THE UNIVERSITY OF MICHIGAN

Memorandum

PDP-8/338 EXECUTIVE SYSTEM

Daniel R. Frantz

CONCOMP: Research in Conversational Use of Computers
F. H. Westervelt, Director
ORA Project 07449

Supported by:
DEPARTMENT OF DEFENSE
ADVANCED RESEARCH PROJECTS AGENCY
WASHINGTON, D. C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

administered through:

OFFICE OF RESEARCH ADMINISTRATION, ANN ARBOR

June 1967

TABLE OF CONTENTS

DF:mb

CREDITS

In the fall of 1966, a series of meetings of the personnel associated with the Terminal Room Operations Group produced ideas leading to the general organization of the system described in this manual. Among the contributors were J. Allan, T. Antrim, D. Frantz, B. Herzog, J. Jackson, S. Lundstrom, W. Seider, and R. Taylor. Subsequently, J. Jackson designed and programmed the interrupt monitor; R. Taylor modified and supplemented the ALICS assembler until it turned into MALICS; and the author programmed the loader/librarian and supervised the assembly of the various parts of the system.

PDP-8/338 EXECUTIVE SYSTEM

INTRODUCTION

This manual describes the first step taken to provide a coherent executive system for the Digital Equipment Corporation's Type 338 Programmed Buffered Display (a PDP-8 computer and a display controller that drives a cathode-ray tube). The hardware may be viewed as two different computers operating on the same storage area, but each with a different scheme for accessing data. In addition, the organization of the storage area (3 "banks" of 4096 words each, and 32 "pages" of 128 words each in each bank) makes it impossible to treat all data and programs in a uniform fashion. The system is an attempt to provide a flexible method of handling these two problems.

The executive resides in core bank zero during the loading and execution of a program. Part of the executive is a relocatable linking loader that will load a binary tape produced by the MALICS assembler. A user's programs are put into core banks one and two, providing him with approximately 8,000 words of working core storage. During execution, a program may call on the Executive Subroutine Caller to aid in calling subroutines across the core bank boundaries. Also during execution a priority interrupt dispatcher (the "monitor") provides a flexible tool for input-output processing and an effective method for computer display interaction. After execution of a portion of his program the user may swap programs into and out of core while maintaining the integrity of his data files. These functions and additional features are explained in detail below.

Thorough knowledge of the PDP-8 internal organization and some knowledge of the display controller is assumed.

This information may be obtained from the PDP-8 User's Handbook (DEC-85) and the Programmed Buffered Display 338 Programming Manual (DEC-08-G61B-0). The ALICS-II Programmer's Manual (Information Control Systems, Inc., Ann Arbor) is required reading for the section on the MALICS assembler.

This manual and the system it describes are subject to change at any time.

1. Organization of Storage: "Program" and "Data"

The 8,000 words of relocatable core are divided into storage assigned for "Program" routines (also called subroutines or programs) and "Data" routines (also called data files or display files). Subroutines generally consist of PDP-8 instructions (and local data) needed to solve a problem. There is no main program, but there is a principal subroutine that will be called first. Data routines generally act as common storage for many subroutines and quite often include sets of instructions for display generation. Each routine has associated with it a name (or names) that may be referenced by other routines. For example, MATMUL may be the name of a subroutine that performs the multiplication of two matrices, and CUBE may be the name of the data file that contains the display instructions for displaying the projection of a cube on the face of the cathode-ray tube. Other routines may refer to these names through the appropriate type of control linkage, either subroutine or data linkage, respectively. The main distinction between program and data, however, arises in the handling of very large problems. A program that requires 13,000 (decimal) instructions obviously can't be put into core all at once. The programmer may be able to divide the instructions into two blocks, one of 7000 and the other of 6000 instructions. He can then load

and run the first block and when it is finished he can "LINK," that is, wipe out the first block, load, and run the second. Because the first block calculates data needed by the second, the data should not be wiped out when the second block is reloaded. The division into program and data routines facilitates this procedure. All data routines and data name definitions are saved intact during LINKing, and all program routines and program name definitions are wiped out. The space the programs previously occupied is then made available to other routines. When the new "link" is loaded (i.e., a new set of subroutines and, perhaps, additional data file definitions), connections can be made to the data files still in core via the data-name definitions.

Data and program routines share the available relocatable storage (i.e., about 8,000 words). In the current system, program routines are allocated storage starting at either location 10200 or 20200 (octal) and continuing into the higher numbers; data routines are allocated storage starting at either location 17777 or 27777 (octal) and continuing downward. Relocation is done by pages, i.e., every program is loaded starting at a page boundary (a multiple of 200 (octal)). Thus, if a program 120 locations long is loaded at 10200, the next program starts loading at 10400.

