SLAC-TN-75-7
March 1975

PDP8 SIMULATOR

K. B. Mallory

PDPSIM is an assembly-language program written for the 360-91 to allow writing and proving an executive program for the accelerator's PDP-8's.(Ref. 1) It allowed writing and debugging the executive, the loader and the interprocessor link handlers before the first of the 9 PDP-8's was delivered. It has been updated for use in the triplex system; and still is used for assembling and debugging the code used online. It is also used for PDP-8 programming classes.

Some parts of the program are specialized to the development of the executive system and the software for SLAC's I/O devices for accelerator control. Code for additional devices is easily added when required.

The simulator program is in several parts: They include a control section, an assembler, a section for defining configurations of several processors with links and associated I/O devices, and an execution section (the simulator itself).

I. CONTROL PROGRAM

The program is directed by control statements identified by '**' in columns 1/2. The ASM, PAL, and LOAD control statements have fixed data fields, described below. The remaining control statements are free-field. The operation field starts in column 3 and is terminated by the next blank character. The operation is identified only by the letter in column 3. (The following character, represented by "$" in some of the examples below, represents the processor ID in a multiprocessor system. Otherwise, the remaining characters of the operation field are ignored.) Parameters are recognized as octal numbers or strings depending on the first character of the parameter. Parameters are delimited by spaces. Parameters are positional, and extra parameters are treated as comments.

ASSEMBLE: (fixed field)

```
 **A   NN
 **ASM NN <comment>
 **A     <comment>
```

The first form shows all of the characters tested by the program. The two characters in columns 7/8 are taken to be the program name. The program NN will be assembled and the binary output will be saved in a 'DISK' buffer. If the name field is blank, the binary output will not be saved.

The source program must follow the control statement. All statements till the next control statement or end of the input file are copied onto a utility file by the control program, and the assembler is then called. If the source deck does not contain a .END statement, the job will be terminated. The assembly is done in place in 'core', and may be executed without using a LOAD control statement. If the source contains no .ORG statement, the assembly will be started at location 0200.

Multiple sources may be assembled, but each must be preceded by an ASSEMBLE control statement. The 'core' is not cleared between assemblies, so that a group of assembled programs may be executed without reloading 'core'. The symbol table is cleared after each assembly (but see SYMBOLS below).

Provision has been made for saving the assembly output from one job to another in an external "DISK" file.

EXECUTE:

```
 **EXE  <time>
 **E$
```

The EXECUTE statement is the 'continue' switch, to resume operation after a pause. If the <time> parameter is written, execution will continue for that number of milliseconds and then pause: If the time parameter is omitted, execution will simulate approximately 1 second of CPU time and then pause. Continuation requires another **E control statement.

FINISH:

```
 **F
```

This statement terminates the job. It is issued by the program automatically if a source deck has no .END statement or if the end of input file is reached.

GO:

```
 **G$  <4-digit start address>   <time>
 **G   2400
```

The GO statement specifies the start address and initiates execution. Location 0000 is used if the start address field is blank. The time parameter is the same as in the EXECUTE statement.

LOAD: (fixed field)

```
 **LOAD  <address>  NN  NP  NQ  <...>
 **L   <address>  NN
 **L$  <address> NN NP
```

The LOAD statement specifies that a particular program NN shall be loaded into a specified address in "core". The address is specified by an octal number following **L. Multiple program names on a load statement will be loaded into successive pages in "core".

```
PAL:  (fixed field)
 **P   NN
 **PAL
```

This command produces an assembly using a language resembling the PAL assembler. It is otherwise the same as the ASSEMBLE statement.

```
SYMBOLS:
 **S
 **SYM
```

This command allows extension of the permanent symbol table of the assembler. It is followed by a source deck of symbol definitions ('tag= val' statements) and a .END statement. A standard assembly is performed, but the symbols are retained for the rest of the job. The source should contain no literals or labeled statements.

