of its input. Messages are produced which inform the programer of the nature of the trouble when any of these checks fail. Some errors are forgivable and CUR attempts to continue to run; others are disastrous and result in bypassing the remainder of the operation cards and aborting the run.

Besides the messages listed below, several others can occur. These will indicate, in general, failures in the hardware or bugs in the system itself. In some instances the additional error messages may be produced by cards missing from an element introduced by an ELT control card (See Section 5.7).

AN ABSOLUTE ELEMENT CANNOT BE LISTED

Self-explanatory.

BLOCK TABLE EMPTY

Self-explanatory.

BLOCK TABLE OVERFLOW

A new block has been inserted which causes the block table to overflow. CUR enters the error mode.

COBOL LIBRARY TABLE EMPTY

Self-explanatory.

COBOL LIBRARY TABLE OVERFLOW

A new COBOL library has been inserted which causes the COBOL library table to overflow. CUR enters the error mode.

COMPLEX TOO LARGE

The complex (texts and tables) has exceeded maximum length.
CUR enters the error mode.

DUPLICATE DESCRIPTOR -- IGNORED

An element being introduced from tape or cards contains two descriptors with the same descriptor type (column 7 on the EOF card). The second EOF card and all cards following it up to the next control card are ignored.

ELEMENT TABLE EMPTY

Self-explanatory.

ELEMENT TABLE OVERFLOW

A new element has been inserted which causes the element table to overflow. CUR enters the error mode.

END OF FILE -- UNIT $_x$

Always produced at the end of an IN operation if completed successfully. Produced on a TRD operation if the tape was positioned to end-of-file at the time. This message should not occur when reading the last element in a file through the TRD operation. CUR remains in the normal mode.

ENTRY POINT TABLE EMPTY

Self-explanatory.

ENTRY POINT TABLE OVERFLOW

A new externally defined symbol has been inserted which causes the entry point table to overflow. CUR enters the error mode.

EOF CARD MISSING

An ELT card, on cards or tape, is not immediately followed by an EOF card. This message does not occur if the type on the ELT card is void or is 1. The remainder of the element is ignored.

ERRORS IN CUR -- STARRED OPERATIONS NOT PERFORMED

Self-explanatory. Occurs when a control card terminates CUR while CUR is in the error mode.

NAME/VERSION HAS BEEN DELETED

The name/version specified on any CUR operation card is in the element table only in deleted form (possibly more than once). This applies to PCH, LIST, TWR, FIND, DEL, FLG, and VER operations. CUR remains in the normal mode.

ILLEGAL CHARACTER - DATE FIELD'

A character other than comma or blank terminates the date field. CUR remains in the normal mode.

ILLEGAL CHARACTER - TIME FIELD'

A character other than comma or blank terminates the time field, or a non-decimal digit occurs in the time field. CUR remains in the normal mode.

ILLEGAL CONTROL CARD IN ELEMENT

A control card, other than an EOF card, has occurred as other than the first card image on a tape block.

ILLEGAL CUR OPERATION

A card has been read and attempted to be interpreted as a CUR operation card, but does not contain a CUR operation. CUR enters the error mode.

ILLEGAL FLAG ON ELT CARD

A name or version on an ELT card is terminated by a left parenthesis, and the first non-alphabetic character following the left parenthesis is not a right parenthesis. CUR remains in normal mode.

ILLEGAL FLG SPECIFICATIONS

The FLG operation is not of any of the three forms FLG, E or FLG, I or FLG, R; or the first non-alphabetic character following the left parenthesis of the flag is not a right parenthesis. CUR remains in normal mode.

ILLEGAL NAME -- xxxxxx

A name of more than 6 characters has occurred on an ELT card. CUR enters the error mode.

ILLEGAL TOC SPECIFICATION

A TOC specification is given which is not EL, EP, BL, CL, or PN, or there are separators other than commas between the specifications (master space, left parenthesis, or slash). The TOC operation looks for more specification until it reaches the end of the card, at which point the next CUR operation is interpreted. CUR remains in the normal mode.

ILLEGAL UNIT DESIGNATION --xxxxxx

A tape unit designation given on a CUR operation card has either more than one character or a non-alphabetic character. CUR enters the error mode. (A frequent cause of this error is reversing unit and name/version specifications on a CUR operation card.)

ILLEGAL VERSION --xxxxxx

A version name of more than 6 characters has occurred on an ELT card. CUR enters the error mode.

IMPROPER COBOL LIBRARY ELEMENT

A COBOL library element being introduced from cards or from CLP contains a table whose contents are unintelligible. Specifically, it contains a COBOL name of more than 10 words in length. CUR enters the error mode.

NONSENSE IN TYPE FIELD ON ELT CARD

A slash, left parenthesis, or master space terminates the name or name/version field, or a non-decimal digit occurs after the comma. CUR remains in normal mode.

NAME/VERSION NOT IN COMPLEX

The name/version specified on any CUR operation card is not in the element table in either deleted or undeleted form. This applies to PCH, LIST, TWR, DEL, FLG, and VER operations. CUR remains in the normal mode.

NAME/VERSION NOT ON TAPE x

The name/version specified on the FIND operation is not on the specified tape unit. If no version appears in this error message, the name is not on the

specified tape unit with zero version, although it may be on the tape unit with non-zero version (such as CODE). CUR remains in the normal mode.

PROCEDURE NAME TABLE EMPTY

Self-explanatory.

PROCEDURE NAME TABLE OVERFLOW

A new procedure name has been inserted which causes the procedure name table to overflow. CUR enters the error mode.

A PROCESSED MAP ELEMENT CANNOT BE LISTED

Self-explanatory.

REQUESTED FILE IS NOT AN ELEMENT

A tape read operation (TRD or IN) has encountered the start of a block, either the first block on the tape or the first block after a sentinel block, in which the first card image is not of the form: master space, string of blanks, ELT, string of blanks, etc. This error can result from the lack of a file mark following a tape complex. Goes to error mode.

SPECIAL INFORMATION TABLE OVERFLOW

An internal error has occurred within the listing program for the text of a relocatable element. A variable number of lines may be printed beneath each word giving special relocation information; if this number is exceeded, the error message is printed. At present the maximum is 200 lines for every eight words.

TABLE OF CONTENTS CANNOT BE READ FROM DRUM

At the outset, CUR must read the table of contents from drum. If this operation produces an abnormal return, this message is displayed. CUR enters the error mode.

TABLE OF CONTENTS CANNOT BE WRITTEN TO DRUM

At the end of CUR, if it is still in the normal mode, the table of contents is rewritten to drum. If this operation produces an abnormal return, this message is displayed. Exit is made to MXXX$.

UNRECOVERABLE READ ERROR

UNRECOVERABLE READ ERROR -- DRUM ADDRESS xxxxxxxx

UNRECOVERABLE WRITE ERROR

UNRECOVERABLE WRITE ERROR -- DRUM ADDRESS xxxxxxxx

The explanation for these printer messages will appear on the Console.

UNRECOVERABLE TAPE ERROR

Self-explanatory.

VOID ELEMENT -- NOT INSERTED

An ELT card is immediately followed by a non-EOF control card, from cards or from tape. CUR enters the error mode.

5.8 THE ALLOCATOR

The allocator is a system routine which collects subprograms and interconnects them. Cross-references between these subprograms are resolved and relative locations are assigned. Depending on how the allocator is called, it may go on to produce an absolute program which is put away on drum. Finally this absolute program may be loaded into core and run.

The allocator will also, unless requested otherwise, construct a group of tables which serves as one of the inputs to the diagnostic system. If desired, the allocator can work with the output of the memory allocation processor in order to provide a flexible and sophisticated segmentation ability. In the absence of a map, the allocator assumes that all subprograms and common blocks will occupy core simultaneously.

Three separate control cards XQT, ABS, and SCD result in calling the allocator. These cards cause, respectively, the allocator to construct (if necessary) an absolute program and execute it, to construct an absolute program and put it into the user's PCF, and to construct a relocatable program and put it into the user's PCF.

5.8.1 THE XQT CONTROL CARD

The general form of the XQT control card is

∇options XQT, λ name/version

in which "name/version" is the name or name and version of the program to be executed:

" λ " is the letter to be used in selecting elements on the basis of their flags; and

"options" represents the option letters detailed below.

The option letters available for the XQT control card provide a means of controlling the allocator. The available options are:

L - Produce a complete listing. This listing will contain a summary of the memory space used by program and that used by each element included in the program. The absolute location of each symbol in those elements equipped with symbol tables will also be listed.

N - Produce no listing. The N option will be overridden if diagnostic messages are to be produced. *

A - Accept the results of the allocation as suitable for execution even though errors were detected during the allocation or even though elements marked as in error were included in the program. If the errors detected are of such a nature to prevent the production of an absolute program the A option will not be effective.

X - Abort. Do not execute the remainder of the run if errors are detected. **

Z - Inhibit the formation of tabular information to be given to the diagnostic system.

C - Insert patch cards.

In selecting the program to be executed the allocator performs the following selections in the order indicated:

a. If the name given is that of an absolute program in user's PCF, then that program is executed.

b. If the name given is that of a processed map in the user's PCF, then allocation occurs with respect to the given map.

c. If the name given is that of a relocatable element in the user's PCF, then allocation occurs beginning with that element.

d. The library PCF is searched for an absolute program, for a processed map, and for a relocatable element, in that order. Action occurs as indicated above.

---

* If neither an N nor L option is indicated, summary information only will be printed.

** If neither an A nor an X is given, the event of an error will inhibit execution of the program but will perform the remainder of the run.

If an absolute program (type 2) is selected, then it is merely loaded and entered. No allocation need occur.

The starting point of an allocated program is determined from examination of its component pieces. Any of them may specify starting addresses. For FORTRAN-generated programs, the lack of an introductory statement FUNCTION, SUBROUTINE, or BLOCK DATA indicates that the program will be given a starting address. For SLEUTH-II-generated programs, a starting address is specified for a program by means of an expression written on the terminating END directive of an assembly. In any event, the allocator will accept the first starting address encountered in the relocatable elements making up a program. Error messages will be produced if multiple starting addresses are found. (See also Section 6.7.)

We shall consider the allocation process assuming that it begins with a relocatable program. The next chapter deals with the behavior of the allocator when being directed by output from the memory allocation processor. The relocatable program makes references to symbolic addresses not defined within itself. The processor concerned collects these undefined symbols and includes a table of them along with the relocatable code for that particular element. Each external symbol definition in a program is likewise recorded in a table associated with the element. All external symbol definitions are gathered into a master table (see Section 5.4) in the PCF. The allocator is responsible for finding elements and including them in the program being constructed which will provide definitions for those symbols that remain undefined in elements already included. The addition of elements to provide these definitions may in turn introduce more undefined symbols for which definitions must be provided by the inclusion of still more elements. The process continues until all references are satisfied. The programer may thus insure the inclusion of a subroutine within his program merely by making reference to it.

In the event that more than one relocatable element in the PCF contains an external definition for a given symbol, the allocator attempts to resolve the resulting ambiguity and select a single element. The algorithms used are called the selection rules. They are applied in the order indicated:

1. If an element already selected for inclusion defines the symbol, then that element is used.

2. If precisely one of the elements defining the symbol is listed on a USE card for the memory allocation processor (see chapter VI), then that element is selected.

3. If a selection letter " λ " was specified on the XQT control card and if precisely one of the elements defining the symbol has a one bit in the corresponding position of its flag, then that element will be selected.

4. In any event elements from the user's PCF will be selected before elements in the library.

If the allocator is unable to choose a unique element on the basis of the selection rules, then an arbitrary element is chosen and a diagnostic message is produced.

5.8.2 THE ABS CONTROL CARD

The allocator, when invoked by a control card of the form:

∇options ABS, λ name/version, name/version

will produce an absolute program as it does for the XQT control card. In this instance, however, the resulting program will not be executed but instead will be entered into the program complex file as an absolute element. The second "name/version" field becomes the name of that element. The same automatic selection rules as described for the XQT card will still apply with the exception that the PCF and library will not be searched for absolute programs. The automatic deletion rules apply to the insertion of the element.