Routines cannot be split across core boundaries. Routines may refer absolutely to any of page zero in banks one and two except locations three through seven (3-7), which are reserved for the Executive Subroutine Caller. (It should be noted, however, that since routines may be loaded into different core banks, directly addressed page zero instructions of two separate routines may be in different core

banks, so that the same instruction might refer to two different locations. Thus, page zero addressing should be used only for communications within a single program.

The assignment of program or data type to a routine refers only to the way routines are manipulated in the linking process. A defining facility in the MALICS assembler assigns the type of storage a routine will receive when loaded. This facility makes a distinction between program and data, but the distinction is not meant to be a restriction of the use of a linkage type to its respective type of routine, although such a division reflects the most common use of the linkages. Thus subroutine control linkage (using the ENTRY and SUBR pseudo-ops) is provided to enable a transfer of the PDP-8 processor to a location defined in another routine and is usually associated with "programs." The data or display control linkage (using the DATNAM and FBA pseudo-ops) is provided for transfer of the 338 display processor, which usually takes its instructions from files that are treated as common storage by many subroutines (i.e., a data file). In certain, somewhat complex uses, however, both types of linkage may be present in both types of routines so that the routines may take advantage of both the swapping algorithm and the different types of linkages.

Warning: Care should be taken in mixing control linkage types within a routine since unexpected results may occur when LINKing. Both types of linkage definitions share a table in the librarian known as the linkage or librarian's table. The definitions are stored according to the type of linkage (i.e., all "subroutines" together and all "datas' together) and not according to the type

of routine in which they occur (i.e., a data linkage definition occurring in a program-type routine is stored with other data definitions). When the program is LINKed, all the data definitions are kept, and all the program definitions as well as the programs themselves are destroyed. Observe that this has two effects: 1) Data definitions that were defined from program subroutines are no longer valid since the routine is gone, i.e., the definition exists in the table, but the routine no longer is in core; 2) some subroutine entry points are safely situated in core, but their definitions are no longer available in the librarian's table (because all program definitions were destroyed), so that they can no longer be called through the Executive Subroutine Caller and are thus effectively unavailable.

Currently the table can accommodate a total of 64 definitions (program and data names).

2. Librarian

The librarian is that part of the executive responsible for keeping track of what is in core and where. It also provides teletype communication to and from the console, thus enabling the user to initiate loading and do limited debugging.

The librarian is divided into two parts: MALS (Multicore Assembler-Loader System), the portion that controls loading, defines routine names, and creates linkages; and the communicator, which accepts and interprets the commands to MALS and the debugging routines.

2.1 The Communicator

After the system has been loaded and started, the communicator indicates its readiness to accept input by typing WHAT

NOW? (Note: messages typed by the computer are capitalized and underlined.) Any of the commands described below may then be given.

2.1.1 Format of Commands

Each command is typed entirely on one line (of at most 72 characters, including spaces) and is terminated by a carriage return. When more than 72 characters are typed, the message RETYPE is given and the line ignored. Where more than one word is needed to specify an action (for example, when parameters are required), the pieces of information must be separated by at least one space; additional spaces are ignored. For example, DUMP 55 is a legal command, while DUMP55 is not. Only the first three characters up to the next blank (or carriage return) are ignored. Thus DUMP 55, DUM 55, and DUMBELL 55 are equivalent commands.

The characters " (double quote) and ? (question mark) are used for input editing. If the user makes a typing mistake, he may erase the last typed character by typing a double quote ("). Two double quotes (" ") erase the last two characters, etc. Thus, both DB"UMP and DBM""UMP are equivalent to DUMP. To delete a whole line and restart it, the question mark (?) may be used. This procedure will initiate a carriage return and line feed so that a whole new line becomes available. As mentioned above, input of a carriage return terminates a line and requests the appropriate action to be taken.

If a command is incorrectly formed (i.e., if it occurs out of context, is misspelled, or has incorrect parameters), the communicator types WHAT?, and the user is given another chance to enter input. During the course of a procedure the user may be asked a question. Should he not want to answer it, he may type in "STOP," and the communicator

will revert to the original command-accepting state.

2.1.2 Debugging Aids

The commands DUMP, REPLACE, and GOTO may aid in debugging programs. Additional commands may be added later.

2.1.2.1 Command: DUMP N1 N2

This command provides a typeout of the core locations specified by the parameters. N1 and N2 (optional) are octal parameters that must be in the range 0-27777. If N2 is present it must be greater than or equal to N1 and must also be in the same core bank as N1. Leading zeros in the addresses need not be specified. If only N1 is given, only the contents of location N1 will be given, with an appropriate label. If both N1 and N2 are given, all locations between and including N1 and N2 are outputted, eight words per line with appropriate labels.

Examples:

```
DUMP  45
00045  XXXX
DUMP 20105  20117
20105 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
20115 XXXX XXXX XXXX
```

2.1.2.2 Command: REPLACE N1 N2 N3..... NI...

This command permits the teletype-user to alter the contents of core. All the NIs must be octal numbers in the range 0-27777. Any five-digit NI (filled out with leading zeros if necessary) acts as an origin, and all following four-digit (or smaller) numbers are treated as data, taken from left to right, to be loaded into origin, original+1,... All four-digit (or smaller) numbers up to the first five-digit address are ignored.