The remaining control statements are somewhat peculiar to the development of the DS/8 executive and a multiprocessor system.

```
KEYBOARD:
 **KEY <string>
 **K$ <string>
```

The KEYBOARD statement allows simulation of tty keyboard input. The characters following the first blank in the line are presented one-at-a-time to the device at device address 03. A carriage return is inserted after the last non-blank character in the string.

```
DEBUG:
 **D$ <debug options>
 **D  S
 **D STEP
 **D  R
 **D  B  <address>
 **D  T  <addr1>  <addr2>
 **D  W  <addr1> <addr2>
 **D  P <address> <value>  <... ...>
 **D  D <addr1>  <addr2>
```

Debug options are identified by the first character of the option parameter. Remaining characters in the option parm are ignored.

STEP: The program-counter value, the instruction and the contents of the accumulator are printed for every instruction executed by the simulator.

RUN: The "step feature" is turned off.

BREAK: The simulator will pause before executing an instruction at the specified address. The break allows changing step and trace addresses, or dumping core contents. Execution may be then resumed in response to a **E command. If no address is provided, the break feature is turned off.

TRACE: The PC, instruction and AC are printed for every instruction executed in a range of addresses. If the second address is omitted, only one location will be traced. If no address is furnished, the trace is turned off. Trace lines are marked 'T' to distinguish them from other lines generated by the STEP feature.

WATCH: The PC, instruction and AC are printed when a memory-reference is made to an address or range of addresses, as above. The lines are identified by 'W'.

PATCH: A value is inserted at the indicated address. More than one address-value pair of parameters msy be written on one command.

DUMP: An octal dump is produced, covering the address range specified. Execution may be resumed following a **E command.

CONFIGURATION:

```
 **C$    <device parameters>
```

CONFIGURATION statements are used to define additional processors and additional I/O devices. At this date, a number of options have been written to add additional teletypes to a processor, to define additional processors, to connect links between processors, etc.

II. ASSEMBLER

The DS/8 assembler has two modes of operation: one resembles the PAL assembler, the other resembles the MACRO assembler used in the PDP9. The latter mode is the principal mode, called by the ASM control statement. The PAL statement introduces minor language modifications discussed in a later section.

Since the exclamation point does not exist on IBM's normal printer train, it has been replaced by the symbol '¦'. The symbol '¬' is used to represent the backward slash.

Each statement may contain optional label, operator, operand and comment fields, delimited by spaces. The label, if included, must start in column 1. The comment field should be preceded by a slash. If column 1 of a statement is a slash, the entire statement is treated as a comment.

Labels must be identifiers: an identifier consists of 1 to 6 alphanumeric characters with the first character alpha, or of a dot followed by 1 to 5 alphanumeric characters.

The operator and operand fields may be expressions. Each is evaluated and the two expressions are then merged. If the operator contains a memory-reference instruction (AND, TAD, ISZ, DCA, JMS or JMP) the value of the operand field must lie on page zero or on the page currently being assembled. The terms of an expression may be identifiers, numbers, or the location counter reference ('.'). Expressions are evaluated from left to right. The operators + and - are recognized as arithmetic operations and | as a merge operation. A star is taken to be an indirect address value to be merged with the operator.

A value or instruction enclosed in parentheses will be assembled as a literal value, and the address of the literal will be assembled in place of the parenthesis. Spaces may be imbedded within the parentheses: TAD (JMP X) is a valid instruction referencing a literal. Literals may not be nested, but they can be used within definitions.

A label may be defined equal to an identifier, an expression or an instruction. The form of the definition is

LABEL= <value>

In .SIXBT mode, text characters enclosed in quotes will be assembled, two characters per word, in six-bit ASCII. A single character or the last of an odd number of characters in a quoted string will be right-justified in a word. A quoted string of one or two characters may be used as the argument of a definition or a literal.

The following operation codes are defined to the assembler:

```
   AND  TAD  ISZ  DCA  JMS  JMP  IOT  NOP  SKP  SMA
   SPA  SZA  SNA  SNL  SZL  HLT  OSR  CLA  CLL  CMA
   CML  RAR  RTR  RAL  RTL  IAC  BSW  MQL  MQA  CAM
   SWP  ACL
```

The following pseudo-operations are recognized by the assembler:

.ORG followed by an expression sets the location counter.

