WN-103
24AUG72

UNIVERSITY OF QUEENSLAND

Computer Centre

NEWSLETTER - SPECIAL EDITION

EXTENDED COMMAND SET

On Wednesday 9 August a new version of the monitor was implemented on the PDP-10 system. As well as correcting a number of errors and implementing some additional internal facilities (e.g. expanded hardware error reporting), this monitor provides an extended command set. These new commands are an extension of the existing command set - all previous commands continue to work as they previously did.

This special edition of the newsletter describes the new facilities and commands now available on the system.

1. PROGRAM FILES

In the PDP-10 system a program file can exist in any one (or several) of the five different states described below.

(a) Source Language File

This is a source language program (e.g. Fortran, Cobol, Macro, Algol etc) that exists as a text file, generally on disk or in punched cards.

(b) Relocatable File

A relocatable binary program file is produced on disk as the result of a compilation of a source language file by the appropriate processor. Relocatable files can be complete programs, individual routines or subroutines, or libraries of subroutines.

(c) Absolute Files

An absolute binary (or core image) program file is the result of loading one or more relocatable binary files with the linking loader and then transferring a copy of the incore program to disk.

The loader loads a program together with its required subprograms into core store, relocating each one, adjusting addresses as required, and resolving linkages between independently compiled subprograms. This produces, in memory, the required program ready to commence execution. The absolute file is simply a copy on disk of the program as it finally resides in core ready to run (hence 'core image' file).

(d) Incore (dormant) file

This is a copy of the absolute program file in core but not executing. It may be ready to commence execution, or may have ceasedexecution due to some external interruption.

(e) File in execution

This is a program executing in core.

Figure 1 shows the transitions that can be made between these different states, and indicates the commands required for each transition. It should be noted that it is not yet possible to go from relocatable binary files directly to absolute files. To generate an absolute file two steps are involved.

(i) From relocatable files use the LOAD command to produce an incore dormant program file

(ii) Produce a copy of the incore file onto disk with the SAVE command.

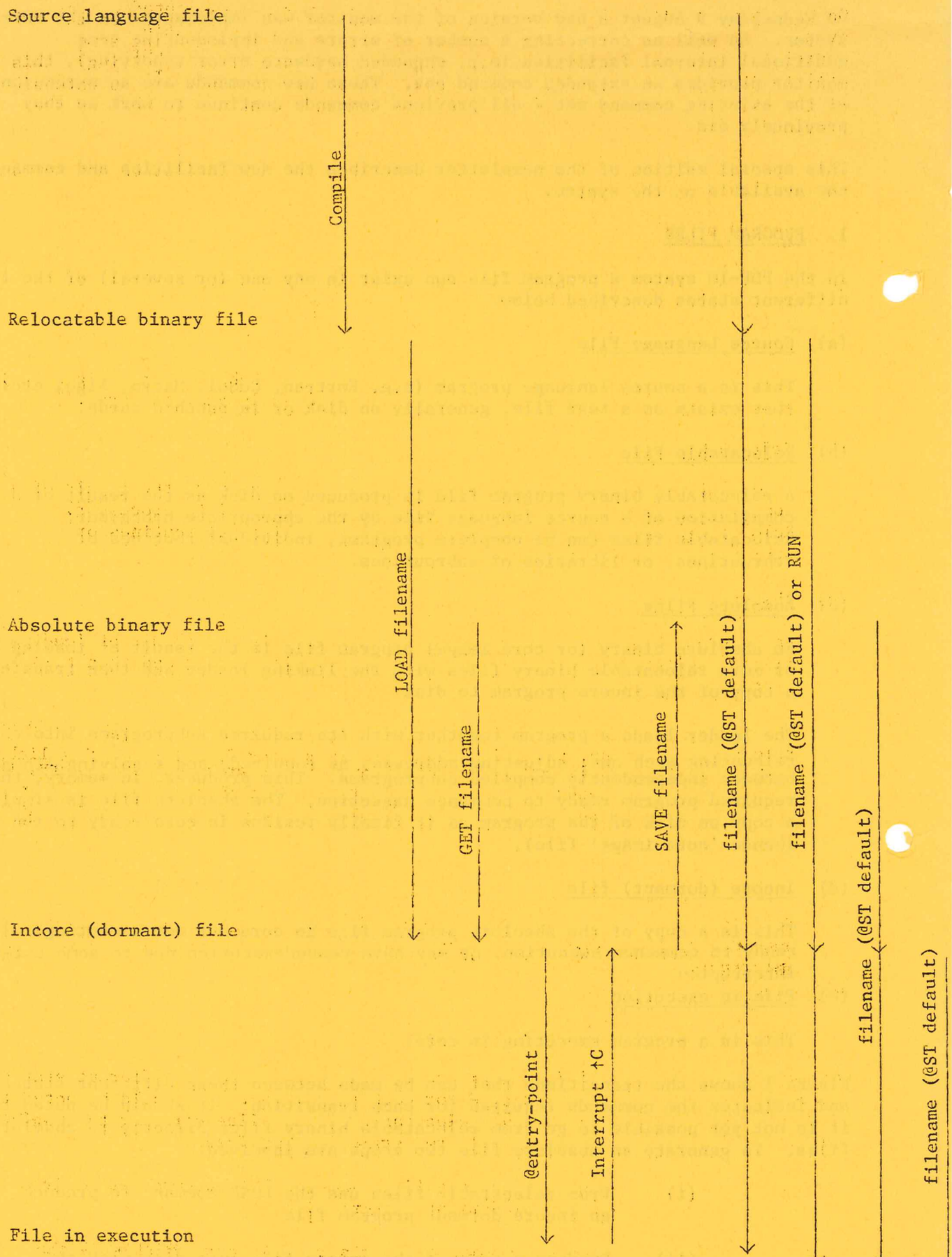
Source language file
Compile
Relocatable binary file
LOAD filename
Absolute binary file
GET filename
SAVE filename
filename (@ST default)
filename (@ST default) or RUN
Incore (dormant) file
@entry-point
Interrupt ↑C
filename (@ST default)
filename (@ST default)
File in execution

