



STANFORD RESEARCH INSTITUTE  
Menlo Park, California 94025 · U.S.A.

April 1970

PDP-15 SIMULATOR

by

John K. Ellis  
Leonard J. Chaitin

Artificial Intelligence Group

Technical Note 25

SRI Project 8259

This work was supported by the Advanced Research Projects  
Agency and the National Aeronautics and Space Administration  
under Contract NAS12-2221.

#### ABSTRACT

This report describes briefly a PDP-15 simulator and its assembler, both of which were written on the PDP-10. The instruction repertoire for the simulator is complete except for input/output transfer instructions. The assembler provides an optional assembly and symbol table listing but currently has no pseudo-op or macro capabilities.

## OPERATING PROCEDURE

Transfer the binary file COD15.DAT from the DECTape to the disk using PIP:

```
.R PIP]  
*DSK:/B/X-DTAn:COD15.DAT]
```

On the DECTape are core images of DDT loaded with three different assembler/simulator combinations:

- SMLTR1.SAV Assembles and loads PDP-15 code. SMLTR1.SAV optionally outputs binary code to a disk file called BIN15.DAT and can be used to obtain an optional assembly listing together with an optional alphabetized symbol table listing.
- SMLTR2.SAV Assembles and loads PDP-15 code. SMLTR2.SAV optionally outputs binary code to a disk file called BIN15.DAT but makes no provision for any kind of listing; however, SMLTR2.SAV does run in only 24K of memory and should be used if 78K of memory is not available.
- SMLTR.SAV Takes the place of a loading procedure and uses the binary file BIN15.DAT generated by either SMLTR1.SAV or SMLTR2.SAV.

If 78K of memory is available, use Procedure 1 described below; otherwise use Procedure 2, which requires only 24K of memory. In either case the program assumes that the code to be assembled is located on a disk file called PGM15.DAT. Procedure 1 or 2 should be used once for each different PGM15.DAT; Procedure 3 can then be followed as long as PGM15.DAT is not changed.

### 1. SMLTR1.SAV (78K)

The following sequence of commands turns control over to DDT:

```
.GET DTAn SMLTR1.SAV]  
JOB SETUP]  
]  
.DDT]
```

---

The following notation is observed in console examples: computer typeouts are underlined, ] denotes a carriage return, \$ designates the ALTMODE key, and !C means control C.

Insert the appropriate breakpoints (see section called Debugging Technique); then type \$G. The assembler will print out the three messages shown below.

OUTPUT? TYPE Y OR N

If output to the disk file BIN15.DAT is desired, type a Y followed by a carriage return. A disk file called BIN15.DAT is created and used later in Procedure 3. If output is not desired, type an N followed by a carriage return.

LISTING? TYPE Y OR N

If an assembly listing is desired, type a Y followed by a carriage return. The assembler will output the following type of listing on the line printer:

<u>Sequence No.</u>	<u>Memory Location</u>	<u>Octal Code</u>	<u>PDP-15 Code</u>
10	100 <sub>g</sub>	707724	EBA;
20	101 <sub>g</sub>	111620	PASS1: JMS TTWRIT;
30	102 <sub>g</sub>	000204	CAL M9MSG;
⋮	⋮	⋮	⋮
52690	12534 <sub>g</sub>	000000	SRCBUF: BLOCK 44;
52700	12600 <sub>g</sub>	000000	PTPBUF: BLOCK 1000;
52710	13600 <sub>g</sub>	000000	END;

If an assembly listing is not desired, type an N followed by a carriage return.

INDEX? TYPE Y OR N

If an alphabetized symbol table is desired, type a Y followed by a carriage return. The assembler will output a symbol table listing on the line printer. If a symbol table is not desired, type an N followed by a carriage return.

2. SMLTR2.SAV (24K)

The following sequence of commands turns control over to DDT:

```
.GET DTAn SMLTR2.SAV]
JOB SETUP.]
.]
.DDT.]
```

Insert the appropriate breakpoints (see Debugging Technique); then type \$G. The assembler will print out the following message:

OUTPUT? TYPE Y OR N

If output to the disk file BIN15.DAT is desired, type a Y followed by a carriage return. A disk file called BIN15.DAT is created and used later in Procedure 3. If output is not desired, type an N followed by a carriage return.

### 3. SMLTR.SAV (18K)

The following sequence of commands turns control over to DDT:

```
.GET DTAn SMLTR.SAV]
JOB SETUP.]
↓
.DDT.]
```

Insert the appropriate breakpoints (see Debugging Technique); then type \$G.