Examples:

REPLACE 14321 15

effect: 14321 ← 0015

REPLACE 23201 21 4 13220 5204

effect: 23201 ← 0021

23202 ← 0004

13220 ← 5204

REPLACE 15 4251 00077 0407 21111 12222

effect: 00077 ← 0407

Any illegal character or number in the input line stops the processing immediately, although all valid replacements to that point will have been made.

2.1.2.3 Command: GOTO ADDRESS DF

This command enables the user to transfer program control while at the teletype. ADDRESS is an octal number that must be in the range 0-27777, and DF must be 0,1, or 2. The Data Field Register is set from DF, and control is transferred immediately to location ADDRESS.

Examples: GOTO 10200 1

GOTO 600 2

2.2 MALS (Multicore Assembler-Loader System)

MALS is the part of the librarian that directs the loading of the user's routines by allocating storage, defining external names (i.e., subroutine and data names), and providing the proper types of linkage. MALS is activated through the communicator by any of the following commands: START, LINK, RESUME, or LOAD.

2.2.1 Initialization Commands

2.2.1.1 Command: START

The command START reinitializes the entire executive system. It clears the library tables, defines all relocatable core to be available, and sets all Interrupt Service Routines (ISRs) and system routines to their standard definitions. Processing then continues as in RESUME.

2.2.1.2 Command: LINK

The command LINK wipes out all program definitions, declares as available all core previously occupied by program routines, and redefines ISRs and system routines. All data definitions and routines are maintained intact. Processing then continues as in RESUME.

2.2.2 Program Continuation Commands

2.2.2.1 Command: LOAD

The command LOAD directs the loader to load the tape in the reader without further ado. When loading is finished, control reverts to command state.

2.2.2.2 Command: RESUME

The command RESUME maintains the current state of all tables and directs preparation for execution. It first asks the question:

SYSTEM--

requesting the input of the six-character at most name of the primary subroutine, i.e., the first subroutine to be

called when the user's system of subroutines and data are loaded. This name must obviously be of the same form as variables defined in the assembler. It is terminated by the first blank on the input line. Trailing blanks need not be supplied in the case of a name of fewer than six characters. This name is placed in the librarian's subroutine table.

The librarian's tables are checked to see if all external names (subroutine and data) are defined. (Obviously, the primary subroutine is not defined the first time through.) If an external name is not defined, the following question is typed out:

DO YOU HAVE A TAPE FOR XXXXXX?

where XXXXXX is the name of the undefined routine. A "YES" or "NO" answer is expected, although the user may type "STOP," thereby returning to command mode, if he decides not to answer.

If the answer is "NO" and the definition requested is of program type, the librarian will make the linkage to a special routine that intercepts calls to the nonexistent routine and returns control to the executive system. (This feature is to allow checkout of independent portions of programs without having to load all subroutines needed.) If the definition requested is of data type, there is no safe way of linking or intercepting the faulty call, and the user takes responsibility for continuing.

If the answer is "YES," the loader takes over. It prints:

START READER

directing the user to load a binary tape in the reader and

to start the reader when ready. The program on the tape is then put into core. During the loading procedure, new names may be put into the librarian's table from the program tape by its definition of external names and requests for linkages. After the program has been loaded (See Loader, Section 2.2.4 for details), the librarian once again checks its tables, and if more names are undefined, it repeats the above process. ISRs and system service routines need not be defined by the user if he wishes to use the standard routines that are available (See Section 2.3).

If two user subroutine entry points are defined as having the same name, all connections will be made to the last defined such name, i.e., the second takes precedence. However, if two data routines are defined with the same name the effect will be different because of the different type of connection that is made. In particular, before a data name is defined the second time, all linkages are made to the first definition of the name, and after the second definition of the name all linkages are made to the second definition.

When all names have been defined and all linkages made, the librarian prints a loading map with external names and definitions, and asks: READY TO GO?
A "NO" answer returns the user to command state; a "YES" answer directs the executive to commence execution.

2.2.3 Execution

The executive starts the execution of a user's program by calling his primary subroutine. The primary subroutine is so designated when the user answers the question

SYSTEM-- (as noted in Section 2.2.2.2). If more than one name has been so defined, in the last-entered name becomes the primary subroutine. Execution begins with the display controller off, the monitor's interrupt table empty, interrupts on, the Data Field and Instruction Field registers both set to the field of the primary subroutine, and the accumulator zero.