FIGURE 1.

2. RUN CONTROL COMMANDS

2.1 LOAD

```
     MAP
LOAD(NOMAP,DDT) filename-1(LIB),...,filename-n(LIB)
     SYMBOL
```

Load performs the same functions as the RUN command except that the files loaded are not actually executed. That is, it loads the relocatable binary file(s) named in the command (or from the loadlist) and places them in core ready for execution.

2.2 SAVE

```
SAVE(S) {core}
```

The SAVE command writes out an image of the user's current core area onto the disk with the filename given. This core image file can then be reloaded quickly into memory at any later time with a GET command.

There are in fact three types of user programs with corresponding core image files.

These are
(a) one segment non-reentrant programs
(b) two segment non-reentrant programs
(c) reentrant programs.

Types (b) and (c) are covered in the appendix on "Segments and Reentrant Programs".

The one segment non-reentrant program is by far the most generally used type of program. When SAVEd, it results in a single core image file on disk with the filename given and the processor program name of /SAV.

The core argument is the amount of core in which the program is to be run, specified in 1K blocks. If omitted the amount of core actually required by the program is assumed.

The S option is used for saving reentrant programs as explained in the appendix.

2.3 GET

```
GET filename {core}
```

GET retrieves an absolute binary file from the disk, placing it in core ready for execution.

filename — is the name of the file

core — is the amount of core to be assigned if different from the minimum core needed to load the program, or from the core argument of the SAVE command that saved the file.

2.4 @entry-point

```
     ST
     RES
     REE
    @PC
     DDT
     CST
     CPC
     octal-address
```

This set of commands will begin execution of the incore program at a number of different entry points. Many of the entry points used are locations discussed in the Job Data area (see appendix).

2.4.1 @ST

Begins execution of a previously loaded program at its normal starting address.

2.4.2 @RES

@RES starts the incore program executing at the normal start address +1. Many of the system programs use this facility when their arguments have been decoded.

2.4.3 @REE

The @REE command causes execution of the program to commence (or recommence) at an alternate entry point called the 'reentry address' which must have been specified by the user or his program.

2.4.4 @PC

@PC continues the program execution at the saved program counter address stored by a ↑C instruction.

2.4.5 @DDT

For programs loaded with DDT, this command commences execution of the program at the DDT start address. This allows the user to give DDT commands and follow the execution of the program (see the DDT manual MNT-12).

2.4.6 @CST

Same as @ST except that the user's console is left in monitor mode.

2.4.7 @CPC

Same as @PC except that the user's console is left in monitor mode.
Further monitor commands can be entered from the console after @CPC or @CST have been given. Note that these commands should not be used when the user program (which is continuing to run) is also requesting input from the console.

2.4.8 @octal-address

This command begins execution of a previously loaded program. The octal address specified is the address at which execution is to begin.