All of the option letters pertaining to XQT are applicable and, in addition, an option letter "P" will result in the punching of the element.

5.8.3 THE SCD CONTROL CARD

The subcomplex definition control card

▽options SCD, λ name/version, name/version

behaves much as does the ABS control card. A relocatable element is inserted into the complex rather than an absolute element. To be useful, a SCD control card will usually require a map. The reader is referred to the next section for details.

5.8.4 RELOCATABLE PATCHES

To inform the allocator that patches are to be made, the C option letter must appear on the ABS or XQT control card. (Patches are not allowed with SCD). The indicated patch cards immediately follow the XQT or ABS control/card in the deck. The following paragraph describes the format of relocatable patches.

The letters "cc" indicate a location counter designation and are of one of three forms.

a. One, two or three digits indicate the location counter number. A zero first digit indicates the number is given in octal.

```
8
19
101
02
```

b. An element name followed by a comma followed by a location counter number.

```
MAIN, 8
MAIN, 19
MAIN, 101
SUBRTN, 02
```

c. A blank indicating the absence of the location counter designation.

Each set of patches for an element is preceded by an element name card of the form

xxxxxx, cc

where xxxxxx is the name of the element into which the patches are to be inserted and cc is the location counter number under which the location of the patches is to be relocated. The element name must start in column 1 and begin with an alphabetic character. In this case, cc may not be of form b.

Examples:

```
MAIN, 38
SUBRTN, 010
A98
```

Patch cards following the element name card are of two formats.

a. Instruction Format

xxxxxxΔxxxxΔxxΔxxΔxΔxxxxxxΔcc

cc indicates that bits 20-35 are to be relocated by control counter cc.

Examples:

```
000001  7404  00  00  0  000101  3
000200  7413  13  00  0  000001  010
000107  7404  00  00  0  001022
000012  0616  02  00  0  000002
000103  0500  00  00  0  000000  SUBR,4
```

b. Data Format

xxxxxxΔxxxxxxxxxxxxΔccΔcc

The first cc field gives the location counter for relocating bits 2-17 and the second cc field indicates the location counter for relocating bits 20-35. If bits 2-17 are not to be relocated then the form becomes

xxxxxxΔxxxxxxxxxxxxΔcc

Examples:

```
000001 000000000000 3 5
000100 000000000001 MAIN, 2 6
000020 001000001000 SUBR, 010 A98, 8
000000 060710111213
000020 000001000000 7
000010 100000001022 010
000012 777777000000 MAIN,1
```

Patch cards following an element name card are terminated by another element name card or by an EOF control card (see 2.1.1) which is used to indicate the end of all patches. If the same location of the same element is duplicately patched, the last one in the physical deck will be the one used. Currently, the allocator will accept a maximum of approximately 100 patches.

If multiple link programs are to be patched an EOF control card must exist for each link, whether or not that particular link is being patched. Patches for a given link then receive the EOF control card for that link.

5.8.5 ALLOCATOR DIAGNOSTIC MESSAGES

{XQT, ABS, SCD} NAME NOT FOUND

No processed map or relocatable element type by this name in user or library complex. If name/version is specified on control card, version name is required in complex. (Abort condition.)

name/version AMBIGUITY IN COMPLEX ELEMENTS

For an element named on the control card or for an element named through MAP, more than one element of the type required qualifies in the user's complex or no element qualifies in the user's complex and more than one element qualifies in the library complex. (One of the ambiguous elements taken. Error exit from allocator.)

name/version ELEMENT NOT IN COMPLEX-DELETED FROM ALLOCATION

An element named through MAP is not in either the user's or library complex. (If USE of version is given, the version must appear in complex.) (Named element deleted. Error exit from allocator.)

ALLOCATION TABLES EXCEED AVAILABLE SPACE

Some condition arises causing space of some table to be exceeded. (Abort condition.)

name/version NO PREAMBLE-ELEMENT DELETED FROM ALLOCATION

Element selected for allocation has no preamble descriptor. (Element deleted from allocation. Error exit taken.)

name/version AMBIGUITY FOR ENTRY POINT eeeeee

An ambiguous situation regarding externally defined symbols has been encountered by element name/version, as follows. More than one element in the user's complex has the same externally defined symbol; or, no element in the user's complex has the name specified for the externally defined symbol; or, no segment has this name; or, more than one element in the library has an externally defined symbol for the specified name.

If an element which defines such an externally defined symbol is already in the allocation, either from the user or library complex, no ambiguity is considered to exist. (Undefined symbol relocated by zero. Error exit taken from allocator.)

name/version UNDEFINED SYMBOL nnnnnn NOT FOUND-SYMBOL DELETED

Element name/version has an undefined symbol nnnnnn which does not match an externally defined symbol name for a qualifying element in the user's complex, the library complex or a segment name. Any non-deleted relocatable element will qualify. (Undefined symbol relocated by zero. Error exit taken.)

name/version DUPLICATE ELEMENT NAMES-UNDEFINED SYMBOL DELETED

Element name/version has an undefined symbol nnnnnn which matches an entry point for a qualifying element in either the user or library complex which has

the name n but is not the same element as n/v. Different elements with the same name but different versions will cause this condition. (Undefined symbol relocated by zero. Error exit taken from allocator.)

{CORE / DRUM} STORAGE TOO SMALL FOR DESIRED ALLOCATION

The available core or drum space indicated to the allocator for use is exceeded by some non-fixed portion of the allocation. (Abort.)

EXECUTION AREA EXCEEDED

The absolute program and its diagnostic tables exceeded the length of the execution area for XQT or exceed the user's complex text and table area for ABS or SCD. (Abort.)

nnnnnn ENT NAME NOT FOUND - NO STARTING ADDRESS USED

The name given by ENT in map cannot be found as an entry point to any element in the allocation for this link. (Error exit taken.)

DRUM UNRECOVERABLE ERROR

Error return from drum packages or block buffering package. (Abort.)

5.9 COMMON BLOCKS

The allocator makes provision for the allocation of storage areas to be shared by several relocatable elements. FORTRAN and COBOL make use of these storage areas through their COMMON statements; SLEUTH II employs the INFO directive to the same end (see section 5.9.2).

5.9.1 ALLOCATION OF COMMON BLOCKS

The name of a common block must not duplicate the name of any other common block or an element involved in the allocation, although no ambiguity is introduced by duplication of common block names and externally defined symbols.

The processors reference a common block in relocatable code relative to the beginning block address, rather than by an externally defined symbol. An externally defined symbol may be used in a common block, however, if it receives only one definition.

Common block storage areas are structured by the program referencing them. References made by any relocatable element in a program complex file to the name of a given common block will be directed to the same area of storage. The size of a common block is determined by its maximum reference, thus assuring sufficient storage for all references to that block. Block lengths specified by the various elements need not be identical.

Common blocks referenced by elements in different segments of a segmented program are still assigned to the same storage area; however, common blocks are not shared across links in a chained program (see Sect. VI).

A special sort of common block, "blank common", is also available which is shared across links in a chained program.

A program may specify initial contents of a common block (including blank common), but the user should be cautioned that each time a segment is loaded which contains such an element the common area is refreshed to these

initial values. This may or may not be desirable. If more than one program specifies initial contents for a given cell in a common block, the last such program loaded will actually contribute its value.

5.9.2 REFERENCING COMMON BLOCKS FROM SLEUTH II

A program written in SLEUTH II references common blocks by means of the INFO directive. For this purpose we take the INFO directive as being of the form

label INFO gn $c_1, c_2, \ldots$

where "$c_1, c_2, \ldots$" is interpreted as a list of control counter numbers. This list defines a block of storage called a group which consists of all cells defined under $(c_1) followed by all cells defined under $(c_2) and so forth. The parameter "gn" is a group number which specifies the type of storage that is to be assigned to the group by the allocator.

A group number of zero specifies that the group is to be absolute. No storage is reserved and no relocation will occur.

A group number of one or two will cause assignment to bank one or bank two respectively. Assignment with group number one or two is dependent. Dependent assignment of a group results in the group being involved in the overlay if the element in which the group appears is overlayed; independent assignment results in storage being assigned to the group which will not be overlayed even though the remainder of the element is. Group numbers of five and six indicate independent assignment to banks one and two respectively. There is no distinction between dependent and independent assignment of storage unless the program involved is being segmented.

Group numbers three or seven cause assignment of drum space to a group. The assignment is dependent or independent respectively. No initia

contents may be specified for a group assigned to drum whether dependent or independent.

Group number four is reserved to indicated "blank common" in the FORTRAN sense. All groups occurring in a program with a group number of four will be assigned the same core area in bank two. When a chain of programs is being allocated the storage will be the same for all links on the chain, thus allowing core communication between the links.

As indicated above, a label may be written for the INFO directive. If present, this label serves to name the group. No definition is made by the assembler for this label. This label becomes the name of a common block.

In the absence of an INFO line containing a particular control counter number used in an assembly, the allocator will assign even-numbered control counters (including zero) to bank two and odd-numbered control counters to bank one. These assignments will always be dependent and will be in order of increasing control counter number.

5.9.3 BLOCK DATA PROGRAMS

In order to accommodate the FORTRAN BLOCK DATA feature, the system keeps a special table (the block table) of common block names for which there are BLOCK DATA programs. Whenever a common block is included in a program, the block table is searched to determine whether there are BLOCK DATA entries for it. If so, the initial contents so specified are loaded with the segment in which the common block resides. If a given BLOCK DATA program specifies initial contents of more than one common block, then all such common blocks must be in the same segment. If a BLOCK DATA program specifies initial content for (say) two common blocks, the first of which is referenced in the program being allocated but the second is not, then the second is simply ignored by the allocator in making up the absolute program.

Finally, if more than one BLOCK DATA program specifies initial contents for a common block, the allocator will choose one or more of them to supply values to the common block according to the following rules:

a. If a USE operation referencing the BLOCK DATA element was given in a map, then that element and that element only will be used.

b. If a flag letter was given to the allocator and some of the BLOCK DATA programs contain that particular flag bit while others do not, then all the elements containing that flag bit will be selected.

c. Otherwise, all elements will be selected.

If the above process results in the selection of more than one BLOCK DATA program to supply initial values to a common block, and more than one of these specifies a value for the same cell of the block, it is in general impossible to predict which value will be chosen.

5.9.4 WRITING BLOCK DATA PROGRAMS IN SLEUTH II

A special set of group numbers is used to pre-load labeled common blocks which will reside in core. This assembly equivalent of the FORTRAN BLOCK DATA feature consists of grouping assembled information into labeled common blocks with group numbers 33, 34, 37, and 38 which indicate bank one dependent, bank two dependent, bank one independent, and bank two independent assignments, respectively. The output of a BLOCK DATA compilation in FORTRAN always uses group number 38. When a BLOCK DATA program is being assembled, all control counters employed must explicitly appear in some labeled INFO line with group number 33, 34, 37, or 38. No other group numbers may be used in the assembly and these particular group numbers may not be used in an assembly containing group numbers 0 through 7.

BLOCK DATA assemblies may not require relocation of the information to be loaded nor may they contain references to undefined symbols.

5.10 THE LIBRARY

In addition to his own program complex file, the user has available to him a collection of elements in a library. This library is itself a complex and is built up at an installation as is any other complex. If, during the allocation process (or for that matter, in nearly any search of the table of contents of the PCF), a given element or entry point is not found, the library table of contents is searched and the appropriate element is taken from the library. Thus the elements in the library serve to "back up" the user's complex. If, however, the user's complex contains an element whose name is the same as one in the library, or an element which has an externally defined symbol which which appears in the library, the user's complex will override the library.

Note that all problems of library maintenance are automatically solved by treating the library as "just another complex". All that is required is a simple utility routine, LIBRY, which copies a complex to the system drum area.

The routine LIBRY is available on the standard library as an absolute element. It may be called by an XQT control card.