.LIT forces assembly of literals. Since literals are inserted in the module during the second pass, the length of the literal pool is not known during pass one. The programmer must therefore ensure that .LIT is followed by a .ORG statement that provides enough space for the literals.

.DEC causes the assembler to recognize numbers as decimal in lines which follow.

.OCT causes the assembler to recognize numbers as octal in lines which follow. The assembler is initialized for .OCT for each assembly.

.ASCII causes the assembler to assemble one 8-bit character per word in strings.

.SIXBT causes the assembler to pack strings into two six-bit (truncated) ASCII characters per word. Each assembly is initialized in .SIXBT mode.

.END marks the logical end of a source program.

IIa. PAL

The following modifications are made to emulate the PAL assembler:

Labels must be followed immediately by a comma. The operation field may start in column 1. An instruction may have any number of operands separated by spaces, which will be merged with the operator. The comment field must be preceded by a slash.

The indirect address is represented by the character 'I' following a memory-reference instruction. (For example, JMP I LAB). Do not usi 'I' as an identifier.

The symbols '*' and '|' are not recognized.

The logical end of a source program is indicated by '$' in column 1 of the last statement.

III. EXECUTION

The execution program simulates all CPU functions of a PDP-8, but the I/O instructions have been specialized for the needs of the executive development. The following I/O commands have been implemented:

ION=	IOT 1	Turn interrupt on.
IOF=	IOT 2	Turn interrupt off
Teletype:		
KSF=	IOT 31	Skip on keyboard flag
KCF=	IOT 32	Clear keyboard flag
KRS=	IOT 34	Keyboard read, static
	IOT 35	Enable TTY interrupt
KRB=	IOT 36	Clear flag and AC, and read keyboard buffer
TSF=	IOT 41	Skip on TTY flag
TCF=	IOT 42	Clear TTY flag
TLS=	IOT 46	Print char and clear TTY flag
Real-time clock:		
SCF=	IOT 141	Skip on clock flag
CCF=	IOT 142	Clear clock flag
CLON=	IOT 145	Enable 360 PPS clock
Link interface:		
STF=	IOT 401	Skip on transmit flag
CTF=	IOT 402	Clear transmit flag
TAC=	IOT 404	Transmit a character
SRF=	IOT 411	Skip on receive flag
CRF=	IOT 412	Clear receive flag
RCB=	IOT 414	Read character buffer

The following I/O instructions are specialized for debugging in the Simulator system. They have no equivalent in a real operational system.

.STP= IOT 24 SET STEP MODE

This instruction is the equivalent of the 'STEP' switch on the computer console. The contents of the PC, the AC, and the next instruction value in core are printed for every fetch cycle. If this instruction is executed again, the program halts.

.RUN= IOT 22 CLEAR STEP MODE

This instrucction is the 'RUN' position and nullifies the 'STEP' instruction.

JMP .-1

This instruction always causes the simulator to pause and wait for an interrupt flag to be set or for the specified execution time to expire. Its purpose is to avoid executing thousands of IBM computer cycles when the program is known to be in a "wait" state.

IV. OPERATION OF PROGRAM

Sample JCL and control cards for operation of the program are shown below:

```
  // JOB ,CLASS=E
  // EXEC LOADGO
```

```
//SYSLIN DD DSN=WYL.IC.KBM.PDPGM,DCB=BLKSIZE=3200,DISP=OLD
//DISK DD DUMMY,DCB=BLKSIZE=3432,SPACE=(3432,10)
//UTIL DD UNIT=SYSDA,SPACE=(TRK,(1,1)),
//       DCB=(RECFM=FB,LRECL=80,BLKSIZE=3520)
//SYSUDUMP DD SYSOUT=A
//SYSIN DD *
**ASM SA
TLS=6046
TSF=6041
<source program>
**L   200  SA
**GO  0200
**FIN
```

The program normally will terminate after about 4500 lines of print. The normal completion code is 30. If the program abends with code 322 or 722, it is probably due to looping in the user program. Please show me any other abend.

KBM

Reference:

1. TN 75-6: "Sam Howry's DS executive translated for a PDP-8", K.B.Mallory