ATTENTION

This is a USER program. Other than requiring that it conform to submittal and review standards, no quality control has been imposed upon this program by DECUS.

The DECUS Program Library is a clearing house only; it does not generate or test programs. No warranty, express or implied, is made by the contributor, Digital Equipment Computer Users Society or Digital Equipment Corporation as to the accuracy or functioning of the program or related material, and no responsibility is assumed by these parties in connection therewith.
Diagnose: A Versatile Trace Routine for the PDP 8 Computer

K. B. Oldham
North American Aviation Science Center
Thousand Oaks, California 91360

Introduction

Trace routines are employed to track down logical errors in a program (the "sick" program). Starting at any convenient location in the sick program, instructions are executed, one at a time, and a record of all operations is printed out via the teletype. To avoid the waste of time that would ensue from tracing proven subroutines, an option is provided to omit subroutine tracing.

Trace routines have advantages over the more usual "break trap" debugging routines in being completely systematic in operation, in not requiring an operator's presence at the console, and in providing a complete and permanent record of the sick program's operation. On the other hand, they are purely diagnostic and do not enable the operator to make any corrections or modifications to his program.

Two trace routines (Decus Nos 8-56 and 8-57) for the PDP 8 computer were recently published by Biavati. The present routine is significantly more versatile than those of Biavati, being able to trace sick programs containing floating-point, extended arithmetical, and a variety of input/output instructions. Diagnose is, however, at a disadvantage compared with Biavati's first routine in requiring more memory space (five pages as opposed to two); and compared with his second routine in not possessing
the trace-suppression features of the latter. The mode of operation of Diagnose is quite different from that of the trace routines of Biavati.

**Requirements**

Diagnose is designed for a PDP8 computer equipped with the Extended Arithmetical Element. Five consecutive memory pages are needed (four for solely fixed-point operation). The tapes supplied use pages 17 through 22 (locations 3400 through 4577) but a single redefinition of ORIGIN (source tape I) will reposition the routine to occupy locations ORIGIN-200 through ORIGIN+777.

Diagnose must not, of course, overlap the sick program, or any of its subroutines. A floating-point package is not incorporated into Diagnose and must be provided if floating-point tracing is required.

The floating-point section of Diagnose occupies its first page (locations ORIGIN-200 through ORIGIN-1) and is programmed on source tape V. However, fixed-point operation of Diagnose in no way depends upon the floating-point section. Hence, an object tape assembled from source tapes I through IV will function perfectly for fixed-point operation and will use only four memory pages. Alternatively, if a full (five page) object tape is used for fixed-point tracing the first page may be overwritten with impunity.

**Operation**

Load Diagnose into memory (either before or after the loading of the sick program) by means of the binary loader.
Set the switch register to ORIGIN or ORIGIN+1 (3600 or 3601 for the tape supplied) and press "LOAD ADD" followed by "START." The ASR-33 teletype will now print

TYPE THE INITIAL C(L):

and the operator should respond by typing either 0000 or 0001 according as he wishes the trace operation to commence with the link cleared or set. The teletype will next ask for the initial accumulator contents, and then for the initial contents of the multiplier quotient. Each of these requests should be answered by typing the desired four octal digits, producing a record such as

TYPE THE INITIAL C(L): 0001
TYPE THE INITIAL C(AC): 1234
TYPE THE INITIAL C(MQ): 4567

At this stage the teletype will ask for the address in the sick program at which tracing is to begin, i.e., the initial contents of the program counter, by printing

TYPE THE INITIAL C(PC):

On receiving four octits (octal digits) in response to this request, the tracing procedure will commence without further user intervention, producing output formats as explained in the following section.

Most commonly the user will wish to have the link, accumulator, and multiplier quotient registers cleared at the start of the trace routine. To save the trouble of typing 0000 three times to achieve this clearance, the Diagnose routine should be started at ORIGIN+2 or ORIGIN+3. The teletype will then make the sole request

TYPE THE INITIAL C(PC):
Note that all these responses must be in the form of four octits; leading zeros cannot be omitted. Note also that no printout will be provided by the teletype until all four octits have been inputted.

The position of switch register bit 11 indicates whether or not subroutines are to be traced. If the bit is set (switch up) no tracing will be carried out. Either position of switch 11 may be selected initially and the switch may be repositioned at any time during the operation of Diagnose.

Tracing will continue either until the "stop" key is depressed or until a HLT instruction is encountered in the sick program.

