Program Name: EMOP
Date: December 29th., 1971.
Language: PDP-9 Assembler (MACRO-9)
Purpose: Extended Monitor Package for use with PDP9-15 Advanced Software System
Keywords: I/O package, storage allocator, contingency handling, interpretive program execution

General Description

This package provides programs run under the PDP9-15 Advanced Software System (KM9-15) with facilities over and above those offered by the standard executive; these are:

Two alternative i/o interfaces.
A contingency handling mechanism.
A simple storage allocator.
Interpretive program execution.

The i/o interfaces offered are firstly, character-by-character sequential files accessed via four independently numbered streams each way; streams can be de- and re-selected at any point without loss of data. Secondly, line-by-line transfers between an array holding one character per word and a logical i/o channel represented by a .DAT slot. These options are mutually exclusive; the former is more demanding in storage and time both at assembly and run time. All transfers are in IOPS ASCII via the manufacturer's standard Keyboard Monitor.

Contingency handling is by an on statement which defines the action to be taken when some specified event occurs. This action may be any legitimate sequence of instructions with some restrictions if a jump out of sequence appears. A trap for re-entrant calls on the same event is included.

The storage allocator permits a specified number of storage words to be assigned to a named pointer and subsequently freed. Storage is recovered when it has (1), been designated 'free' and (2), all blocks nearer the free end of the stack are free; no attempt is made to move 'busy' blocks of storage. As a special case, a de-allocated but as yet unrecovered block is re-allocated immediately if it is exactly the size required by a fresh allocation command; to avoid fragmentation problems, it will not be subdivided to meet requests for smaller blocks of store.

Interpretive execution of a program can be specified when all address references and op-codes will be checked before being executed. The store is split into three zones to allow fairly stringent checking of a user program without unduly hampering external routines. A list is maintained at all times of the last twenty memory reference instructions executed, and is subsequently printed out when terminating the run of a faulty program.

To help where the user program merely loops indefinitely, this fault sequence can be artificially initiated by changing the accumulator switch setting.

Because it would reduce the stringency of address checking, program segments which are to be interpreted cannot use areas of store obtained through the storage allocator.

Options are independently selected except that the character-stream i/o interface uses the storage allocator and the /storage

storage allocator uses the contingency mechanism.

## Implementation

The package is implemented as a set of macro definitions containing calls on routines contained in a library file; for any particular assembly, only those macros' appropriate to the particular i/o option chosen are loaded. To make this selection, the symbol '%X' must be defined as zero(0) for line-by-line i/o and any non-zero value for character stream i/o; by default, %X=0.

When the character stream option is used, input(0:3) maps onto .dat(1:4) and output(0:3) onto .dat(5:10).

Note that if interpretive execution is specified, it will start at the point where the request appears in the source program; set-up routines are therefore not unnecessarily restricted.

## Operation

MACRO9-15's F option is used to read the macro definition file before processing the user program; if the character stream i/o option is required, use the P option to set %X to any non-zero value, e.g. %X=1.

If assembly listings are printed out, the G option may well be used to suppress all printing of macro expansions; if it is not, the C option should always be used to suppress the printing of unsatisfied conditionals.

No special care is needed at load time if the library items are in the main system library; if they are in the 'user library'(.LIBR5) an appropriate assignment must be made to .DAT -5.

In the former case, note that the interpreter(INT9) must precede all standard library items to obtain the correct layout in core.

The standard DEC Fortran compiler outputs code which may contain direct references to the 'floating accumulator' within the floating point package; if such code is to be interpreted, the floating point routines must be loaded into the user zone.

## Modification

Since its users access the routines in this package by calling predefined macros, it is easy to alter the format, and within limits, the effects, of the extra commands provided merely by changing the macro definition file; in this way it is possible, for example, to re-number the I/O streams if necessary.

Calling sequences for all routines in the package are given on the following pages.