It is suggested that LIBRY be given only a "clean" PCF to put in the system library, i.e., there should be no deleted elements present. A PCF may be cleaned up by using CUR and a tape as follows:

```
▽   XQT     CUR
    TRW     Q
    OUT     Q
    TEF     Q
    TRW     Q
    ERS
    IN      Q
    TRI     Q
    TOC
▽   XQT     LIBRY
```

Thus tape "Q" may be saved to serve as the input to the next library update.

VI. THE MEMORY ALLOCATION PROCESSOR

6.1 INTRODUCTION

The memory allocation processor (MAP) operates on a source language which is used to specify in detail the structure of a program. Its output is a series of tables called a map, which may be given to the allocator to instruct it of the storage layout the user desires.

The allocator will produce a "reasonable" storage layout in the absence of a map. MAP language may indicate:

1. A chain of programs to be run more or less independently of one another, sharing only an area of core and an area of drum. (I/O devices are automatically shared since all the programs would operate under a single RUN control card.);

2. Any non excessive number of levels of segmentation with a program and the make-up of the segements;

3. The areas of core and drum memory which are to be available for the program;

4. The precise position that a particular piece of a program is to occupy in core or on drum;

5. Which technique of loading is to be used with a particular segment;

6. The arrangement of common storage areas to be used by various subprograms;

7. The first instruction to be executed in a program; and

8. Which of several versions of a particular relocatable element is to be used in ambiguous cases.

Rules are used by the allocator for making most of the above decisions automatically in the absence of direct specification. Only in unusual circumstances must the user avail himself of many of these specifications. When automatic decisions are available, they will be carefully defined in the following pages.

The source language input for MAP follows the same general rules of syntax as does the SLEUTH II assembler. In particular, if a line is to have a label (SEG cards only), it must begin in column one. If column one is blank, it is assumed that no label is intended. Following the label (if any), and separated from it by a string of one or more spaces, is a mnemonic operation. Following the mnemonic operation, again after a string of spaces, is the specifications field. Names and versions of elements and names of segments must begin with a letter and consist of no more than six letters and digits. If a semicolon is encountered in the specifications field, it and the remainder of that card are bypassed and MAP continues scanning with the next card.

6.2 CHAINED PROGRAMS

Several relatively independent programs called links in a chained program may be allocated simultaneously. The programs may share magnetic tape assignments by virtue of the fact that they operate as a single run (see Chapter II), and thus may have a common set of ASG control cards. These programs may also have a core area in common. This core area has traditionally been called simply "common", but we shall follow the FORTRAN IV usage here and refer to it as "blank common".

The manner in which a program references common storage (blank or named), depends on the particular processor which generated the program. The reader is referred to the individual processor manuals for the appropriate explanations.

Each link on a chain is associated with a positive integral number which must be no greater than 32,767. Links are referenced by this identifying number. A particular link of a chain is executed by placing its number in film register 12 and giving the linkage

```
LMJ     11, MCHN$
```

The system will delay until the previous program has released all I/O channels it was using and will then load and jump to the program loaded. There is no restriction on the order of execution of the links of a chain nor on how many times a link may be executed. The link executed first will always be the first one defined in a map. Each link of a chain is introduced in MAP language by a card of the form

```
CHN     n
```

where "n" is the identifying number to be associated with that link. All MAP cards following a CHN card refer to that particular link until another CHN card is encountered or the end of the source language is reached.

Since the various links in a chained program are allocated independently (with the exception of blank common), the restriction that external labels be unique within a program is relaxed to say that uniqueness must be retained only within a link. If uniqueness is not maintained within a program, however, the allocator must be informed of the particular subprogram intended to define an external label (see USE operation described in Section 6.7).

Uniqueness may also be obtained by writing the CHN operation as

CHN n, λ

where "λ" is a letter of the alphabet. This letter will be used as a flag selection criterion in place of the one specified on the XQT or ABS control card.

Single programs (although themselves segmented) do not require the CHN operation at all. Although useful in its own right, the primary motivation for including the CHN operation was to maintain compatibility with existing FORTRAN programs which employ the feature.

When coding in FORTRAN, a link of a chain may be loaded by the line

CALL CHAIN (r, t)

where "r" is the link number. The parameter "t" is ignored and may be omitted if desired.

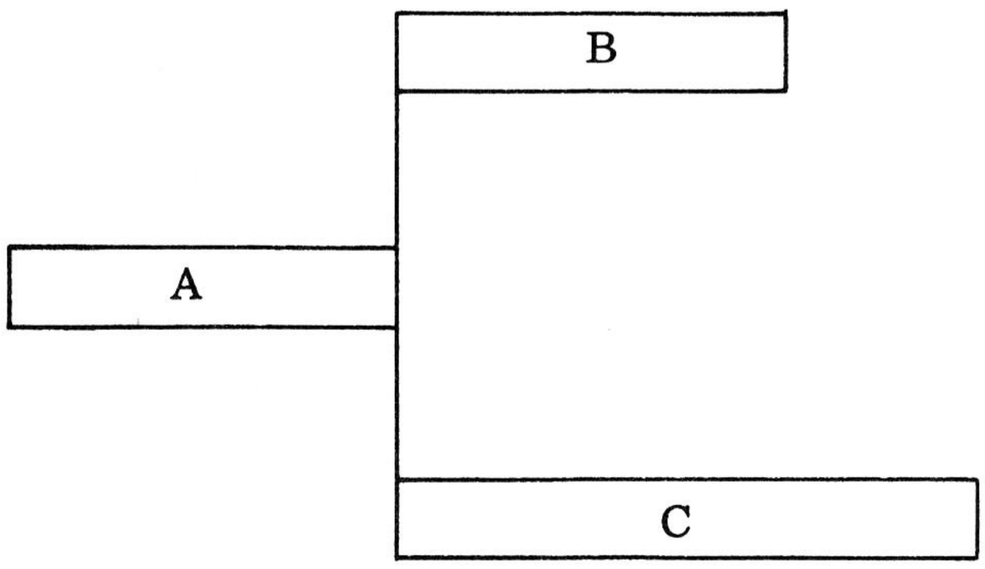
The CHN operation may not be used when a subcomplex is being defined.

6.3 SEGMENTATION

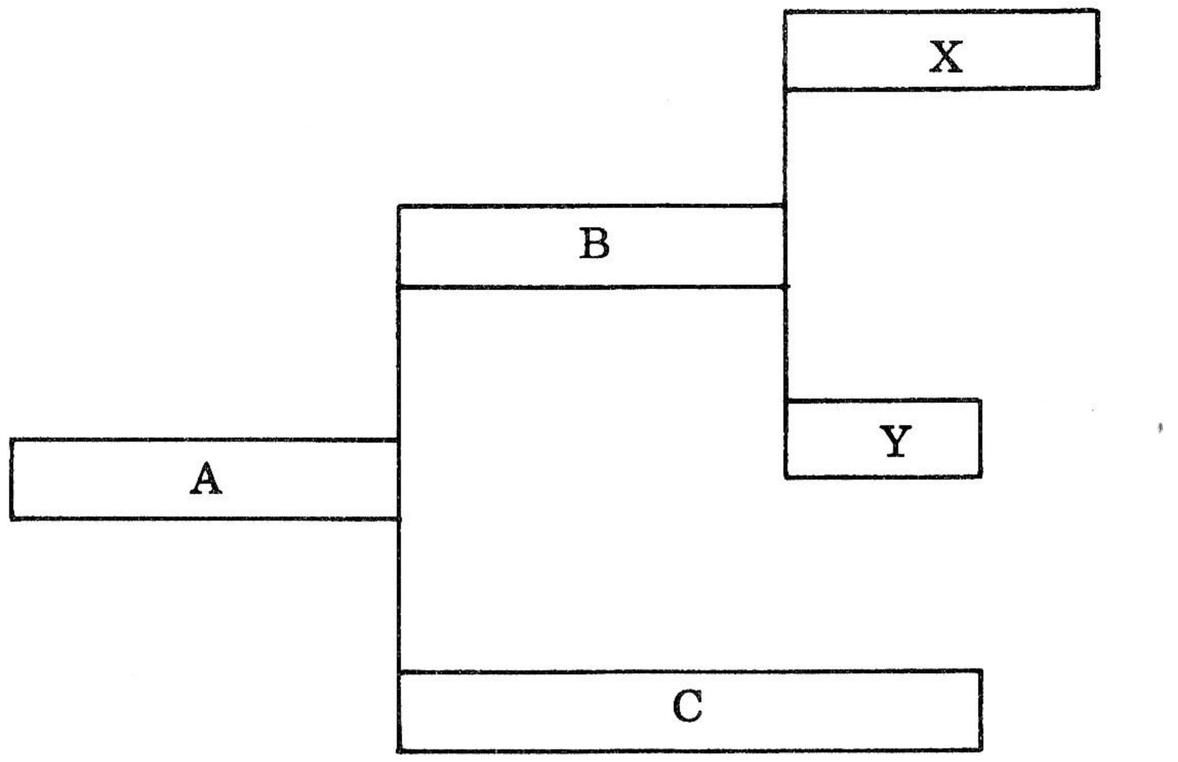
6.3.1 THE NOTION OF SEGMENT

A segment of a program is defined as that portion of memory which is committed by a single reference to the loader. Usually a segment overlays some other segment and may have within itself other portions which in turn overlay one another; i.e., subsegments. That part of a segment which is actually brought into memory when the loader is referenced is called the fixed part of a segment. Segments are built up from separate relocatable elements, common blocks, or other segments.

A diagrammatical representation of segments and subsegments will be useful in the discussions which follow. We shall represent a segmented program as a "tree". A horizontal coordinate is used to denote increasing memory (core or drum) addresses from left to right; a vertical coordinate is used to represent different points in time in no particular order. Thus, for example, if a program is to consist of a portion A which is to remain in memory at all times and two portions D and C which follow A and are to overlay one another, we would use the diagram



If portion B is to be followed by two subsegments of its own, X and Y, the diagram becomes



and so forth.

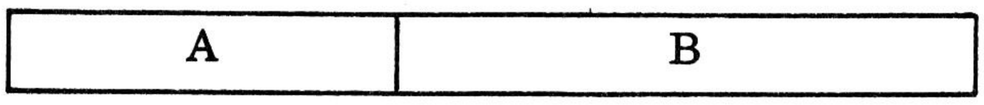
6.3.2 ALGEBRAIC REPRESENTATION

Segment specifications are given to the MAP by means of a pseudo-algebraic language employing the symbols:

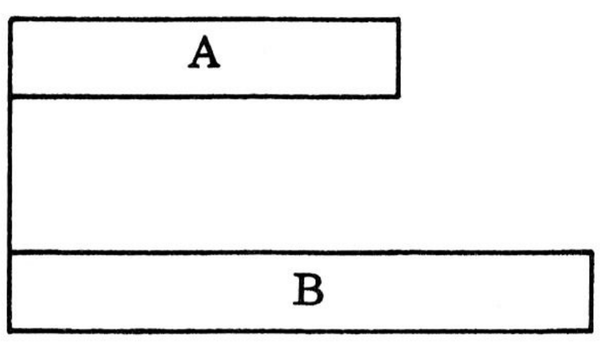
labels	are the names of the constituents of a segment. They may be the names of relocatable elements, names of common blocks, or the names of other segments;
-	a binary operation which indicates that two constituents are to occupy memory simultaneously;
,	a binary operation which indicates that two constituents are to begin at the same point and thus overlay each other;
*	a unary operation to be defined later; and
( )	which indicate grouping after the usual manner of parentheses.

The operation "-" is considered to have greater precedence than the operation ",".