### DEBUGGING TECHNIQUE

Each time the simulator encounters a HLT instruction in a PDP-15 program, it prints out the following message on the teletype:

```
DEBUGGING? TYPE Y OR N
```

If you are trying to debug a PDP-15 program, type a Y followed by a carriage return. The PDP-15 instruction HLT will jump to memory location QNOP in the simulator where a breakpoint can be set using DDT. By varying the location of HLT instructions, DDT can then be used to assist in debugging a PDP-15 program. If you want to use this feature, set a breakpoint at memory location QNOP in the simulator, replace the given PDP-15 instruction by a HLT instruction, and replace the contents of memory location QLOC in the simulator by the given PDP-15 instruction. If, after a breakpoint halt occurs at memory location QNOP in the simulator, you want the position of the HLT instruction to stay the same, simply do an \$P. However, if you want to change the position of the HLT instruction, set a breakpoint at memory location GETNXT+2 in the simulator. Do an \$P, and when a breakpoint halt occurs at memory location GETNXT+2 in the simulator, replace the HLT instruction with the contents of memory location QLOC in the simulator, remove the breakpoint at memory location GETNXT+2 in the simulator, and proceed as above.

If you are not trying to debug a program, type an N followed by a carriage return. The PDP-15 instruction HLT will jump to memory location COMMON in the simulator, where the contents of the three timers are printed out, thus:

```
PDP15 TIME 5466.40 MICROSECONDS.]
IO WAIT TIME 2886 MILLISECONDS.]
RUNTIME 0 MIN, 0.50 SEC.]
↓
EXIT.]
↑C
↓
.
```

PDP15 TIME shows the total execute time required by the PDP-15 program in microseconds. Since the automatic priority interrupt system is not currently simulated, input/output is accomplished using program-controlled transfers. The input instructions KRB,KSF and the output instructions TSF,TLS are currently programmed to be used together in the following manner only:

```

        TSF;                KSF;
        JMP .-1;            JMP .-1;
        TLS;                KRB;

```

IO WAIT TIME, therefore, shows the total teletype input/output wait time required by the PDP-15 program in milliseconds. RUNTIME shows the actual time required by the simulator to run the PDP-15 program.

#### SIMULATOR ERROR MESSAGES

If an illegal instruction or an illegal memory reference is encountered in a PDP-15 program, the simulator types out, respectively, the following two error messages:

```

ILLEGAL INSTRUCTION AT LOCATION 137]
ILLEGAL MEMORY REFERENCE AT LOCATION 137]

```

together with the contents of the three timers (see above). Since the simulated program counter is decremented in either case, the location given corresponds to the actual PDP-15 instruction which caused the error message to be printed out on the teletype.

#### SIMULATOR DESCRIPTION

The size of the simulated machine can be varied from 1K to 128K of memory by adjusting memory location SIZE.

The assembler generates code which the simulator assumes to be in the following format:

```

PDP15:      data,,garbage
PDP15+1g:   data,,garbage
           :
           :
PDP15+77g:  data,,garbage
PDP15+100g: data,,garbage
PDP15+101g: data,,garbage
           :
           :
PDP15+    g: data,,garbage

```

} Reserved addresses

} PDP-15 program

The PDP-15 program is loaded beginning with memory location PDP15+100g (i.e. location 100g in the simulated machine).

The right half of each data word is then zeroed so that the code is in the following format:

```

PDP15:      data,,0
PDP15+18:  data,,0
      :
      :
PDP15+778: data,,0
PDP15+1008: data,,0
PDP15+1018: data,,0
      :
      :
PDP15+      8: data,,0

```

} Reserved addresses

} PDP-15 program

Unless the normal program sequence is altered, program control is determined by the following two instructions:

```

GETNXT: MOVE T,PDP15(PC)
        JSR INCPC

```

The program counter (PC) is originally set to 100<sub>8</sub>. INCPC is a subroutine that increments the program counter modulo 4096 (modulo 8192 if bank mode addressing is in effect).

The simulator decodes PDP-15 instructions by scanning in the order shown below until one of two things happens: (1) the instruction is discovered to be a legal, currently implemented PDP-15 instruction, in which case the program jumps to the appropriate subroutine, or (2) the instruction is discovered to be an illegal or currently unimplemented PDP-15 instruction, in which case the program jumps to an illegal instruction subroutine.