## Macro Descriptions

### internal macros

**%M1** generate calling sequence for 2-parameter library routine.

```
%M1(i,j,k) => .GLOBL i; LAC j; JMS* i; .DSA k
```

e.g. READ(DAT,PNTR) => %M1(%RD,DAT,PNTR)

```
                    => .GLOBL %RD; LAC DAT; JMS* %RD; .DSA PNTR
```

**%M2** generate a 3-word directory entry block.

```
%M2(name,ext) =>
        %X=.;   .SIXBT name; 0
                .LOC %X+2; .SIXBT ext'SRC'
                .LOC %X+3
```

**%T1** for calls such as 'select input', 'select output' etc., 'input'/'output' is the first macro parameter. %T1 checks it for validity. ( input = 0, output = 4 )

```
%T1 =>
                .IFNZR -OUTPUT-1&stream
        ** FAULTY PARAM
                .ENDC
```

Part 1: READ, WRITE, PACK, UNPACK, SEEK, ENTER, CLOSE

name: READ/WRITE dat,pntr

e.g. READ (1),(BUFF)
WRITE (2),P

Read/write one IOPS line buffer, including headers, starting at the point in store specified by the contents of pointer pntr and transferring from/to the .DAT slot indicated by the contents of dat. Normally AC=777777(8) on return; AC=0 means end-of-file defined as IOPS code 5 or 6.

Expansions:
READ(dat,pntr) => %M1(%RD,dat,pntr)
WRITE(dat,pntr) => %M1(%WR,dat,pntr)

name: PACK/UNPACK from,to

e.g. PACK (BUFF1),(BUFF2)
UNPACK P,Q

Convert between 5/7('IOPS') ASCII line format and a rather more convenient format peculiar to this package. In this latter, characters are held, right-justified, one per machine word, preceded by a single header word which specifies the (+ve) number of characters in the line exclusive of the header. No terminating character is used; any carriage returns are removed on input and one supplied automatically on output.

```
  8  o  n  e     l  i  n  e
:--:--:--:--:--:--:--:--:--:
0  1  2  3  4  5  6  7  8
```

In performing the conversion, data is moved from the place specified by from to that specified by to. Since IOPS ASCII is more compact than the format used here, PACK will work correctly when these two are in fact the same; this is not generally true for UNPACK.

Expansion:
PACK(from,to) => %M1(%PK,from,to)
UNPACK(from,to) => %M1(%UN,from,to)

name: SEEK/ENTER dat,name,ext

e.g. SEEK (2),'FRED'
ENTER (1),'FRED','NEW'

Connect an input file(SEEK) or output file(ENTER) with the name and extension given, on the bulk storage device attached to the specified DAT slot; if unspecified, the extension is 'SRC' by default.
If a hard-copy device is attached to the .DAT slot specified, the /file-name

file-name information is redundant and this command merely ignored.

Expansion:

```
SEEK(dat,name,ext) =>
                    %M1(%SE,dat,(.+2))
                    JMP .+4; %M2(name,ext)

ENTER(dat,name,ext) =>
                    %M1(%SE,dat,400000 (.+2) )
                    JMP .+4; %M2(name,ext)
```

name: CLOSE dat

e.g. CLOSE (2)
CLOSE N

Close off the file(input or output) attached to the .dat slot specified.

Expansion:

```
CLOSE(dat) => .GLOBL %CS; LAC dat; JMS* %CS
```

Part 2: ON, EVENT

name: ON event,action

e.g. ON 5,<DZM P; DZM Q>

Precondition the contingency handling mechanism so that when event occurs the sequence of instructions specified as action is entered. A trap on re-entrant calling of any particular contingency routine is included.

To implement this trap, a marker is set on entry to and cleared on exit from each contingency routine; the routine should therefore not itself contain any branches back to the interrupted process. Failure to observe this limitation will result in the above-mentioned marker not being cleared and possibly in spurious 're-entrant call' faults being signalled. This can be avoided by resetting the the contingency trap with an ON statement at the point to which the branch is made.

Currently event numbers from zero to fifteen are permissible with zero having the special property that it cannot be trapped and so always causes the job to be terminated; it is thus especially suited to signalling the occurrence of terminal error conditions.

Note below that a fault message can be specified independently of the event number, thus allowing a range of error conditions to be grouped into one category yet still retain their identity for diagnostic purposes.

Expansion:

```
        ON(event,action) =>
                        .GLOBL %ON,%EV
                        LAC (event); JMS* %ON; JMP T2+1
                        0
                T1      0
                        action
                        DZM T1-1
                T2      JMP* T1
```

name: EVENT number,text

e.g. EVENT 2,<INPUT ENDED>

This macro causes a contingency routine defined by a previous ON statement to be entered if such a routine exists. If it does not a message of the form:

```
                        *EVENT number           text
```

is printed out and the Keyboard Monitor reloaded.

Expansion:

```
        EVENT(number,text) =>
                        LAC number; .GLOBL %EV; JMS* %EV; JMP TAG
                        .ASCII '@text@'<15>
                TAG=.
```

Part 3: ALLOT, FREE

name: ALLOT size,pntr

e.g. ALLOT (500),P
ALLOT N,P

Size machine words are taken from a stack and pntr made to point to the base of this block.