2.5 E

E {location}

E examines a core location in the user's area (high or low segment).

location — If the address is specified, the contents of the location are typed out in half-word octal mode. The address is required the first time the E or D command is used.

If the address is not specified, the contents of the location following the previously specified E address, or the location of the previous D address are typed out.

responses:

OUT OF BOUNDS

The specified address is not in the user's core area, or the user does not have read privileges to the file that initialized the high segment.

2.6 D

D lh rh {location}

D deposits information in the user's core area (high or low segment).

lh	The octal value to be deposited in the left half of the location.
rh	The octal value to be deposited in the right half of the location.
location	The address of the location into which the information is to be deposited. If the address is omitted, the data is deposited in the location following the last D address or in the location of the last E address.

responses:

OUT OF BOUNDS

The specified address is not in the user's core area, or the high segment is write protected and the user does not have write privileges to the file that initialised the high segment.

EXAMPLES:

```
.LOAD ABC/REL<cr>
LOADING

LOADER 1K CORE

EXIT
↑C

.SAVE ABC<cr>
JOB SAVED
↑C

.GET ABC<cr>
JOB SETUP
↑C

.E 137<cr>
000137/ 000002 000030      .@ST<cr>

;begins execution

EXIT
↑C

.DIR(F)   ABC/ALL<cr>

DIRECTORY        PROJECT 46          13:24          12-AUG-72

FILE NAME        KWDS KEPT           ACCESS         CREATED

ABC    /REL      0.25 K               F N           16-MAR-72
ABC    /SAV      0.38                 F N           22-AUG-72

TOTAL SIZE       0.63

EXIT
↑C
```

3. FACILITY ALLOCATION COMMANDS

The monitor allocates peripheral devices and core memory to users upon request and protects these allocated facilities from interference by other users. The monitor maintains a pool of available facilities from which a user can draw.

A user should never abandon a timesharing console without returning his allocated facilities to the monitor pool. Until a user returns his allocated facilities to the pool no other users may utilize them.

All devices controllable by the system have associated with them a physical name, consisting of three letters and zero to three numerals to specify the unit number. A logical device name may also be assigned by the user. This logical name of one to six alphanumeric characters of the user's choice is used synonymously with a physical device name in all references to the device. In writing a program, the user may use arbitrarily selected device names which he assigns to the most convenient physical devices at runtime. All references to devices in the monitor pool are made by physical names or by assigned logical names. When a device is assigned to a job, it is removed from the monitor's pool of available facilities. The device is returned to the pool when the user deassigns it or ends his job.

3.1 ASSIGN

ASSIGN $dev{=log-dev}

ASSIGN allocates an I/O device to the user's job for the duration of the job, or until a DEASSIGN command is given. ASSIGN may be abbreviated to AS.

$dev Device DSK or TTY. This argument is required.

log-dev A logical name assigned by the user (optional argument).

responses:

dev: ASSIGNED

The device has been successfully assigned to the job.

NO SUCH DEVICE

Device name does not exist.

ALREADY ASSIGNED TO JOB n

The device has already been assigned to another user's job.

LOGICAL NAME ALREADY IN USE DEVICE dev: The user has previously assigned this logical name to another device. This is a useful command in allocating logical device numbers to devices for Fortran programs. For example,

AS $DSK=6 will send output to the disk (as filename FORØ6) instead of the job output device,

AS $TTY=1Ø will receive/send data via the teletype instead of the disk.

3.2 DEASSIGN

DEASSIGN {$dev}

DEASSIGN returns one or more devices currently assigned to the user's job to the monitor's pool of available devices. DEASSIGN may be abbreviated to DEAS.

$dev If this argument is not specified, all devices assigned to the user's job are deassigned.

If this argument is specified, it can be either the logical or physical device name.

responses:

NO SUCH DEVICE

Device name does not exist.

DEVICE WASN'T ASSIGNED

The device isn't currently assigned to this job.

3.3 EOF

EOF $dev

EOF terminates any input or output currently in progress on the device.

$dev — The logical or physical name of the device on which I/O is to be terminated.

If no name is specified, I/O is terminated on all devices assigned to the job.

responses:

NO SUCH DEVICE

Either the device does not exist or it was not assigned to this job.

3.4 CORE

CORE {core}

CORE is used to modify the amount of core assigned to the user's job.

core = o — The low and high segments disappear from the job's virtual addressing space.

core > o — Total number of 1K blocks of core to be assigned to the job from this point on.
If this argument is omitted, the monitor types out the same response as when an error occurs, but does not change core assignment.