If the user wishes to initialize the display, instead of issuing the direct command (SIC=IOT 145) to the display controller, he should call the system service routine SICDSP that allows the monitor to determine which interrupts from the display are enabled. This call must be made each time the interrupt conditions of the display are to be altered and, in particular, before it is started the first time. The calling sequence for SICDSP is:

```
                CLA             /needed if AC≠0
                TAD INCOND      /get initial conditions in AC
                JMS 3           /call the ESC
                SUBR SICDSP     /for routine SICDSP
RETURN          ----------
```

where INCOND is a location containing the bit configuration specifying the initialization (See Programmed Buffered Display 338 Programming Manual). Control returns from the subroutine to the next instruction after the call (i.e., to location RETURN) with the accumulator zero (See Appendix 1 for the remaining calling conventions).

After execution is completed, control may be returned to the executive by the normal subroutine return sequence (See Appendix 1) from the primary subroutine, in which case RETURN TO EXEC is typed out; or control may be returned directly from any part of the user's program by a jump to

location 600 (octal) in core bank zero, with both the instruction Field and Data Field registers set to zero. In either case, the executive enters command mode immediately.

2.2.4 The Loader

The loader is the section of the librarian that reads the binary program tapes produced by the MALICS assembler and processes the program to relocate it in core and to link it to other subroutines and to the librarian itself. The format of the binary tape and the relocation codes used are given in Appendix 3.

Immediately after the loader starts executing it types out:

LOAD TAPE, START READER.

The user should then position the binary tape so that leader code (200 octal) is in the reader and start the reader when ready. The loader manipulates the data according to the various "relocation codes" (RC); each word of the program has such a code associated with it. Some of the RCs are direct instructions on how to load the data and some are requests for information that is known only at load time (e.g., relocation constant, linkages, etc.).

At the end of the tape is a "checksum," the sum of all the characters on the tape except the checksum. This number was computed and punched by MALICS when it produced the tape. As it reads the tape, the loader calculates the sum of all the characters and at the end compares its computed checksum with the one punched on the tape. If they are identical the statement

LOAD OK

is typed out, indicating that the information on the tape was probably transferred to the machine correctly. The machine then stops. The user should turn off the tape reader and press the CONTINUE key; control will then return to the librarian.

Upon detection of an error during the loading process, a suitable message is typed out and the computer stops. The user should turn off the reader and press the CONTINUE key. Control is then returned to command mode to await the user's decision on the next step. The error messages and their consequences are listed below.

CHECKSUM

The checksum computed by the loader does not agree with the checksum on the tape. The information loaded into core is probably incorrect.

ILLEGAL RC

The relocation code of the word just read is not one of the legal codes (See Appendix 3).

Either of these errors indicates a misreading of information on the tape.

The misreading may be of the data, i.e., an incorrect word was loaded into core, or the misreading may have been of instructions on how to load, i.e., the data were put into the wrong place. Thus, a word may have been loaded into unused core, or it may have destroyed useful information by being loaded over parts of the user's program that had previously been loaded correctly, or, worse yet, it may have been loaded over the executive itself. In either of the latter two cases the respective programs must be

reloaded since they are no longer trustworthy.

Unfortunately, there is no way of assessing the damage (although the first, least harmful case is most probable). The most reasonable error procedure is:

1. Try to reload the program, i.e., type LOAD and continue from the beginning of the loading procedure.

2. If the tape loads properly the second time, the probability is high that the contents of core are correct and that nothing was seriously damaged.

3. If the error persists, reload the executive system and then try loading the tape again. If the error persists, the program is probably a worn or defective tape.

The other loading error messages are:

MEMORY OVERFLOW

Memory cannot accommodate the program about to be loaded.

TABLE OVERFLOW

The librarian's table cannot accommodate the external definitions from this tape.

To correct these errors the program must be re-organized.

2.3 Service Routine Connections

2.3.1 System Service Routines

The librarian's table contains a list of predefined subroutine names which are available for reference by user programs. These subroutines are resident in core and provide such services as teletype I/O buffering and other

fairly standard operations. To call these subroutines, the user need only request linkage to them by means of the SUBR pseudo-op in his program. If he wishes to define a subroutine with the same name as one of the standard routines of this type, his definition overrides the standard one. As additional system service routines become available, descriptions of them will be published.

2.3.2 Interrupt Service Routines (ISR)

An ISR is a routine that processes an interrupt. It interacts with other ISRs and the monitor in a special way. The following subroutine names are "reserved" to refer to ISRs:

LPHIT	DATFON
EDGFLG	TTYIN
INTSTP	TTYOUT
MANINT	PBHIT

These names are associated with the interrupts: light-pen hit, edge flag, internal stop, manual interrupt, 201 Data Phone, keyboard flag, teleprinter flag, and push-button hit, respectively.

The standard routines may be used for handling interrupts without making any special arrangements. STARTing, MALS, or LINKing reset the handling of interrupts to the standard routines. A user who wants to write his own interrupt handlers must inform the monitor of these routines by one of two methods: the first calls on the monitor directly, during execution (See Section 4.3.2); the second uses the appropriate "reserved" name in an ENTRY statement. Upon recognizing the "reserved" name, the librarian will automatically make the necessary arrangements with the monitor before execution starts.