Expansion:
ALLOT(size,pntr) => .GLOBL %AL; LAC size; JMS* %AL; DAC pntr

name: FREE pntr

e.g. FREE P

De-allocate the storage area assigned to pntr; de-allocated areas are recovered for re-allocation whenever they are at the free end of the stack. No attempt is made to move 'busy' areas or to sub-divide de-allocated but as yet unrecovered areas; they will however be re-allocated if they are of the exact size required by a new ALLOT statement.

Expansion:
FREE(pntr) => .GLOBL %AL,%FR; LAC pntr; JMS* %FR

Part 4: ATTACH, SELECT, RSYM, PSYM, CLOSE

name: ATTACH stream,number,name,ext

```
e.g.    ATTACH INPUT,2,'FRED'
        ATTACH OUTPUT,3
```

Attach an input/output file with the appropriate name to the stream specified and generate the necessary .IODEV's and .INIT's; it is unfortunately necessary for number to be defined at assembly time to make this possible.

The data transfer direction is specified by one of two predefined symbols, INPUT=0 and OUTPUT=4.

Expansion:

```
ATTACH(stream,number,name,ext) =>
                .GLOBL %AT; %T 1 stream
        %X=OUTPUT-1&number+stream; LAC (%X)
        %X=%X+1; .IODEV %X
                JMS* %AT; .DSA (.+2); JMP .+4
                %M2 name,ext
```

name: SELECT stream,no

```
e.g.    SELECT INPUT,(2)
        SELECT OUTPUT,N
```

Select the input/output stream to be used by RSYM/PSYM, claiming a line-buffer from the storage allocator if need be. Note that, unlike IMP, no default streams are selected on entry; like PDP9-15 IMP but unlike ERCC IMP, streams may be de- and re-selected at any point without loss of data.

A stream must be ATTACH'ed before any attempt is made to select it and SELECT'ed before RSYM/PSYM is used.

Expansion:

```
SELECT(stream,no) =>
                .GLOBL %AT,%SL; %T 1 stream
                LAC no; JMS* %SL
                .IFZER stream; .GLOBL %IO; .DSA %IO; .ENDC
                .IFNZR stream; .GLOBL %OO; XCT %OO; .ENDC
```

name: RSYM name

e.g. RSYM S

Fetch the next symbol from the currently selected input stream and place it in the address specified by name; successive calls access successive symbols in the input stream.

Expansion:

```
RSYM(name) => .GLOBL %AT,%RS; JMS* %RS; DAC name
```

name: PSYM value

e.g. PSYM S

Output, to the currently selected output stream, the symbol whose seven-bit ISO value is specified by the least significant seven bits of VALUE.

Expansion:

```
PSYM(value) => .GLOBL %AT,%PS; LAC value; JMS* %PS
```

name: CLOSE stream

e.g.
```
CLOSE INPUT
CLOSE OUTPUT
```

If input, notionally rewind the file attached to the currently selected stream, if output, flush the output buffer and make any necessary directory entries. In both cases, de-allocate the line buffer used by that stream.

Expansion:

```
CLOSE(stream) =>
                .GLOBL %AT,%CL; %T1 stream
                .IFZER stream; .GLOBL %IO; LAC %IO; .ENDC
                .IFNZR stream; .GLOBL %OO; LAC %OO; .ENDC
                JMS* %CL
```

Part 5: .TRACE

name: .TRACE

e.g. .TRACE

Load the interpreter(INT9) and enter it if the AC switches are non-zero; this test makes it possible to obtain a non-interpretive run quickly, simply and without altering the user program or the absolute position in store of the various load modules.

INT9 cannot be re-entered after its initial call; note especially that if INT9 and DDT are both loaded it is possible to proceed from the point currently reached($P) but not to restart($G). The whole area between INT9 and the bottom of the bootstrap is taken to be 'user program'; jumps into DDT('breakpoints') are therefore correctly executed but DDT is unprotected and can be overwritten.

INT9 applies the checks detailed below before executing each instruction; when a fault is detected, a list is printed of the last twenty memory reference instructions executed along with values for program counter, transfer vector if indirect, and the contents of the addressed location before the instruction was executed. To help in dealing with cases where the user program just loops indefinitely, INT9 can be forced to enter this fault sequence at any time by changing the setting of the accumulator switches; as a consequence, while any non-zero initial value may be set on these switches, it cannot be changed without aborting the program.