The response is

m+n/p CORE
VIR.CORE LEFT=v

where m=number of 1K blocks in low segment
n=number of 1K blocks in high segment
p=maximum K per job (max. physical user core)
v=number of K unassigned in core and swapping device.

3.5 RESOURCES

RESOURCES

RESOURCES will print out all the available devices in the system device pool (except teletypes) and the number of free blocks on the public disk packs. RESOURCES may be abbreviated to RES.

4. ADMINISTRATION COMMANDS

4.1 PJOB

The monitor responds by typing the job number to which the user's console is attached. If the console is not attached to a job, the monitor responds with

'LOGIN PLEASE'. PJOB may be abbreviated to PJ.

4.2 SYSTAT

SYSTAT {option job-no}

SYSTAT types out the status of the system: the system name, time of day, date, uptime, percent null time. It also gives:

Status of each job: job number, project number, TTY number, program name being run, size of low segment, state of program (RN=runable, TT=TTY input wait, ↑C = monitor command mode) and run time.

Status of high segments being used: name, directory name, size, number of users in core or on disk.

Status of each assigned device: name, job number, how assigned (AS = ASSIGN command, INIT = INIT UUO).

The arguments are optional; if omitted a full systat will be given.

The options specify various phases of the systat.

B	gives busy devices
D	dormant segments
J	job status
N	non-job status
S	short job status

job-no gives the job status relevant to a particular job

SYSTAT may be abbreviated to SYS.

4.3 HELP

HELP {name}

The HELP command prints out helpful documentation on various Computer Centre facilities.

If a name is specified, HELP will look for, and print out the information about the facility named in the command. For example, HELP MANUAL will print out the current status of the Computer Centre manuals.

HELP ALL will give a list of all currently available information. Only the first six characters of the argument are looked at. They must consist of the characters A-Z, 0-9.

Help is initially available for the following:

NEWS	;	the latest weekly newsletter
MANUAL	;	the up to date status of the Centre's manuals
LOGIN	;	how to login into the system.

APPENDIX

1. SEGMENTS and REENTRANT PROGRAMS

In a non-reentrant system, each user has a separate copy of a program even though a large part of it is the same as for other users. In a reentrant system, hardware allows a user area to be divided into two logical segments which may occupy non-contiguous areas in core. The monitor allows one of the segments of each user area to be the same as one or more other users, so that only one copy of a shared segment need exist no matter how many users are using it.

1.1 Segments

A segment is a continuous region of the user's core area maintained in core or on the swapping device. A program or user job is composed of one or two segments. A segment may contain instructions and/or data. The monitor determines the allocation and movement of segments in core and on the swapping device.

1.2 Sharable Segments

A sharable segment is a segment which is the same for many users. The monitor keeps only one copy in core and/or on the swapping device, no matter how many users are using it. A non-sharable segment is a segment which is different for each user in core and/or on the swapping device.

The PDP-10's hardware permits a user program to be composed of one or two segments at any point in time. The required low segment starts at user location 0. The optional high segment starts at user location 400000 or at the end of the low segment, whichever address is greater. The low segment contains the user's accumulators, Job Data area, instructions and/or data, I/O buffers, and DDT symbols. A user's core image is composed of a low segment, in multiples of 1K (1K=1024_{10} words), and a high segment also in multiples of 1K. A high segment may be sharable or non-sharable, whereas a low segment is always non-sharable.

1.3 Reentrant Programs

A reentrant program is always composed of two segments - a low segment which usually contains just data, and a high (sharable) segment which usually contains instructions and constants. The low segment is sometimes referred to as the impure segment. The sharable high segment, if write-protected, is referred to as the pure segment.

A one-segment non-reentrant program is composed of a single low segment containing instructions and data. A two-segment non-reentrant program is composed of a low segment and a non-sharable high segment.

The SAVE command supplies standard processor program names to the segments it creates. A one segment non-reentrant program has a processor program name /SAV. For two segment programs the low segment has the processor program /LOW. The high segment is named /HGH or /SHR depending on whether or not it is a sharable segment.

The S option in the SAVE command is used to make the high segment sharable when it is loaded with the GET command. To indicate this sharability, the high segment is written with processor program /SHR instead of /HGH. A subsequent GET will cause the high segment to be sharable. Because an error message is not given if the program does not have a high segment, a user can use this command to save without having to know which are sharable.