A note of warning: The list of names is "reserved" in the following sense: whenever one of the names is defined to the librarian as a subroutine name, it is assumed to be a non-standard ISR and the monitor is informed of its existence. If the routine is not an ISR, and an interrupt of the indicated type occurs, the monitor will transfer control to that routine—which probably won't do what the monitor expects. In addition, a standard ISR is meant to be called directly only by the monitor, and any other part of a program calling on it will only create confusion. A program ignoring either of these two facts will probably "blow up." On the other hand, if the indicated interrupt never occurs (e.g., if the dataphone is turned off), then no problems result from use of the "reserved" names. It is recommended, however, that these names be used only in their "reserved" (i.e., ISR) sense to avoid confusion and possible disaster.

2.4 Executive Subroutine Caller (ESC)

This feature of the system facilitates subroutine calls across core boundaries. The standard subroutine calling sequence is:

```
JMS     3
SUBR    QQSV
```

for the subroutine QQSV. Locations 3,4,5,6, and 7 of each core bank contain an ESC caller. From it, control is transferred to the ESC in bank zero, which uses the identification created by the SUBR pseudo-op to transfer control to subroutine QQSV, wherever it may be. When subroutine QQSV is given control, the following two conditions prevail:

1. The Data Field register is set to the field of the calling subroutine.

2. Location QQSV contains the address (twelve bits worth) of the location following the SUBR QQSV statement, i.e., QQSV thinks it was called directly from the location of the SUBR QQSV statement. This enables subroutines to be written as if the ESC hadn't intervened and SUBR pseudo-op were not necessary. (See Appendix 1 for further calling conventions.)

3. The MALICS Assembler

MALICS (Michigan Modified ALICS) is a modification of the ALICS-II assembler from Information Control Systems, Inc., of Ann Arbor. ALICS was chosen because it was capable of producing the relocatable code which is deemed necessary for a flexible system; thus only minor additions were needed to adapt it to our needs. ALICS also allows the programmer to use literals. It has an automatic paging feature which allows one to ignore the peculiar addressing structure of the PDP-8 within a core bank—i.e., it makes the PDP-8 look like a machine with four thousand words of directly addressable storage.

As of this writing, the version of ALICS on the system does not have the "convenience" pseudo-ops AORG, BSS, and SCI. These op codes will be added when Information Control Systems releases the next modification level of the assembler. Examples using these op codes are included for completeness. The ALICS-II manual can be referenced with the following emendations, changes, and notices.

3.1 ALICS Changes

It is questionable whether the automatic paging feature

can be used to much advantage except in large, practically self-contained programs and then only with a great deal of care. The point in question is that in an automatic paging system the assembler generates indirect instructions for off-page references. In a multicore system the Data Field register is consulted on TAD, AND, DCA, and ISZ indirect instructions so that an assembler-generated indirect reference may refer to a location in another (wrong) core bank if the IF and DF registers are not the same. This problem might be circumvented by careful programming, but only use will determine such a possibility. In a non-automatic paging mode, literals must also be used with care. The programmer must allow room on every page for the literal definitions, and must indicate to the assembler that literals should be assigned for a page by using the pseudo-op PAGE at the end of every page of code. He must also write his own off-page indirect linkages.

The pseudo-ops ENT, CALL and EXT of the ALICS assembler are no longer defined in MALICS. RORG has been redefined to relocatably reorigin the program at the location given in the address field. For example,

```
RORG    432
CLA
```

has the effect of putting the CLA into relocatable location 432 (i.e., location 432 with respect to the beginning of the program, which is based at 200 octal).

3.2 Additional Pseudo-ops

Several new pseudo-ops have been added to the assembler to facilitiate a program's operation in a multicore environ-

ment and to enable it to define display files readily. These additional pseudo-ops are most easily divided into three groups: those referring specifically to core banks, those referring to names of subroutines and data files, and those referring to type of routine.

3.2.1 Data Field Reference Pseudo-ops

3.2.1.1 FLDHER

In conception, this pseudo-op is equivalent to a (CDF*) if such an instruction were allowed. That is, it is an instruction to change the Data Field register to the core bank in which the program currently resides. For instance, if the program were loaded into core bank one, FLDHER would be 6211 (CDF ONE), or if it were loaded into bank two, FLDHER would be 6221 (CDF TWO). since the program doesn't know where it is going, the librarian will supply the proper code at load time.