The program checks the contents of accumulator T for one of six input/output instructions now implemented: TSF, TLS, KSF, KRB, EBA, or DBA. If T doesn't contain an input/output instruction, the program jumps on the operation code (bits 0-3):

```

ENTRY:  ROT T,4
        JRST @.+1(T)

```

to a subroutine corresponding to a memory reference instruction or to one of three general instruction groups: index operate/input/output instructions (IO), EAE instructions (EAE), or microcoded instructions (MCRCD).

#### 1. Memory Reference Instructions

All memory reference instructions JSR to the subroutine MODE. If bank mode addressing is in effect (memory location BNKFLG is set to all 1's) the program JRST's to the subroutine BNKMOD. If bank mode addressing is not in effect (memory location BNKFLG is set to 0) the program jumps on the address mode:

```

MODE: 0
      HRRI T,0
      SKIPE BNKFLG
      JRST BNKMOD
      ROT T,2
      JRST @.+1(T)

```

to subroutines which handle one of four address mode combinations: direct (NONE), indexed (BIT5ON), indirect (BIT4ON), or indirect-indexed (BOTH). Any one of these combinations can be handled by the following three subroutines: current page address (CPADR), indexing (INDEX), and indirect addressing (INDRCT). INDEX and INDRCT both JSR to the subroutine MEMORY, which JRST's to an illegal memory reference subroutine if the memory capacity (depending upon memory location SIZE) is exceeded.

## 2. Index Operate/Input/Output Instructions (IO)

If bank mode addressing is in effect, the program JRST's to an illegal instruction subroutine, since all indexing operations are eliminated in favor of bank addressing. If bank mode addressing is not in effect, the program tests for input/output instructions; any input/output instruction discovered at this point is not currently implemented and causes a JRST to an illegal instruction subroutine.

The program then jumps on bits 5-8 to the appropriate subroutines:

```

IO:   SKIPN BNKFLG
      TLZN T,400000
      JRST ILLGLI
      HRRI T,0
      ROT T,5
      JRST @.+1(T)

```

## 3. EAE Instructions (EAE)

Figure 7-8 and Table 7-2 in the PDP-9 User Handbook illustrate EAE instruction microcoding. The EAE instructions are microcoded using only logical test instructions:

```

EAE:  TLZE T,400000      SHAL:  ADDI TIMER,442
      JRST SHAL          :
XCLQ: TLZE T,200000      JRST XCLQ

```

together with a switch which jumps on bits 9-11:

```

BRNCH: HRRI T,0
      ROT T,10
      JRST @.+1(T)

```

#### 4. Microcoded Instructions (MCRCD)

Figure 7-9 in the PDP-9 User Handbook illustrates the microcoded instructions; however, the actual scanning sequence appears to be (from left to right):

0=ORof 1=ANDof	SNL SZA SMA SZL SNA SPA	CLA	CLL	OAS	CML	CMA	Bit7=0 Bit7=1	RAR RAL RTR RTL	HLT
8	9 10 11	5	6	15	16	17	7	13 14	12

which is undocumented but consistent with all available PDP-9/PDP-15 documentation.

The microcoded instructions are microcoded using only logical test instructions in a manner analogous to the EAE instructions. The microcoded instruction OAS (OR Console Accumulator Switches to the Accumulator) is not currently implemented and generates an illegal instruction.

#### USING THE ASSEMBLER

The assembler (until we get MACRO-15 working) effects to find a disk file called PGM15.DAT. For editing purposes this file should have sequential line numbers incremented by 10, thus:

```

10      ABC: LAC .+2;
20      DAC ABC#;
30      HLT;
40      DEF: DATA 777;

```

The assembler will take one instruction per line. Each instruction must end with a semicolon. The format is free field--all spaces are ignored. There are three fields per instruction: the label field (optional), the op-code field, and the variable field.

The label field (if present) is identified by a colon following the label. No more than six characters are permitted; the first character must be a letter. Only letters and numbers are permitted.

The op-code field contains any legal PDP-15 code (as defined in the manual) plus the pseudo-ops DATA, BLOCK, and END. The DATA instruction generates a word of numeric data. The BLOCK instruction reserves a block of zeroed words. The size of the block is designated in the variable field. The END instruction must be the last instruction in a program and must be present.

The variable field may contain a label, a period, or a number, optionally followed by a + or - followed by another number. All numbers are taken to be octal. If a decimal number is wanted, it must be followed by a \$. The period stands for the current setting of the location counter, which is initialized to 100<sub>8</sub>. The variable field may be preceded by an \*, which indicates indirect addressing, and may be followed by a #, which indicates indexing.