Using this notation, the memory description "A-B" would be diagramed as:



and the description "A, B" would be diagramed as:



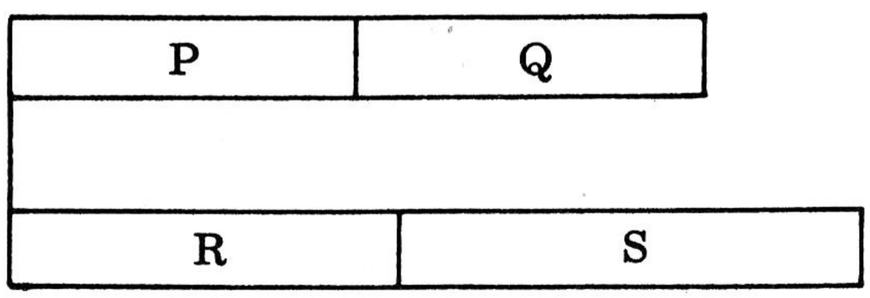
The two diagrams of the previous section have the descriptions

A - (B, C)

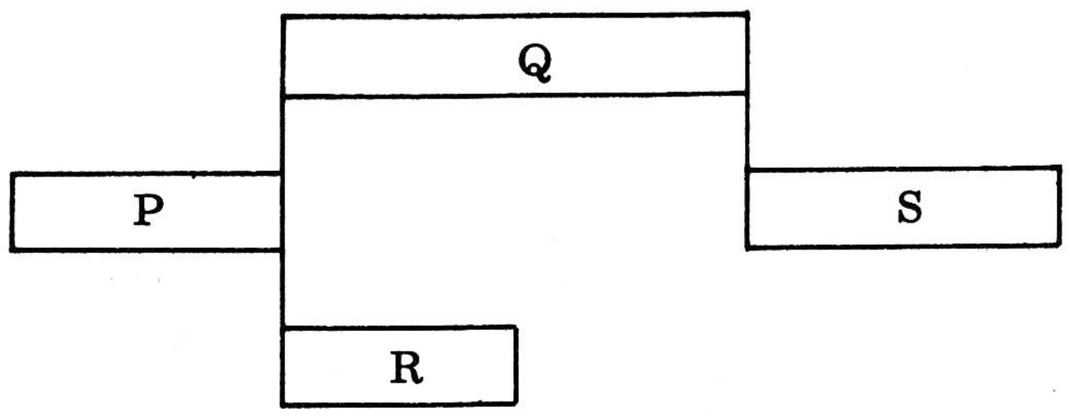
and

A - (B - (X, Y), C)

respectively. As an example of the relative precedence of the "," and "-", consider "P-Q, R-S" which represents



and "P - (Q, R) - S" which represents



Clearly, even the most complex memory structures can be represented easily and in a highly readable fashion.

6.3.3 THE SEG CARD

The SEG operation card has the form

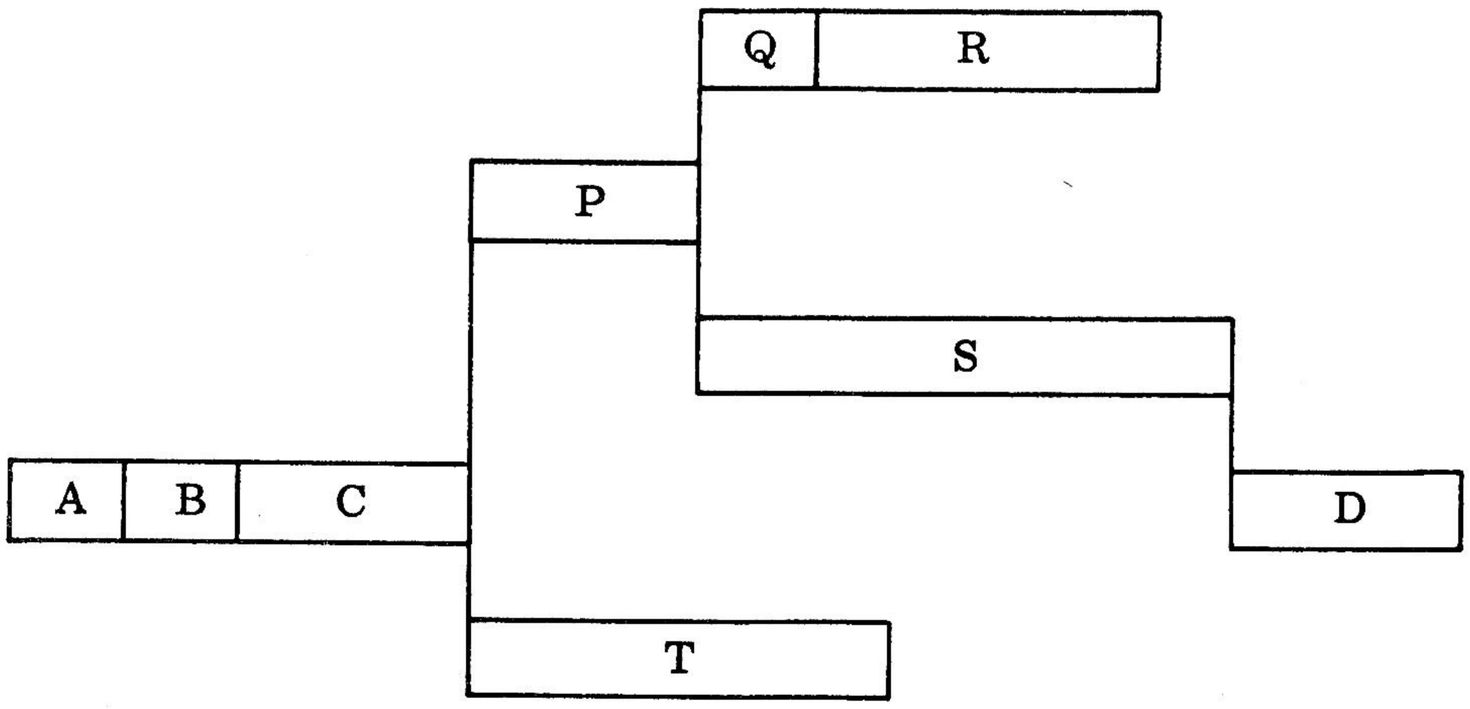
label SEG specifications

in which "label" is the name of the segment being defined, and "specifications" is the algebraic representation of the segment being defined. If the label is omitted, the first name which appears in the specifications portion becomes the name of the segment. The programer is cautioned about the possibility of introducing ambiguities in this way. Any and all subsegments of a segment must be enclosed in at least one level of parentheses. This is tantamount to saying that a comma may not appear at zero parenthesis level.

Any label mentioned which is enclosed in one or more levels of parentheses will be loaded into memory only by calling it. Consider the example

ORBIT SEG A-B-C-(P-(Q-R, S), T)-D

which represents



When the loader is used to load segment ORBIT, then segments (or subprograms) A, B, C, and D are brought into memory. Separate references to the loader must be used to load each of the segments, P, Q, R, X, and T.

Should it be desirable to have Q and R brought into memory by a single reference to the loader, the programer must provide two SEG cards:

```
Z        SEG        Q-R
ORBIT    SEG        A-B-C-(P-(Z,S),T)-D
```

Now, calling for segment Z will load both Q and R. The order in which segments are defined is immaterial.

In contrast to the requirement that a segment which appears enclosed in parentheses must be loaded by a separate call on the loader, a segment which does not appear enclosed in parentheses must not be referenced by the loader. Consider the example

```
SGMT     SEG        X-Y-Z
Y        SEG        A-B-(F,G)
```

The segment Y may not be loaded individually. Indeed, this example has precisely the same effect on the structure of the program as would the single line

```
SGMT     SEG        X-A-B-(F,G)-Z
```

There can still be an advantage to using the two lines, however, since segment Y retains its identity as far as the diagnostic system is concerned. Segments of the type exemplified by Y are called virtual segments.

A subsegment will not necessarily overlay all other subsegments at that same level. Consider the line

```
M        SEG        A-(B,C)-(D,E)
```

B and D are both subsegments of M, yet do not overlay one another.

One further requirement is made for the loading of segments: A subsegment may not be loaded unless that segment to which the subsegment belongs

is currently in memory. Stated another way, a complete path through a tree of subsegments must be present in memory; detached twigs are not allowed.

6.3.4 SUBSEGMENT TABLES

For each segment which itself has subsegments, a table is automatically associated with its fixed part. This table contains a seven-word entry for each of the subsegments. The entry holds information for the loader which describes the action necessary to load the corresponding subsegment and the limits of memory in each bank which are used by the segment. It also contains some pointer addresses to allow the diagnostic system to analyze the current state of a program. Using the information in this entry, it is possible to maintain up-to-date indications as to which segments are currently intact in memory and which were destroyed by the loading of some other segment.

6.3.5 MANUAL AND AUTOMATIC LOADING

Two types of references to the loader are provided by the Monitor System. The first is an explicit request that a segment be loaded. This request is made by the linkage

```
LMJ     11, MLOAD$
        + jump, segment
```

in which "segment" is the name of the segment to be loaded. This half-word actually references the subsegment table entry for the desired segment as discussed in the preceding paragraph. The parameter "jump" indicates the location to which control will be transferred when the loading is complete. The use of an exit for the loader of this form is required since the segment in which the calling sequence resides may be overlayed by the segment being loaded. This type of reference to the loader is called <u>manual loading</u>. Manual loading will result in a segment being brought in from drum whether or not it is currently in core. Thus code may be reinitialized by reloading.

The manual loading linkage is generated by the procedure call

M$LOAD segment, jump

If the second parameter is omitted, control will be returned to the next instruction in line.

The user may indicate that a segment is to be set up for automatic loading. To load a segment automatically, a jump is made to an externally defined symbol within the segment. Any jump type command with the exception of the SLJ is permitted. The address supplied by the allocator for the jump command will not be that of the externally defined symbol. Instead, the jump will be to the load table which accompanies the subsegment table. The load table will in turn jump to the externally defined symbol referenced or will perform an SLJ to a special entrance of the loader. Which one of these two actions occurs depends on whether or not the segment containing the referenced externally defined symbol is currently in core. If it is not, the loader will bring in the required segment and belatedly perform the desired jump. Registers, etc., will be preserved by the loader.

The user may stipulate that a subsegment is to be set up for automatic loading by placing an "*" in front of the name of that subsegment. Thus,

SCAN SEG GETCH-GETWRD-(*OCT,DEC,ALF)

would specify that segment (or subprogram) OCT is to be automatically loaded.

An "*" preceding a left parenthesis indicates that all subsegments at the parenthesis level being introduced will be automatically loaded. For example

SCAN SEG GETCH-GETWRD-*(OCT,DEC,ALF-(DIGIT,SPACE))

indicates automatic loading for OCT,DEC, and ALF but not for DIGIT and SPACE. The effect is precisely the same as writing

SCAN SEG GETCH-GETWRD-(*OCT,*DEC,*ALF-(DIGIT,SPACE))

All externally defined symbols of a subprogram which are defined with respect to control counters of unlabeled groups with group number 1 will receive entries in the load table. (In particular, those externally defined symbols defined

with respect to odd numbered control counters not appearing in INFO lines.) References to any other externally defined symbols will be treated in the usual manner.

6.3.6 INDEPENDENT SEGMENTS AND THE "SUBROUTINE RULE"

A segment which is not made a part of some other segment is said to be independent. The fixed parts of all independent segments will always be in memory as the program is being executed. Memory space is allocated to the independent segments one after another; no overlap occurs.

It is not necessary that all subprograms which will ultimately be a part of the program be explicitly mentioned on SEG cards. When the allocator determines that a subprogram is to be included and the programer has expressed no particular position for that subprogram, it will be assigned memory space according to the subroutine rule:

> The unmentioned subprogram will be attached to the fixed part of that segment which encompasses all references to the subprogram. If the subprogram is referenced only by one other subprogram, it will be loaded whenever the referencing subprogram is loaded. If the subprogram is referenced from more than one independent segment, it becomes an independent segment unto itself. (These latter two statements are actually special cases of the first statement.)

Note that if no map is supplied to the allocator, each subprogram is treated as an independent segment. The subroutine rule applies also to the assignment of dependent labeled common blocks.

6.3.7 CONGRUENT MEMORY ASSIGNMENT

The allocator assigns storage to bank one, bank two, and drum. The same segment description serves for all three types of assignments by means of the device of congruent assignments. This means, simply, that storage requirements for the various elements of a segment will be made in the same pattern and order in all three cases, regardless of the particular amount of storage required of each medium.