3.2.1.2 DJHER Expression

This op-code is used especially in conjunction with display jumps. The occurrence of DJHER defines the low-order three bits of the word being assembled as the data field which the program currently occupies. For example, if DJHER appeared alone as an instruction and if the program ended up in bank two, the word would be loaded into storage as 0002 (octal). A display jump takes two words, with the three low-order bits of the first word specifying the core bank to jump to. A display jump or push jump to a location within the same routine may be coded using DJHER, since the jump instruction and target location will certainly be in the same core bank. For example,

```
ABC: - - - - - - - -  /define name ABC internally
          .
          .
          .
     DJHER JUMP       /Display JUMP to location in same
                      /core bank, JUMP = 2000.
     ABC              /low-order 12 bits of jump
                      /instruction
```

DJHER may also be used for helping to set the initial conditions flags in the display controller. One of the start-up instructions for the display sets the break field (core bank) of the first display instruction to be executed. If the current program contains the first display instruction, the following instructions will properly initialize the break field register:

```
     TAD BFINST       /get DF in low-order bits
     RTR+CLL          /rotate to high-order bits for
     RTR              /display IOT format
     CML+RAR          /set enable bit (bit zero)
     LBF              /=6155--load break field
     .
     .
BFINST: DJHER         /get current core bank
```

3.2.1.3 FIELD Expression

This op-code is employed when the user wants to load in absolute mode (i.e., no relocation). The expression in the operand field denotes the core bank into which the remainder of the program is to be loaded. See the section on absolute programs for an example of use.

3.2.2 Program Name Pseudo-ops

3.2.2.1 ENTRY NAME

ENTRY defines "NAME" as a subroutine entry point to be

made available to other routines. "NAME" must have been previously defined. For example,

```
JUNK:          Ø
               ----/first executable instruction
                   /of subroutine JUNK
                .
                .
                .
                .
          ENTRY JUNK/define JUNK to outside world.
```

ENTRY produces no code in the program but just sends information to the librarian. More than one entry to the program may be defined. See the sections on the librarian and monitor for the special meaning of the following symbols when they are defined by ENTRY: LPHIT, EDGFLG, INTSTP, MANINT, DATFON, TTYIN, TTYOUT, PBHIT.

3.2.2.2 SUBR NAME

SUBR is a request for linkage to another subroutine. "NAME" must be defined by appearing in an ENTRY statement in another subroutine. At load time the librarian will insert an identification code into the location of the SUBR instruction. This identification is the location in the librarian's table (in bank zero) of a two-word block defining the location of the variable. The normal use of SUBR is for a subroutine call through the Executive Subroutine Caller. For example, to call subroutine QQSV, the following two consecutive instructions are sufficient:

```
JMS            3      /call to exec sub. call
SUBR           QQSV   /to call routine QQSV
```

3.2.3 Data Name Pseudo-ops

3.2.3.1 DATNAM NAME

DATNAM defines "NAME" as a data file definition to be made available to other routines. "NAME" must have been previously defined. DATNAM produces no code but just sends information to the librarian. More than one data name may be defined in each file.

3.2.3.2 FBA NAME

FBA (Fifteen-Bit Address) is a request for display linkage to another data file. "NAME" must be defined by appearing in a DATNAM statement in another data routine. The occurrence of FBA causes two locations to be altered by the librarian at load time: 1) the location of the FBA op-code, into which is inserted the low-order twelve bits of the address of "NAME," and 2) the location immediately preceding the FBA, whose low-order three bits are replaced by the data field into which "NAME" has been loaded. This pseudo-op is intended for use in conjunction with display JUMPs and PJUMPs, which are two-word instructions, requiring fifteen bits of address to specify a location. For example,

```
JUMP            /JUMP     = 2000 (octal)
FBA   QQSV
```

will create (at load time) the proper two words to transfer display control to the data file defined by QQSV. (See the DEC 338 Programming Manual for more information on display instructions.) FBA may also be used to help set the initial conditions of the display break register (as explained in Section 3.2.1.2).

3.2.4 Routine Pseudo-ops

3.2.4.1 PEND

PEND indicates the end of an assembly and that the routine is to be loaded and assigned storage as a "program" routine.

3.2.4.2

DEND indicates the end of an assembly and that the routine is to be loaded and assigned storage as a "data" routine.

3.2.4.3 END

END indicates the end of an assembly. The routine is to be loaded without assigning special storage for it and without altering the relocation constant from that of the last routine loaded. END is used in programs which are overrides or which are loaded in absolute mode.

3.3 OVERRIDES

Rather than completely reassembling a long program when only a few corrections are needed, a user may choose to write a symbolic override correcting only the faulty instructions in the program. This override or correction should be assembled in relocatable mode and be terminated by the END pseudo-op rather than PEND or DEND. The resulting binary (object) tape should then be loaded immediately after the program it is correcting. For example, if only location 140 (relative to the first location of the program, which is a relocatable 200 octal) is in error and the correction is to be a TAD ABC (where ABC is a relocatable

55), the following would create a binary tape with the desired information:

```
RORG        55    /define ABC as relocatable 55
ABC: --
RORG       140    /reorigin with respect to the
                  /beginning of a relocatable
                  /program
TAD        ABC    /this instruction goes into re-
                  /locatable location 140
END               /end of override.
```

3.4 Absolute Programs

Currently, there is no way safely to integrate absolute and relocatable programs. The user must take all responsibility for storage allocation if he choses to use any absolute coding. However, all of the librarian's other features are available to such programs, i.e., subroutine and data file definition and linking, and subroutine calling through the Executive Subroutine Caller.