Address checking is as follows:

1) The store is regarded as consisting of three zones; the checks applied depend on which of these zones the program counter is currently in. Irrespective of what checks are or are not applied, once the interpreter has been entered, it can only be left by terminating the run or, albeit temporarily, to service a hardware interrupt. A list of the last twenty memory reference instructions executed is maintained at all times whether or not fault checking is performed.

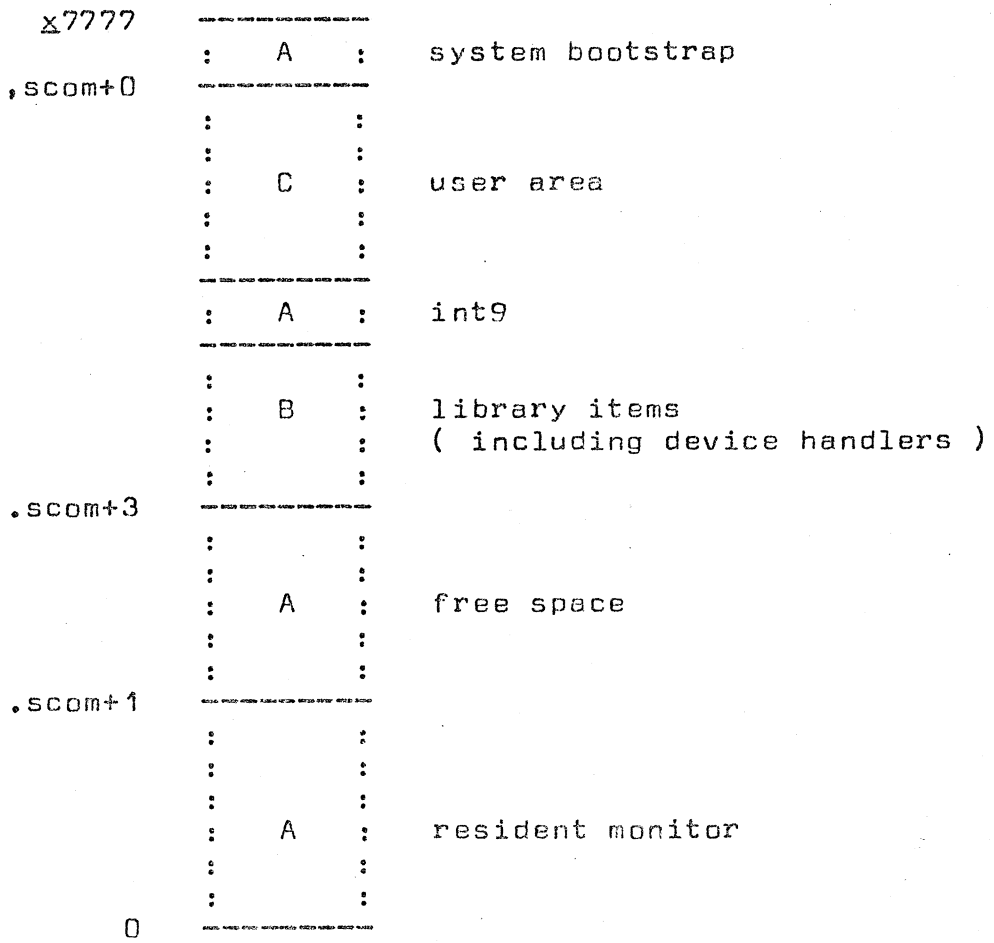
2) If PC is in the user program area(zone C), all address references outwith this zone are trapped except that a JMS, direct or indirect, is permitted into zone B.
CAL's are trapped except for .EXIT which is executed directly.

3) If PC is in zone B, typically consisting of library items and other tested program units, all instructions and address references are permitted. Note that since the CAL and DBR instructions have to be simulated, the interpreter does not work properly with API enabled.

4) Zone A is defined as everything not included in B or C.
EAE instructions are illegal and are trapped wherever they occur, as are the operate group and indirect-but-not-JMP* after a DBR.

Expansion:

.TRACE => .GLOBL %I9; LAS; SZA; JMS* %I9




where x depends on the system size.

## Error Conditions

A list is given below of trappable error conditions which may arise within the the package routines; various others not listed here may occur but are untrappable and cause a message to be typed on the console.

| | |
|---|---|
| READ: | AC=0 on return signals end-of-file. |
| ALLOT: | event 1 signals store overflow. |
| RSYM: | event 2 signals end-of-file. |
| SELECT: | SELECT INPUT may cause ALLOT to signal event 1 if more buffer space is required but unavailable. |