6.4 COMMON BLOCKS

Labeled common blocks may be treated in the map language as may elements, in that they may appear on SEG cards. In this respect it is necessary to impose the restriction that the name of a common block not duplicate the name of an element involved in the same allocation.

In order to identify a common block to the memory allocation processor, the BLK operation is used. It has the form:

```
BLK     label,label,...
```

Each "label" is the name of a common block. There is no restriction on the position of a BLK card relative to use of the labels on SEG cards. The inclusion of more than one BLK card has a cumulative effect.

Each processor has its own method of designating common blocks. The processor will associate a length with each common block referenced; the allocator will provide enough storage for each common block to satisfy the maximum of all such lengths. Common blocks not mentioned on SEG cards will be allocated according to their group numbers (see Section 5.9.2), and the subroutine rule.

6.5 DETERMINATION OF MEMORY AREAS

The user may, if desired, give specific directions to the allocator by means of the MAP language operations FIX and SET. The FIX operation is used to determine the precise starting addresses of a segment. A FIXed segment may be mentioned within another SEG operation but it cannot be part of an overlay of that segment. In the example

```
Z    SEG    P-Q-(X, R)
Q    SEG    A-(B, C)
X    SEG    D-(E, F)
Y    SEG    M-N
```

the segments Z, Q and Y may be fixed, whereas the segment X may not.

The FIX operation has the form

```
FIX    name, address 1, address 2, address 3
```

where

"name" is the name of the segment to be FIXed;

"address 1" is the starting address of bank one storage;

"address 2" is the starting address of bank two storage; and

"address 3" is the starting address of drum storage.

At least one of the three addresses must be present. Any of those omitted will be chosen by the allocator. The user is responsible for insuring that impossible memory assignments will not result from the use of the FIX operation. Note that only segments may be FIXed. A segment may, of course, consist of only one element, as in the example

```
Z      SEG    P-Q-(X, R)
       SEG    Q
```

The area of core or drum memory which is to be used for the program being allocated may be altered by the SET operation. Pairs of addresses of the form "fwa/lwa" indicate that memory cells "fwa" through "lwa" are available for allocation. The notation "D fwa/lwa" indicates available drum addresses. The SET

card has the form

SET pair$_1$, pair$_2$, ...

As many pairs may be written for one SET operation as desired, and any reasonable number of SET cards may be given in a MAP. It is the responsibility of the user, however, to assure that the defined areas are unique.

Storage used by a FIXed subprogram or segment will not be removed from the table of available memory within the allocator. If no SET operations appear within a MAP, the allocator will assume that any memory not used by the resident system is available for the object program.

SET operations may be used with chained programs, but only with the restriction that all the SET operations must appear in the first link described. Areas of memory (core and drum) so designated will be used for all links.

6.6 THE DEF OPERATION

The MAP language includes a DEF operation, which is used for the construction of subcomplexes. This will be ignored when the allocator is called by the ABS or the XQT control card.

The DEF operation is used to set up externally defined symbols of the relocatable code element being defined. These externally defined symbols may only be a subset of those available as externally defined symbols of the constituent subprograms.

The DEF operation is written as

$$\text{DEF} \quad \text{def}_1, \text{def}_2, \text{def}_3, \ldots$$

in which each "def_i" may be an external symbol of one of the programs making up the subcomplex which is to be an external symbol of the subcomplex itself. Alternately, "def_i" may be written as

$$\text{name}_1 = \text{name}_2$$

where "name_2" is an external symbol of a constituent element and "name_1" will become an external symbol of the subcomplex. The symbol "name_1" is referenced in lieu of "name_2", i.e., "name_1" is an alias for "name_2".

6.7 MISCELLANEOUS MAP OPERATIONS

Ambiguous external symbol definitions within a program complex file may be resolved by the USE operation. If more than one subprogram in the PCF defines a particular symbol and the selection rules described in Section 5.8.1 do not suffice to determine which subprogram should be included by the allocator, the designation of a subprogram on a USE card will force that particular one to be used. The USE card has the form

```
USE    name/version, name/version, ...
```

It is possible to specify to the allocator any external symbol as the initial instruction of the program (or link in a chain of programs). The operation has the form

```
ENT    name
```

in which "name" is the externally defined symbol to which the system will jump after the program. If no ENT operation card is given for a program or a link, the allocator will use the starting address specified on the end line of an assembled program or the entrance to a FORTRAN generated main program.

6.8 OPTION LETTERS FOR MAP

The following option letters are effective for the memory allocation processor.

A	-	Accept the results of the MAP processing even though errors are detected.
X	-	Abort the remainder of the run if errors are detected.
N	-	Suppress listing. Diagnostic messages will be printed unconditionally.
I	-	Single space listing.
P	-	Produce punched card output for the processed map.
S	-	Produce punched card output for the updated source language.

6.9 ERROR MESSAGES PRODUCED BY MAP

The memory allocation processor detects a large variety of errors in syntax and semantics in the source language cards it receives. These errors are divided into three classes -- fatal errors, non-fatal errors, and warnings.

A fatal error is one from which MAP cannot recover. No output element is produced. A non-fatal error results in the output element being marked in error. A warning message calls attention to some peccadillo found in the source language which is otherwise ignored.

The error messages produced by MAP appear below.

MAP CARD FOLLOWED BY CONTROL CARD

The MAP control card was followed by another control card. No MAP is produced. Fatal error.

THIS CARD DOES NOT BELONG UNDER MAP - CARD DELETED

The card does not fit the specifications of any MAP card and is omitted from the MAP. Non-fatal error.

CHN CARD HAS NO LINK NUMBER

The CHN card has no link number to identify it from other links. Non-fatal error.

FIRST ENT NAME REPLACED

This ENT card is not the first in the link and therefore its name replaces the name from the previous ENT card. Warning message.

THE MAP CANNOT BE WRITTEN TO DRUM

The routine was unable to write the mapped output to drum or enter it into the complex. Fatal error.

THE ELEMENT nnn CANNOT BE FIXED

The name "nnn" is not the name of a level 0 segment. Thus the allocator cannot FIX it. Non-fatal error.

NAME NOT FOLLOWED BY ANYTHING

This FIX card scan routine found the name as the last value on the card, thus having nothing to FIX. Non-fatal error.

END OF FIX TABLE - TOO MANY REFS

More than 166 FIX cards were processed. Fatal error.

VERSION NOT FOLLOWED BY A COMMA

An illegal character terminated a name/version pair (such as a "/" when an entry was omitted) on a USE card. Non-fatal error.

NAME AND VERSION NOT SEPARATED BY A /

An illegal character followed the name of a name/version pair (such as a "," when an entry was omitted) on a USE card. Non-fatal error.

END OF SET TABLE - TOO MANY REFS TO SET

More than 20 sets in any one bank or for drum were processed. Fatal error.

SET VALUES NOT SEPA RATED BY A /

An illegal character followed the first value of a SET pair. Non-fatal error.

ERROR IN SECOND DRUM ADDRESS

When setting a drum address using a SET card, the second value of the pair is begun by a non-numeric character (such as another "D"). Non-fatal error.

END OF BLK TABLE - TOO MANY BLKS

More than 30 blocks are defined. Fatal error.

nnn BLK NAME NOT DEFINED IN A SEG

The name "nnn" named on a BLK card was not named also as an element. Non-fatal error.

THE BLOCK nnn IS A DUPLICATE

The name "nnn" named on a BLK card was named as a block earlier in this link. Warning message.

END OF SEG NAME TABLE - TOO MANY SEGS

There are too many "loads" in this MAP. The number of segments named on SEG cards and elements requiring separate loads exceed 50 (in the 32K version, and 100 in the 65K version). Fatal error.

TOO MANY ELEMENTS - MAP QUITS

The allocation table is too large. (A maximum of 429 elements are allowed for 32K, 858 elements for 65K). Fatal error.

THE SEGMENT IS NAMED TWICE - SECOND SEG DELETED

Two segments have the same name. Only the first is accepted. Non-fatal error.

A COMMA FOLLOWS A p

A comma follows the character "p" (such as a slash, parenthesis, comma, etc.) which creates an illegal segment construction. Non-fatal error. The card will be processed but deleted when the processor is finished searching for further errors.

BLANK FOLLOWS NAME - UNBALANCED PARENTHESES

A blank has terminated the card scan but the parentheses are still not balanced. The scan will continue to search the rest of the card to try to rectify the condition. Warning message.

PARENTHESES UNBALANCED

End-of-card condition and the parentheses aren't balanced. Non-fatal error. Card is deleted.

A RIGHT PAREN FOLLOWS A p

Indicates an illegal segment construction. Non-fatal error. When the rest of the card has been processed, the card will be deleted.

TOO MANY RIGHT PARENTHESES

Indicates an illegal segment construction. Non-fatal error. When the rest of the card has been processed, the card will be deleted.

SEGMENT HAS SAME NAME AS NON-LEADING ELEMENT

Both the segment and a non-leading element have the same name. Non-fatal error.

TWO ELEMENTS HAVE THE SAME NAME OF nnn

The MAP contains two elements with the name "nnn". The second one is not used. Non-fatal error.

THE SEG HAS AN * AT FIRST LEVEL

Warning message for extraneous "*" on elements of automatic load conditions when the element is part of a larger load.

EXTRANEOUS PARENTHESES

Extraneous parentheses were used on a SEG card. They are automatically ignored. Warning message.

SEG DELETED

Follows a message which describes an error condition on a SEG card whenever the segment is to be deleted. Non-fatal error.

THE MAP INPUT EXCEEDS MAP TABLES - MAP QUITS

The segment description tables are exceeded by MAP input. Fatal error.

COMMA AT FIRST LEVEL

A comma was detected by the card scan routine without having first encountered a parenthesis. Non-fatal error.

DUPLICATE LINK NUMBER

Two CHN cards have identical link numbers. Non-fatal error.

ALPHABETIC SYMBOL DETECTED - DRUM FIX OMITTED

While processing a FIX card, the drum address was begun by an alphabetic character. Non-fatal error.

NO SEGMENTS IN THIS LINK - LINK OMITTED

Two consecutive CHN cards were encountered. Non-fatal error.

CORRECTION CARD IN ERROR

The above correction card is not correct. Fatal error.

NAME NOT FOLLOWED BY A COMMA

DEF card name not followed by "," or "=", but not blank. Non-fatal error. Last DEF is omitted.

DEF TABLE EXCEEDED

Too many DEF entries. Non-fatal error.

SET ADDRESS TOO LARGE

A SET address for bank 2 was larger than 177777. Non-fatal error.

VII. THE AUXILIARY PROCESSORS

7.1 THE PROCEDURE DEFINITION PROCESSOR

7.1.1 GENERAL

The procedure definition processor, PDP, accepts source language defining SLEUTH II procedures and builds an element to be included in the program complex file. These procedures may subsequently be referenced in assemblies precisely as though the definitions for them accompanied the source language input to the assembler. Both the user's and the library PCF may contain procedure elements. If a given procedure is referenced, the assembler will use any definition of that procedure included in its input source language. If none is present, the user's PCF is interrogated for a definition. If none is found there, the library PCF is interrogated. No error is indicated during assembly unless none of the three sources provides a definition. Procedures processed by PDP may not be used to redefine SLEUTH II directives.

7.1.2 THE PDP CONTROL CARD

The procedure definition processor control card has the general form

$$\nabla \text{options} \quad \text{PDP, loc} \quad n_1/v_1, n_3/v_3$$

where

"n_1/v_1" is the name and version of the source input element, and may be either a source language element (type 1) or a procedure element (type 7);

"loc" indicates whether the element is to be taken from cards ("loc" is omitted), from drum ("*"), or from tape (alphabetic character referencing the proper tape unit);

"n_3/v_3" is the name and version to be applied to the procedure element code resulting from this processing (if omitted, the procedure element produced by PDP will be entitled "n_1/CODE");

"options" may contain the following:

L - Produce a complete listing of the source language with error flags and line numbers appended. (Error flags have the same meaning as in Sleuth II assemblies.)