1.4 User's Core Storage

A user's core storage consists of blocks of memory whose sizes are an integral multiple of 1024_{10} (2000_8) words. There are two methods available to the user for loading his core area. The simplest way is to load a core image stored on disk (see GET). The other method is to use the relocatable binary loader to link-load binary files. The user may then write the core image on disk for future use (see LOAD and SAVE).

1.5 Job Data Area

The Job Data area provides storage for specific information of interest to both the monitor and the user. The first 140 (Octal) locations of the user's core area always are allocated to the Job Data area. Locations in this area have been given mnemonic assignments whose first three characters are .JB (or JOB). Therefore, all mnemonics referred to with a .JB (or JOB) prefix refer to locations in the Job Data area. It is planned eventually to replace the JOB prefix by .JB, but in the meantime either prefix is acceptable.

TABLE 1

Job Data Area Locations

(for user-program reference)

NAME	OCTAL LOCATION	DESCRIPTION
.JBUUO	40	User's location 40_8. Used for processing user UUO's (001 through 037). Op code and effective address are stored here.
.JB41	41	User's location 41_8. Contains the beginning address of the user's programmed operator service routine (usually a JSR or PUSHJ).
.JBERR	42	Left half: Unused at the present. Right half: Accumulated error count from one program to the next. Programs should be written to look at the right half only.
.JBREL	44	Left half: 0. Right half: The highest relative core location available to the user (i.e., the contents of the memory protection register when this is running).
.JBDDT	74	Contains the starting address of DDT. If contents are 0, DDT has not been loaded.
.JBHRL	115	Left half: First relative free location in the high segment (relative to the high segment origin so it is the same as the high segment length). Set by the LOADER and subsequent GETs, even if there is no file to initialize the low segment. The left half is a relative quantity because the high segment can appear at different user origins at the same time. The SAVE command uses this quantity to know how much to write from the high segment. Right half: Highest legal user address in the high segment. Set by the monitor every time the user starts to run. The word is $\geq$ 401777 unless there is no high segment, in which case it will be zero. The proper way to test if a high segment exists is to test this word for a non-zero value.
.JBSYM	116	Contains a pointer to the symbol table created by Linking Loader. Left half: Negative count of the length of the symbol table. Right half: Lowest register used.
.JBUSY	117	Contains a pointer to the undefined symbol table created by Linking Loader.

TABLE 1 (Cont)

Job Data Area Locations

(for user-program reference)

NAME	OCTAL LOCATION	DESCRIPTION
.JBSA	120	Left half: First free location in low segment (set by Loader). Right half: Starting address of the user's program.
.JBFF	121	Left half: 0 Right half: Address of the first free location following the low segment.
.JBREN	124	Left half: Unused at present. Right half: The reentry starting address. Set by user or by Linking Loader and used by @REE command as an alternate entry point.
.JBOPC	130	The previous contents of the user's program counter are stored here by Monitor upon execution of an @DDT, @REE, @ST, or @CST command.
.JBCHN	131	Left half: 0 address of first location after first FORTRAN IV loaded program. Right half: Address of first location after first FORTRAN IV Block Data.
.JBCOR	133	Left half: Highest location in low segment loaded with non-zero data. No low file written on SAVE if less than 140. Set by the loader. Right half: User argument on last SAVE or GET command. Set by the Monitor.
.JBVER	137	Left half: 1st digit represents who last edited program, next 3 digits give major version number, last 2 digits gives minor number (alpha). Right half: incremental edit number. The number is never converted to decimal . After a GET command, an E command can be used to find the version number.
.JBDA	140	The value of this symbol is the first location available to the user.

NOTE

Only those JOBDAT locations of significant importance to the user are given in this Table. JOBDAT locations not listed include those which are used by the Monitor and those which are unused at the present time. User programs should not refer to any locations not listed above since such locations are subject to change without notice.

Some locations in the Job Data area, such as .JBSA and .JBDDT, are set by the user's program for use by the monitor. Others, such as .JBREL, are set by the monitor for use by the user's program. In particular, the right half of .JBREL contains the highest legal address set by the monitor whenever the user's core allocation changes.

JOBDAT exists in binary form in the Systems Library for loading with user programs that refer to Job Data area locations symbolically. User macro programs must reference locations by means of the assigned mnemonics, which are declared as EXTERNAL references to the assembler. JOBDAT is loaded automatically, if needed, during the Loader's library search for undefined global references, and the values are assigned to the mnemonics.