To load in absolute mode two pseudo-ops must appear, one to specify the core bank of loading and the other to specify the address within the bank. For example, the instructions to direct the loader to start loading a program at location 24100 (octal) are

```
FIELD       2
AORG        4100
```

4. MONITOR

The monitor is an interrupt dispatcher that identifies the cause for an interrupt and transfers control to an Interrupt Service Routine (ISR) which will do the required

processing.

4.1 Interrupt Scheduling

In the case of overlapping interrupts, the monitor will stack the requests for service (one level deep for each type of request) and then schedule the ISR's according to priority assignable by the writer of the ISR's.

A standard ISR is one which resides in core bank zero and whose entry point address appears in octal location 170+n where n is the interrupt number (defined below). If several interrupts are assigned the same priority, they are scheduled for service on a first-come, first-served basis. The priority of an interrupt service may be altered at any time by any program. However, only subsequent requests for that service (i.e., occurrences of the associated interrupt) will be scheduled with the new priority.

4.2 Interrupt Service Routine Format

An ISR is called by the monitor upon detection of the associated interrupt. It is entered with the instruction and data field registers set to the core bank the routine occupies.

The ISR must first save all registers external to itself which it will use (e.g., the accumulator, the multiplier-quotient register, the index registers, and the instruction and data field registers). It then performs all necessary data transmissions to or from the PDP-8, resets the flag that caused the interrupt, and reenables interrupts as soon as possible. It may also transmit information to the user's program about the state of the input/output procedure. When it is finished, it must

restore all registers and make a special exit to the monitor to location 200 (octal) in core bank zero.

An ISR with the name SERV must be written in the following format:

```
SERV:    PRIOR          /entry point, contents=priority #
         -----          /first executable instruction
           .
           .
           .
         CIF            /required if not in bank zero
         JMP* K200      /k200 contains 200 octal.  This
                        /is the standard return to the
                        /monitor.
```

The monitor can be initialized to schedule all interrupts for service by the system-supplied standard ISR's. A user may replace any of the standard routines with his own. The librarian provides an automatic mechanism for this replacement (See Section 2.3.2), although the user may choose to initiate this action himself during execution (see below).

4.3 Monitor Calls

During execution, the user may call on the monitor through the system service routines to effect the following functions: monitor initialization, notification of non-standard ISR's, and display initialization. For the format of the call for display initialization see Section 2.2.3.

4.3.1 Monitor Initialization

Subroutine INTMON initializes the monitor so that all interrupts are serviced by the standard ISR's. All unprocessed interrupts are deleted from the monitor task queue and the display is stopped. The calling sequence is through the Executive Subroutine Caller:

```
JMS     3
SUBR INTMON
```

4.3.2 Non-Standard ISR's

If a user wishes to use his own interrupt handlers, he may inform the monitor of their existence and location during execution by calling system service routine OWNISR through the ESC. The format of the call is:

```
JMS     3
SUBR OWNISR
SERV
DF
NUMBER
```

Where bits 6-8 of DF contain the data field of the entry to the ISR (the remaining bits are ignored); SERV is the twelve-bit address of the entry point; NUMBER identifies the interrupts to be serviced according to the following list:

0: Light pen
1: Edge flag
2: Internal stop
3: Manual interrupt
4: 201 dataphone
5: Keyboard
6: Teleprinter
7: Pushbutton hit

APPENDIX 1

Subroutine Calling Conventions

In general, a subroutine will not be in the same core bank as the program that called it. The following conventions have been adopted to help standardize communications between subroutines that may not be in the same bank.

Conventions

Upon entry, the subroutine may assume that:

1. The Instruction Field Register (IF) is equal to the core bank which the subroutine occupies.

2. The Data Field Register (DF) is equal to the core bank occupied by the calling program.

Upon return, the subroutine should:

3. Set the DF to the core bank of the calling routine.

Calling Programs

The Executive Subroutine Caller (ESC) aids the calling program in meeting the first two requirements. For example, the following two instructions (given in assembly language) will produce the code for the ESC that will enable it to set the DF to the calling bank, set the IF to the bank of the subroutine, and transfer control to the subroutine as if it had been called without the ESC.

```
JMS     3          /call ESC
SUBR    SNAME      /in order to call "SNAME"
```

locations 3,4,5,6, and 7 of each core bank contain part of the ESC. Location 3 is the entry to the ESC, hence the "JMS 3" instruction, a call on the ESC. "SUBR SNAME" produces code at load time that uniquely identifies subroutine "SNAME" to the ESC. Note that the ESC makes the call on the subroutine look as if the ESC had not intervened and as if the "SUBR SNAME" instruction were not necessary; that is, the subroutine thinks it was called from the location at the "SUBR SNAME" instruction and not the "JMS 3" instruction.