A - Accept the results of the processing as correct even though errors were detected.

X - Abort the remainder of the run if any errors are detected by the processor.

I - Produce single-spaced listing.

If the element is taken from the tape, corrections may be read from cards and applied to the input as the processor operates (see Section 5.5 for details). Since the output of PDP may be reprocessed, this precludes the necessity of the updated source language element required for processors such as FORTRAN or MAP.

7.1.3 PROCEDURE NAMES IN THE PCF

The program complex file's table of contents contains a table of all procedure names currently part of the PCF. All procedure elements in the PCF contribute to this table. If a procedure element is added to the PCF which defines a procedure name already represented in the PCF, the former reference is deleted. The element itself, however, is not deleted.

Those procedure names entered into the table of contents are precisely those which would be made available to an assembly were the procedure definitions included with the assembly source language.

7.2 THE COBOL LIBRARY PROCESSOR

7.2.1 GENERAL

The COBOL library processor, CLP, accepts COBOL source language defining file descriptions and record descriptions from the data dimension, and paragraph names from the procedure dimension. This source language is included in a COBOL compilation by means of the COPY verb.

7.2.2 THE CLP CONTROL CARD

The COBOL library processor control card has the general form

$$\nabla \text{options} \quad \text{CLP, loc} \quad n_1/v_1, v_3/v_3$$

The pair "n_1/v_1" indicates the name of the source language input element which may be taken from cards, drum, or tape depending on "loc". "n_1/v_1" may designate either a type 1 or a type 6 (COBOL library) element.

The pair "n_3/v_3" provides the name of the resulting COBOL library element. If omitted the resulting element titled "n_1/CODE".

Option letters L, A, I, and X are applicable to CLP. These options are the same as for the PDP control card.

VIII. OPERATING INSTRUCTIONS

8.1 GENERAL COMMENTS

This chapter deals with that portion of the operating instructions which directly concerns the EXEC II System. No attempt is made to describe hardware functions such as the threading of Uniservos or adjustments to the printer. Also, no attempt is made to recommend the machine room procedures and disciplines so necessary to a smoothly functioning installation. These topics are best left to other documents.

8.2 TAPE BOOTSTRAP ROUTINE

The EXEC II Load Tape (COSM) has as its first block a 224-word Bootstrap Routine which serves to bring in the remainder of the resident and various other parts of the system. The Bootstrap Routine also provides a simple card load routine, a panic dump, and a method of patching the resident system prior to writing it to drum.

8.2.1 SYSTEM SETUP

To read the system into an empty machine, clear all pertinent channels and mount the COSM Tape on the lowest-numbered tape channel, unit 0. All console jump switches should be off, and the interlock disable switch on the drum synchronizer should be set ON. The real-time clock should be off. Place the setup deck in the card reader and depress the card reader RUN button. Perform a manual bootstrap operation from tape.

The COSM Tape will now be read until the resident and the control card interpreter are in core. A jump will then be made to the system routine which writes these routines onto drum. A drum bootstrap is simulated and the system is now under normal operation. The setup deck is read and the remainder of the system is read from the COSM Tape and transferred to drum. A log of the setup operations is produced on the printer and a message is typed out on the console to verify that the system setup was accomplished properly.

The interlock disable switch on the drum synchronizer should now be returned to the OFF position.

8.2.2 CARD LOADER

The Bootstrap Routine may be used as a simple card loader by performing a manual bootstrap operation with console jump switch 15 set.

Cards will be read and their contents put into memory. Any part of core, with the exception of octal locations 000200 through 000337, may be loaded from cards in this manner.

Four card formats are recognized:

a. Single Word Octal Card

Cols.	1 - 6	Address
	7 - 8	Blank
	9 - 20	Word to be loaded

b. Multiple Word Octal Card

Cols.	1 - 6	Address of first word
	7	Blank
	8	Word count (1 to 6)
	9 - 20	First word
	21 - 32	Second word
	33 - 44	Third word
	45 - 56	Fourth word
	57 - 68	Fifth word
	69 - 80	Sixth word

c. Instruction Word Card

Cols.	1 - 6	Address
	7 - 8	Blank
	9 - 10	F field
	11 - 12	J field
	13	Blank
	14 - 15	A field
	16	Blank
	17 - 18	B field
	19	Blank
	20	H and I bits (0 to 3)
	21	Blank
	22 - 27	M field

d. Jump Card

Cols. 1 - 8 Blank

9 - 14 Address

The jump card terminates the card read operation and executes a jump to the address given in columns 9 through 14.

8.2.3 PANIC DUMP ROUTINE

A manual bootstrap operation with console jump switch 14 on will initiate a simple dump routine. The computer will halt. Clear the P register (do not use Master Clear or Computer Clear) and enter the address of the first word of the area to be dumped. Press START. Again the computer will halt. Clear the P register and enter the address of the last word of the area to be dumped. The requested area will be formatted and put out on the printer. After the dump is complete, the dump routine returns to the first halt mentioned above to accept further parameters.

The dump is formatted six words to the line with the address of the first word on the line at the far left. The occurrence of two asterisks between this address and the first word indicates that the information on the previous line was duplicated and was thus omitted from the printed output, i.e., duplicate lines are omitted.

The contents of registers B and A at the time the dump was entered are moved into octal locations 000130 and 000147 and 000144 through 000163, respectively. The R registers are not disturbed by the dump and thus may be read from addresses 000100 through 000117.

8.2.4 SYSTEM PATCH ROUTINE

Patches may be made to the resident system and to the control card interpreter by performing a manual bootstrap operation with console jump switch 13 on. This will result in reading the resident, etc., into core and

transferring to the card read routine described above. The deck of patches should always terminate with a jump card referencing address of 000005. These patches must precede the system setup deck.

8.2.5 MISCELLANEOUS INFORMATION

If the Bootstrap Routine is currently in core, any of the above routines may be entered by depressing MASTER CLEAR, setting the desired jump, and depressing START twice. (Cell 000000 contains HJ 000001.)

If an interlock interrupt is received from the card reader, printer, or tape unit, the typewriter will announce

INLK

and the computer will stop. Remove the source of the interlock (depressing RESUME if the card reader was involved) and depress START. The routine will continue. An occurrence of an input/output error interrupt, a real-time clock interrupt, or an internal error interrupt will cause the typewriter to announce

ERR

and the computer will stop. Depressing START will cause the Bootstrap Routine to be re-entered from the beginning.

8.3 DRUM BOOTSTRAP ROUTINE

Once the system has been set up as described in the previous section, it may be reinitiated by performing a manual bootstrap operation from the appropriate drum channel. No console jump switches are effectual for a drum bootstrap.

Whenever the system has been bootstrapped from either drum or tape, date and time key-ins should be performed and the real-time clock turned on. In addition, the printer forms should be adjusted to print on physical line one.

8.4 BASIC UNSOLICITED KEY-INS

The primary control exercised by the operator over the Monitor System is by means of unsolicited key-ins. An unsolicited key-in is one made that is not in response to a specific request by the running program. The first character of the key-in always determines the routine which is to process the information entered into the computer. What follows this first character depends on the specific type of key-in.

All unsolicited key-ins are terminated with a carriage return. Once started, a key-in may be discontinued by depressing the SPEC key. This will terminate the key-in and return control to the program being executed.

If the system is unable to recognize an unsolicited key-in or a recognized key-in is in error, the typewriter will announce

KEY ER

and the key-in is ignored. The program or parasites running are not affected.

8.4.1 THE E AND X KEY-INS

Unsolicited key-ins of E or X are used to terminate the program currently being run executed. The E key-in forces an error (MERR$) termination; the X key-in forces an abort (MXXX$) termination. In the former case the program will take any dumps specified by the programmer before beginning the next job. In the latter case the run is immediately terminated.

The E key-in is normally used in situations such as the program entering a closed loop. An X key-in is usually used when the program reaches some impasse such as a request for a nonexistent tape reel or such like.

8.4.2 THE D AND T KEY-INS

An unsolicited key-in of the form

DΔdd/mm/yy

is used to enter the current date into the system. The fields "dd", "mm", and "yy" represent the day, month, and year respectively.

In a similar manner, a key-in of the form

TΔhh:mm

is used to set the system clock to the current time of day. A twenty-four hour clock notation is used. The fields "hh" and "mm" represent the time in hours and minutes respectively.

The date and time key-ins must be used each time the system is brought in from tape or drum. The operator should start the real-time clock just before performing a time key-in.

8.4.3 THE S KEY-IN

The S key-in is used primarily to cause the system to continue after having delayed. Such a delay is always announced by the typewriter message

WAIT

and may be caused by

a. The program running;

b. An MSG control card with an "H" option;

c. The occurrence of a new run when the system is in the "stop between jobs" mode; or

d. An unsolicited key-in of W (see below).

If for some reason the operator chooses not to continue the current job, he may also remove the wait condition by a key-in of E or X. Any unsolicited key-in may be made while in the wait condition.

The "stop between jobs" mode may be forced by the operator by a key-in of

S ON

This will cause a wait condition to occur after the contents of a RUN control card have been displayed on the typewriter. The "stop between jobs" mode may be discontinued by typing

S OFF

If neither ON nor OFF accompany the S key-in, it is assumed that the operator is continuing after a wait condition.

8.4.4 THE W KEY-IN

The operator may force a wait condition by a key-in of

W

causing the message WAIT to be typed and a delay to occur. In this and any other wait condition, only the user's program is delayed; any parasite operation continues. For this reason it is usually preferable to use a W key-in rather than the STOP button.

8.4.5 THE LF KEY-IN

Approximately four inches of paper from the typewriter may be spaced up by an unsolicited key-in of LF (line feed). This allows sufficient room for the paper to be conveniently torn off.

9.4.6 THE A KEY-IN

Magnetic tape assignments are made by an unsolicited key-in. For a description, the operator is referenced to Section 2.2.2.

8.4.7 THE L KEY-IN

An unsolicited keying of the form

LΔnnΔmm

will cause the printer on channel "nn" to be spaced forward "mm" lines. This adjustment is independent of the spacing control provided for the user's output. The L key-in is intended to allow for sporadic clutch failure or for operator misadjustment of the printer forms. The value of "mm" is restricted to a maximum of 33.

8.5 UNSOLICITED KEY-INS FOR INPUT/OUTPUT CONTROL

The unsolicited key-ins C, R, and F are used to exercise control over input/output operations which did not function normally. They are used to respond to type-outs produced by the system. The proper response depends on the kind of input/output device involved and the particular message produced.

Each of the key-ins is of the form

λΔn

or

λΔnn

where "λ" represents C, R, or F and "n" or "nn" is the channel number (in decimal). For example

F 6

and

C 10

indicate an F key-in directed to channel six, and a C key-in directed to channel ten, respectively.

The messages produced by the various input/output devices and the appropriate responses are described in the paragraph below.

8.5.1 THE CARD READER

If the card reader fills a stacker or if the last card detected is not a properly punched FIN control card, the message

CARD INTLK nn

will be typed. The field "nn" here and below represents the channel number of the card reader. If there are more cards to be read, put them in the hopper, depress RESUME, and respond

C nn

The reader will continue reading cards. If there are no more cards to be read, a response of

F nn

will have the same effect as a FIN control card.

If the card reader fails to read a card properly the message

VERIFY ERROR nn

will be typed. The card in error and two* others will have been dropped in the error stacker. Two operator actions are possible: the card image may be accepted "as is" and reading continued, or an attempt may be made to reread the card. In the former case remove the cards from the error stacker, put them in the normal stacker, depress RESUME, and respond

C nn

on the keyboard. In the latter (and recommended) case, remove the cards from the error stacker, place them in front of any cards remaining to be read, and return them to the hopper. depress RESUME Respond with

R nn

and then the card reader will attempt to reread the card.

If an invalid character code is read, the typewriter will announce

CODE ERROR nn

The card in error and three** following cards will be in the normal stacker. The operator may accept the image as is (bad columns being input as 008) by responding

C nn

or, if feasible, the card may be corrected on the spot. If the choice is to correct the card, remove the last four cards from the stacker and repunch the card in error.

---

*If there are less than two cards following the card in error, this number will be equal to the number of cards remaining in the deck.

**If there are less than three cards following the card in error, this number will be equal to the number of cards remaining in the deck.

Replace the corrected card and the three following it in front of any cards remaining to be read and respond

R nn

The message

CARD FAULT nn

indicates that the card reader is inoperative. In general, no error recovery is possible. Allow any work on the machine to terminate and call the Field Engineer.

8.5.2 THE CARD PUNCH

If the card punch exhausts its supply of blank cards, fills a stacker or fills its chip box the message

PUNCH INTLK nn

is typed. Correct the difficulty, depress RESUME and key in

C nn

The card punch provides automatic recovery from verification errors. In doing so cards will be dropped in the error stacker. These cards are of no significance and should be discarded by the operator.

A message

PUNCH FAULT nn

indicates that the card punch is inoperative. Allow any work on the machine to terminate and call the Field Engineer.

8.5.3 THE PRINTER

If a printer interlocks (out of paper, carriage out, etc.) the message

PRINT INTLK nn

is produced. Remove the cause of the interlock and respond

C nn

or

R nn

depending on whether the paper was left in its position at the time of the interlock or was repositioned to line one in the process of removing the interlock.

8.5.4 MAGNETIC TAPE

If a magnetic tape unit is addressed by a program and there is no reel mounted, no power has been applied, or (for write operations) there is a write inhibit condition, the typewriter will announce

TAPE INTLK nn/uu

where "nn" is the channel number and "uu" is the unit number (both in decimal). If the interlock condition can be removed, then the operator should do so and respond

C nn

If not, the response

F nn

will inform the running program that the requested operation could not be performed. What effect this has depends entirely on the program.

In some cases improper operation of a tape unit can be recovered. The system itself attempts to make the recovery an appropriate number of times. If these recovery cycles are unsuccessful, the message

TAPE ERROR nn/uu

is typed. The operator may respond with

R nn

which signals the system that another attempt should be made to perform the tape operation; with

C nn

which indicates that the bad tape block should be used "as is"; or with

F nn

which indicates that the program should be informed that the requested operation could not be executed.

The message

TAPE FAULT cc/uu

indicates that the requested operation on channel "cc" of unit "uu" could not be performed.

8.5.5 GENERAL COMMENTS

The operator must always respond to ERROR and INTLK messages. A FAULT message does not require response. The Field Engineer on duty should always be informed when a FAULT message occurs. If tapes are involved he will probably be interested in the reel number of the tape and the physical unit causing the trouble. Following a FAULT message will be a 12-digit octal number which represents the contents of the external status word which signaled the malfunction. The Field Engineer will likely want to know its value.

8.6 UNSOLICITED KEY-INS FOR SYMBIONT CONTROL

User programs in the 1107 Monitor System are controlled primarily by means of a card deck. Symbionts, on the other hand, must receive all of their control through the operator's keyboard. This section describes the key-ins used for the control of symbionts.

8.6.1 THE BASIC FORMAT

All key-ins addressing symbionts have the form

.pppΔsΔmmmmmm

where "ppp" is the name of the symbiont addressed, "s" is a single character called the signal, the "mmmmmm" is a six-character field called the message. When the message field is not required by a particular parasite, it is simply omitted from the key-in.

A key-in (other than I, see 8.6.2) addressing a parasite which is currently not in operation will always produce the response

-ppp NOT ACTIVE

8.6.2 INITIATING A SYMBIONT

In order to initiate a symbiont named "ppp" a key-in of the form

.ppp I

is used. Some symbionts will also require an accompanying message field. The system will either initiate the requested symbiont, or will respond with one of the following typeouts:

-ppp LATER

which indicates that symbiont "ppp" cannot be initiated now;

-ppp ALREADY ACTIVE

which indicates that symbiont "ppp" is currently operating; or

NO SYMBIONT NAMED ppp

which indicates that the system never heard of parasite "ppp".

8.6.3 CARD-TO-DRUM SYMBIONT -- CR1

The card-to-drum symbiont, CR1, is used to read cards and file the images away on drum. It may be initiated as indicated in 8.6.2 or by an unsolicited key-in of "I".

Once initiated, the card-to-drum symbiont will read as many runs as are stacked in the card hopper. It is not necessary to initiate the symbiont for each run. When CR1 detects a FIN control card (see section 3.1.1) as the last card of a deck it terminates with the message

-CR1 TERMINATED

Reinitiation is necessary to continue reading cards.

The operator may temporarily suspend the card-to-drum symbiont by means of the key-in

.CR1ΔS

which evokes the response

-CR1 SUSPENDED

When it is desired to continue, the key-in

.CR1ΔC

will inform the symbiont to go ahead.

The card-to-drum symbiont writes the card images onto drum a few at a time. As soon as a "block" of card images has been read, those cards are immediately available to the program running. Thus it is possible for the computer to catch up with the card-to-drum symbiont. When this occurs the user's program is forced to wait on the symbiont with a resulting increase in apparent running time.

8.6.4 DRUM-TO-PRINTER SYMBIONT -- PR1

The drum-to-printer symbiont, PR1, serves to transmit to the printer those line images stacked on drum by user programs. PR1 is a self-initiated symbiont; it will begin printing as soon as the user program has constructed the first block of line images. The operator may lock out self-initiation by means of the key-in

.PR1ΔL

If PR1 is active, it will continue printing until the current print file is exhausted and then terminate. If not active, the message

-PR1 NOT ACTIVE

will be produced. In either case the symbiont will not be reactivated until the operator provides the key-in

.PR1ΔI

The usual reason for locking out PR1 is to enable some other symbiont -- such as a tape-to-printer symbiont -- to use the printer. Should the operator allow more than one symbiont to drive the printer simultaneously, lines from both files will be interspersed on the printout.

The drum-to-printer symbiont may be suspended by the key-in

.PR1ΔS

and will continue from where it left off upon receiving

.PR1ΔC

If while PR1 is suspended it is necessary to move the paper, adjust it to print on physical line one. Then the "recover" key-in

.PR1ΔR

should be used to begin printing again. This key-in will cause the symbiont to restart from the beginning of the page it was on when the interruption occurred.

When PR1 is in control of the printer and a print interlock occurs (see 8.5.3) the key-in

R nn

will also effect recovery of the last page begun.

The operator may inhibit the printing of the remainder of a file of print images by means of the key-in

.PR1ΔX

This will result in bypassing the rest of the print file (producing a noticeable delay if the file is long) and then either terminating or printing the next file waiting for the printer.

8.6.5 DRUM-TO-PUNCH SYMBIONT-- CP1

The drum-to-punch symbiont, CP1, serves to transmit to the card punch those card images stacked on drum by user programs. The CP1 response to the S, C, X, L, and I signals is analogous to PR1. The R signal is not meaningful.

8.7 TYPEWRITER MESSAGES PRODUCED BY THE MONITOR

During the execution of programs under the EXEC II System there are various messages produced on the typewriter. The operator should be familiar with these and appropriate responses, if any.

9.7.1 RUN hh:mm

A new run is beginning. The field hh:mm represents the current time of day in hours and minutes. The next line typed is an image of the run control card (see Section 2.1.3).

8.7.2 MSG:

A message control card has been encountered. The next line typed is an image of the control card (see Section 2.1.4).

8.7.3 WAIT

The system is in a wait condition pending operator action. An unsolicited key-in of S removes the wait condition (see Sections 8.4.3 and 8.4.4).

8.7.4 ERROR

The current program has encountered an error condition. Dumps will be taken if requested, but the remainder of the job will not be processed (see Section 2.3.2).

8.7.5 ABORT

The current program has encountered an abort condition. The remainder of the job will not be processed (see Section 2.3.2).

8.7.6 MAX TIME

The elapsed time for this job has exceeded the maximum specified on the RUN control card. It is the operator's responsibility to kill the run (see Section 2.1.3).

8.7.7 MAX PAGES

The number of pages produced by this job has exceeded the maximum specified on the RUN control card. It is the operator's responsibility to kill the RUN (see Section 2.1.3).

8.7.8 KEY ER

An improper unsolicited key-in was given by the operator. It is ignored (see Section 9.4).

8.7.9 TAPE ERROR nn/uu
CODE ERROR nn
VERIFY ERROR nn

A potentially correctable input/output error has occured. In the case of tape, a series of recovery attempts has already been made. The field "nn" indicates the channel and "nn" the unit involved. The operator should respond by the C, R, or F insolicited key-ins (see Section 8 5).

8.7.10 CARD INTLK nn
PUNCH INTLK nn
PRINT INTLK nn
TAPE INTLK nn/uu

An interlock condition has occurred. It should be remedied if possible and a response made with the C, R, or F unsolicited key-ins (see Section 8.5).

8.7.11 CARD FAULT nn ssssssssssss

PUNCH FAULT n ssssssssssss

TAPE FAULT nn/uu ssssssssssss

DRUM FAULT nn ssssssssssss

An uncorrectable input/output error has occurred. No response is required. The field "ssssssssssss" is the status word indicating the nature of the fault (see Section 8.5).

8.7.12 NSI ON CH nn

An unexpected or "non-sense" interrupt has occurred on channel nn. It may be due to a program bug, to hardware malfunction, or to the fact that some input/output device is undergoing maintenance. Usually no damage is done. No response is required.

8.7.13 LABEL IN USE

SERVO IN USE

LATER

cc/un EMPTY

These messages are produced in conjunction with the tape assignment (A) key-in (see Section 2.2.2).

8.7.14 END RUN

The current job has terminated. No response is required.

8.7.15 BEGIN IDLE

END IDLE

These messages delimit idle periods. The parasites may still be functioning, but all jobs entered into the system have been completed.

8.7.16 Any message beginning with "-"

These messages are concerned with parasite control (see Section 8.6).

8.7.17 MONITOR

This message indicates that the Monitor has been bootstrapped from drum. No response is necessary (see Sections 8.2.1 and 8.3).

8.7.18 ERR

This message occurs when an error condition is detected during bootstrapping or a panic dump. Restart (see Section 8.2.5).

8.7.19 INLK

This message occurs when an interlock condition is detected during bootstrapping or a panic dump. Remove the cause of the interlock and hit START (see Section 8.2.5).

8.8 SUMMARY OF UNSOLICITED KEYINS

A	Magnetic tape assignment (2. 2. 2)
B	Not used
C	Continue (8. 5)
D	Date (8. 4. 2)
E	Force error termination (8. 4. 1)
F	Fault or "FIN" (8. 5)
G	Not used
H	Not used
I	Initiate card read parasite CR1 (8. 6. 3)
J	(Reserved for additional card read parasite)
K	Not used
L	Not used
M	Not used
N	Not used
O	Not used
P	Not used
Q	Not used
R	Recover (8. 5)
S	System control (8. 4. 3)
T	Time of day (8. 4. 2)
U	Not used
V	Not used
W	Wait (8. 4. 4)
X	Force abort termination (8. 4. 1)
Y	Not used
Z	Not used
	Parasite control (8. 6)

IX. CONSTRUCTION OF SYMBIONTS

9.1 GENERAL COMMENTS

9.1.1 THE ROLE OF A SYMBIONT

A symbiont should be envisioned as a small, multiprogramed routine whose duty is to control an input/output device having a low information transfer rate compared to the rest of the computer. Its primary function is to allow the use of tape or drum buffers to match the high speed but intermittent requirements of the central computer to a slow but continuous transfer of information to "peripheral" devices. It is possible to use the facilities of the EXEC II System which are concerned with symbionts to perform more complex operations than peripheral simulation under concurrent processing. In such cases, however, the EXEC II System makes no attempt to provide such features as core and drum space assignment normally thought of as part of a multiprograming facility.