This is done so that the parameter list of a subroutine call is immediately available to the subroutine without having to skip over the location of the "SUBR SNAME" instruction. For example, if the main program were in bank one and were calling subroutine SINE from location 1543, the following would be sufficient:

```
      11543          JMS          3          /ESC call
      11544          SUBR         SINE       /for subroutine SINE
      11545          PARX                    /with parameter X.
      11546 RETURN:  - - -
```

if subroutine SINE were in core bank two with its entry point at location 1200, upon entry it would notice that: the IF is 2, the DF is 1, and the contents of location 1200 (in bank two) are 1545, as if the subroutine has been called with a JMS from location 11544.

Subroutines

To function properly, a subroutine must first save the contents of the DF so that it can be restored before returning (if the DF is changed during execution of the subroutine), and then prepare the instructions to return to the other bank. Only after these two tasks are completed can the subroutine pick up the parameters and do its specified task. The format for the SINE subroutine might be:

```
SINE:     Ø                    /entry point
          RDF                  /get field of calling routine
          TAD    CDFINS        /from CDF to calling field
          DCA    RET           /put in return sequence
          RDF                  /get field of calling routine
          TAD    CIFINS        /from CIF to calling field
          DCA    RETP1         /put in return sequence
          TAD*   SINE          /get parameter to subroutine
          ISZ    SINE          /set return from subroutine
          FLDHER               /change data field to this core bank
                               /so that internal indirects work
```

```
                .
                .
                .
                .
                .                       /processing of SINE
RET:            0                       /CDF to calling field
RETP1:          0                       /CIF to calling field
                JMP*    SINE            /return
CDFINS:         CDF
CIFINS:         CIF
```

Appendix 2
Start-up Procedures

The executive system (librarian/loader, monitor, and system service routines) is on paper tape and should be loaded by the BINARY loader (Digital-8-2-U) into core bank zero. To start, enter 00 0600 into the switch register, press the LOAD ADDRESS key and then the START key. The librarian is then ready to accept commands.

Appendix 3
Relocation Codes

When the loader is loading a word it makes use of information about the word gathered by the assembler. The information is contained in four bits, called the relocation code, for each twelve bit input word.
Each word is specified by two eight bit characters from the input tape. The format of these characters is:

$$R_0R_1R_2R_3W_0W_1W_2W_3$$

$$W_4W_5W_6W_7W_8W_9W_{10}W_{11}$$

$R_0R_1R_2R_3$ is the four bit reloction code and $W_0 \ldots\ldots W_{11}$ is the 12 bit word. The relocation codes are given below.

Code 0 0000 No relocation. The input word is loaded as it stands.

Code 1 0001 Simple relocation. The input word is added to the relocation constant for the routine before loading.

Code 2 0010 Request for subroutine identification Code. The next six characters (in ASCII Code) on the input tape make up a subroutine name (left justified with trailing blanks) which is in the librarian's table. The location in the librarian's table which defines the symbol is loaded as the input word. This code is produced by the SUBR pseudo-op.

Code 3 0011 ENTRY definition. The input word contains the address of the entry point of a subroutine. The next six characters (in ASCII Code) on the input tape make up the subroutine name (left justified with trailing blanks) to be entered in the librarian's table.

Code 4 0100 Relocatable origin. The next location to be loaded into is given by the input word plus the relocation constant. This code is produced by the RORG pseudo-op.

Code 5	0101	FLDHER request. A CDF instruction to the bank in which the program is being loaded is loaded as the input word.
Code 6	0110	Not used.
Code 7	0111	Not used.
Code 8	1000	Leader-Trailer Code. The input word contains the tape check sum. This code signals the end of the input tape.
Code 9	1001	Absolute origin. The next location to be loaded into is specified by the input word. This code is produced by the AORG pseudo-op.
Code 10	1010	Request for fifteen bit address of data routine name. The next six characters (in ASCII Code) on the input tape make up the name of a data routine (left justified with trailing blanks) defined in the librarian's table. The high order three bits of definition at the symbol is added to the last location which was loaded. The low order twelve bits of the definition are loaded as the current input word. This code is produced by the FBA pseudo-op.
Code 11	1011	FIELD definition. The input word defines the core bank into which the remainder of the program will be loaded.
Code 12	1100	DJHER request. The binary representation of the core bank into which the program is being loaded is added to the low order bits of the input word and then the result is loaded.
Code 13	1101	DATNAM definition. The input word contains the address of the definition of a data name. The next six characters (in ASCII Code) on the input tape make up the data name (left justified with trailing banks) to be entered in the librarian's table.

Code	14	1111	Start of a "Data" routine. The input word contains the length of the sub-routine (extended to the next highest multiple of 200 (octal)).