As detailed in previous chapters of this manual, the "user" program is card controlled and has at its disposal all but certain fixed portions of the available hardware. A symbiont, on the other hand, is controlled by means of the console typewriter and has only limited facilities available. The user program is given the opportunity to engage in several input/output operations simultaneously with the execution of internal operations; a symbiont is explicitly denied this ability. The user program is subject to being interrupted at any time in favor of one or more symbionts able to make use of the central computer; a symbiont will never be so interrupted, either by the user or by another symbiont. However, whenever any input/output operation is initiated by a symbiont, it gives up all claim on the central computer until that input/output operation is complete.

9.1.2 THE MACHINE ENVIRONMENT OF A SYMBIONT

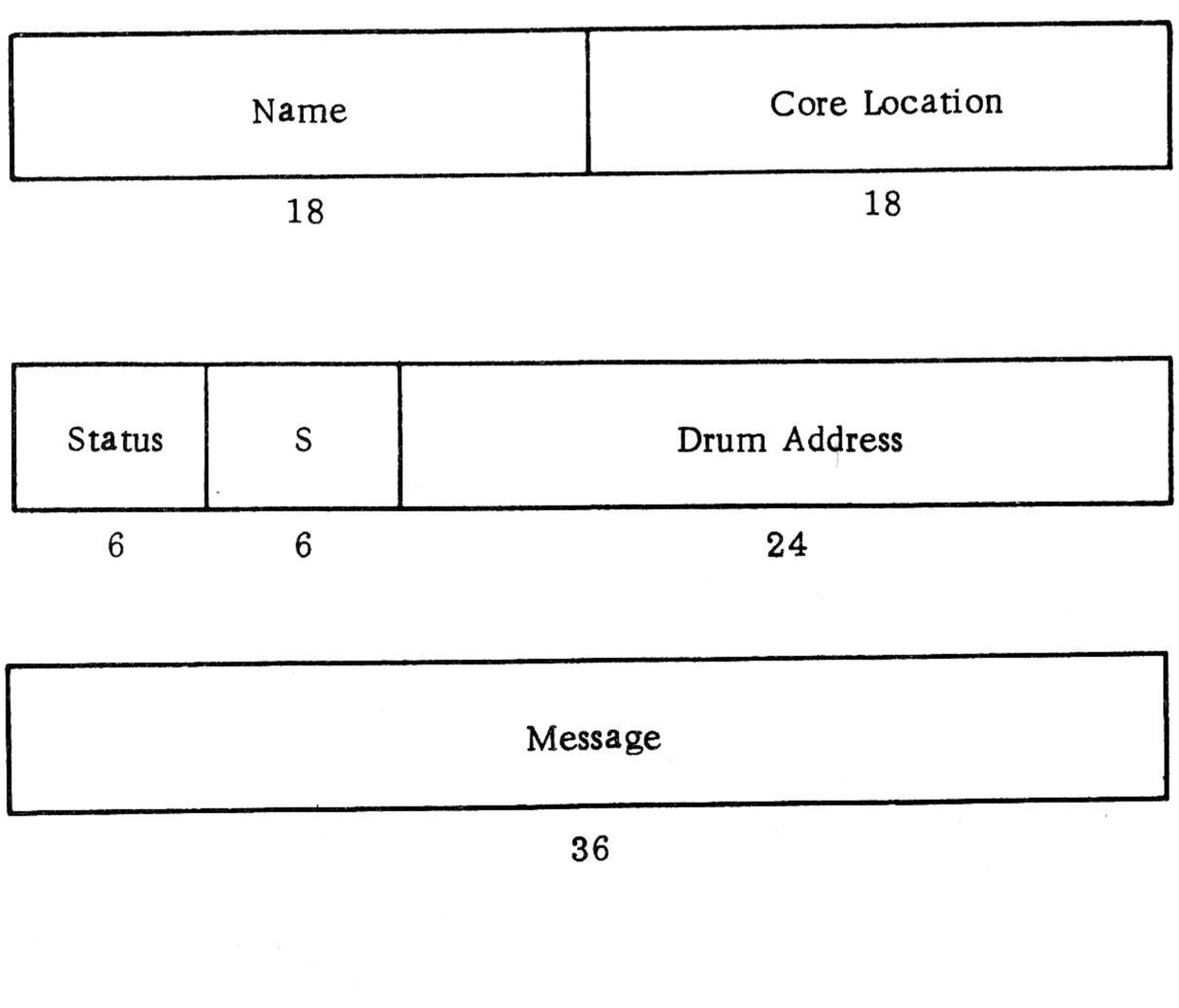
When a symbiont is in control of the central computer it may freely use all of the volatile registers (B11 through A5 and R1 through R3) without

saving and restoring them. Other registers used must be saved and restored between input/output operations and calls to the subroutines ZRCB$, ZLCB$, ZRDB$, ZLDB$, ZSPND$, ZTERM$ (see below). Memory lockout is always released.

A symbiont is not relocated when it is loaded into core; it exists as an absolute program on drum which is called "as is" into available core space. This situation requires that a symbiont be written as a "self-locating" program. The location of the first instruction of a symbiont is always placed in index register B1 whenever it is operating. Thus, all references of a symbiont to itself (e.g., jumps or references to literals) must be modified by the contents of B1. Once assigned for a given execution of a symbiont, this value will not change. Hence, it is safe to compute addresses using the quantity in B1 and modify the symbiont program with them.

Buffer areas are available to a symbiont which may be used as desired for input/output areas or, perhaps, for general working space. These core buffers are 256 words long. The symbiont program itself occupies one of these buffers and is thus restricted to a maximum length of 256 words.

For each symbiont in the system there is a three-word entry in a resident table called ZPT$. These entries have the form



where

"Name" is the three-character code which identifies the symbiont;

"Core Location" is the location of the core buffer currently containing the symbiont program;

"Status" is a flag to indicate the current condition of the symbiont (0 - inactive, 1 - suspended, 2 - active);

"S" is the signal character (see below);

"Drum Address" is the drum location of the symbiont program; and,

"Message" is a six-character message to be given the symbiont (see below).

Whenever a symbiont is operating, the cells ZSW$ will contain the (non-zero) index into ZPT$, which locates the entry for that symbiont.

9.1.3 OPERATOR-SYMBIONT COMMUNICATIONS

A symbiont may use the KTYPE$ routines for messages to and from the console and the resident editing routines to construct outgoing messages. It is suggested that the first five characters of any outgoing message be

-xxxΔ

where "xxx" is the name of the symbiont producing the message. All messages concerning symbionts produced by the system itself follow this convention.

As well as using the KEYIN$ subroutine, there is provided an asynchronous message input facility (see Section 8.6). An unsolicited key-in addressing a symbiont is of the form

.xxxΔSΔmmmmmm

where

"xxx" is the name of the symbiont addressed;

"S" is a single character called the "signal"; and

"mmmmmm" is a six character field called the "message".

If the message field is omitted on a key-in, it is set to spaces. The signal and message are stored into ZPT$ whenever a "." unsolicited key-in

is received and may be examined by a symbiont at appropriate points in its processing cycle.

If the system is unable to recognize the symbiont named, the message

NO SYMBIONT NAMED xxx

is typed and the key-in ignored. With the exception of an "I" signal, the symbiont must be currently active in order to receive a key-in. If it is not, the message

-xxx NOT ACTIVE

is typed and the key-in ignored.

The system itself recognizes the signal "I" which indicates that a symbiont is to be initiated. The symbiont loader is called into action, the indicated symbiont program is read from drum into a core buffer, and entered. If, for some reason the loader is unable to initiate the desired symbiont at this time, the message

-xxx LATER

is typed and the key-in ignored. If there is a drum failure in attempting to load the symbiont being initiated, the message

-xxx LOAD ERROR

is typed and the key-in is ignored. If the symbiont indicated is already active, the message

-xxx ALREADY ACTIVE

is typed and again the key-in ignored.

"L" indicates that the device is inoperative and the symbiont should not reference the unit.

9.2 THE SYMBIONT CONTROL ROUTINES

In addition to those portions of the dispatcher and interrupt control routines concerned with symbionts, the resident contains a collection of subroutines called the the symbiont control routines. These subroutines perform utility functions for symbionts and provide the required symbiont-system linkages. Some of these subroutines may also be called from the user's enviornment and if so called may act slightly differently than when called from symbiont environment.

9.2.1 CORE BUFFERS

A core buffer may be requested by the linkage

```
LMJ          11, ZRCB$
```

which leaves the address of the core buffer in A0. If the call is from the user's environment and no core buffer is currently available, the subroutine will return with A0 negative. The subroutine will never return to a symbiont until a core buffer is indeed available.

A core buffer is released by placing its address in A0 and executing

```
LMJ          11, ZLCB$
```

Even if only a single buffer area is required it should be released whenever emptied and re-requested before putting new information in it. This allows the same buffer area to be shared between several requesting routines, thus allowing them to continue in the face of a shortage of core buffer area.

The subroutines ZRCB$ and ZLCB$ may be used from interrupt coding.

9.2.2 DRUM BLOCKS

An area of drum is set aside for use by the symbiont routines. Its size is installation selectable. This drum area is broken up into blocks of 256 words. A table indicating which of these drum blocks currently contains

information is maintained by the resident. The normal usage of these drum blocks is to hold files of images produced by or committed to a peripheral device. Each drum block will contain the drum address of the block in the file.

To request a drum block, execute

```
LMJ          11, ZRDB$
```

which will return with a drum address in A0. If the call is from the user's environment and no drum block is currently available, the subroutine will return with A0 negative. No return is made to a symbiont until a drum block is indeed available.

A drum block is released by placing its address in A0 and executing

```
LMJ          11, ZLDB$
```

The drum block now becomes available for other use.

9.2.3 THE DRUM FILE DIRECTORY

Files stored in linked drum blocks as described in the previous section may be recorded in a directory held in the resident. Two subroutines are provided for the maintenance of the directory: ZFILE$, which inserts a new file, and ZNEXT$, which removes a file. Entries in the file directory are of the form

where

"T" is the type code indicating the type of file (types thus far assigned are: Output Printer File, 0; Input Card File, 1; and Output Card File, 2);

"P" is the priority level of the file (larger priority numbers will be treated first); and

"Drum Address" :is the location of the first block of the file.

The linkage

```
LMJ          11, ZFILE$
```

will insert the contents of A0 into the file directory unless it is full. In this case, return is made with A0 negative. If the insertion is successful, return

is made with A0 positive.

The linkage

```
LMJ          11, ZNEXT$
```

will search the file directory for the highest priority file of the type indicated by A0. If no move files of the indicated type remain in the directory, return is made with A0 negative. If a file is found, its starting drum address is left in A0.

9.2.4 INPUT/OUTPUT OPERATIONS

The reader will remember that the request channel subroutine MRQC$ returns following a linkage to it, whereas the release channel subroutine MRLC$ does not. This behavior is unique to the user's environment When called from a symbiont environment, MRQC$ does not return to the caller but MRLC$ does. Linkage to MRLC$ is always executed from interrupt coding. When it returns to a symbiont, however, the system is no longer in interrupt coding, but is back in the normal mode.

The result of these alterations in behavior of the channel request and release is that a symbiont is not active from the time that it initiates an input/output operation until that operation is complete.

The tape and drum input/output packages make special provision to be called from either the user's or a symbiont environment. When called by a symbiont, a tape or drum package linkage will not return. In order to regain control an end-action routine must be specified. The end-action routine should not end by a return to 0, 11 as is the case for the user's environment, but by executing

```
LMJ          11, MRLC$
```

The symbiont will regain control in normal coding beginning with the instruction immediately following this call to MRLC$.

Calls to the drum package may use the packet mode only.

9.2.5 SUSPENSION AND TERMINATION

A symbiont **may temporarily suspend operation by means of the linkage**

LMJ 11, ZSPND$

which will produce the message

-xxx SUSPENDED

Return is made from the subroutine whenever the operator provides an unsolicited key-in addressing the symbiont **(see Section** 9.1**).**

In a similar manner, a symbiont **may terminate operation entirely by the linkage**

LMJ 11, ZTERM$

which will produce the message

-xxx TERMINATED

and release the core buffer area containing the symbiont**. No return will be made.**

EXEC II

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

U-3671