**Output Format**

This format varies somewhat according to the type of instruction being executed, as follows

**Operate Instructions**

The output format when an operate instruction (group 1 micro-instruction, group 2 microinstruction or combined microinstruction) is encountered is

\[
\begin{align*}
& l \text{ accu} \quad m\text{quo} \\
& L \text{ ACCU} \quad M\text{quo}
\end{align*}
\]

where \( l \), \text{accu}, \text{mquo}, \( L \), \text{ACCU}, and \text{Mquo} are the contents of the link, accumulator, and multiplier quotient before and after the execution of the instruction, \text{PROG} is the contents of the program counter and \text{INST} is the instruction code. For example, if the RAR instruction was encountered at location 207 the output might be
EAE Instructions, First Group

This group, comprising the instructions NMI, MQL, SCA, MQA, CLA, and CAM, is handled in exactly the same way as operate instructions. For example, the MQL instruction could produce the printout

\[ \emptyset \quad 4112 \quad 4567 \quad \emptyset21\emptyset \quad 7421 \]

EAE Instructions, Second Group

The remaining EAE instructions, MUY, DVI, SHL, ASR, and LSR, require two memory locations to specify their operation and so need different treatment. The output format is best illustrated by an example. If the instructions MUY and 2 are contained respectively in locations 211 and 212, the following printout could result

\[ \emptyset \quad \emptyset\emptyset\emptyset \quad 4112 \quad \emptyset211,12 \quad 74\emptyset5,\emptyset\emptyset2 \]

Here we have an exception to the general rule that instructions are executed one at a time; in this case the two instructions 74\emptyset5 and \emptyset\emptyset2 are treated together as a unit and the program counter will next contain \emptyset213.

AND, TAD, DCA, ISZ, and JMP Instructions

The output format in these cases is
l accu  mquo
PROG INST=MNE ADDR PREV>POST
L ACCU  MQUO

where MNE is the instruction mnemonic (AND, TAD, DCA, ISZ, or JMP).
ADDR is the effective memory reference address, i.e., in the case of
direct addressing it is the location (on the current page or page zero)
specified in INST, whereas when indirect addressing is used it is the
contents of the specified location. The contents of ADDR before and
after the instruction is executed are represented by PREV and POST.

For example, a sequence of memory reference instructions could produce
.he printout:

ø øøø1 ø224 ø213 53øøJMP ø3øø 1277 >1277
ø øøø1 ø224 ø3øø 1277=TAD ø277 7776 >7776
ø 7777 ø224 ø3ø1 3ø66=DCA øøøø øøøø >7777
ø øøøø ø224 ø3ø2 2ø66=ISZ øøøø 7777 >øøøø
ø øøøø ø224 ø3ø4 1277=TAD ø277 7776 >7776
ø 7776 ø224 ø3ø5 1676=TAD 5øøø 3451 >3451
1 ø224 ø3ø6 ø276=AND ø276 54øø >54øø
1 14øø ø224

JMS Instructions

If bit 11 of the switch register is clear, JMS instructions are
handled exactly the same was as are the other memory reference instruc-
tions, leading to a printout such as

1 14øø ø224 ø3ø7 42øøJMS ø24ø øøøø >ø31ø
1 14øø ø224 ø241 ...
If, however, switch 11 is in the up position when a JMS instruction is encountered, the subroutine will be executed but not traced. The printout differs in having two linefeeds following the "JMS." The previous example would thus produce the following printout

```
1 1400  0224
   0307  4240=JMS
   
0240  0000>0310

L ACCU MQUO
   0310
```

where L, ACCU, and MQUO are the register contents when the subroutine is exited. The asterisk shown above is not printed; it indicates the point at which any output or input called for by the subroutine occurs.

---

**Input/Output Instructions**

Diagnose is designed to handle IOT instructions for the following equipment: ASR33 teletype keyboard/reader/printer/punch, 750C high-speed reader, and the 75E high-speed punch. When the routine encounters such an instruction, it and the next two or three instructions are treated as a unit and executed in a single operation of the Diagnose routine. This procedure is necessary to ensure that the flags are correctly set. The instruction unit has four members in the case of the 750C reader, three members for the other equipment. Thus on encountering a TSF, JMP -1, TLS sequence at locations 234 through 236 with 0252 as accumulator contents, the printout would be

```
l 0252  mquo
   
l 0252  mquo
   0234,36  6041,5234,6046
```

* is the character printed by the sequence. The output of a non-printing
character (line-feed, carriage-return or space) can be detected on the printout as a displacement (downwards, to the left, or to the right) of the next two lines of print. A printed character also causes a rightwards displacement, as in the above example.

**JMS I 7 Instruction**

The JMS I 7 instruction is interpreted by Diagnose as a call for entry to floating-point. It produces the format

```
I accu mquo PROG 447=FENT
±φ.XXXXXXXXXE±XX
```

where FENT is a mnemonic for floating-point entry and ±φ.XXXXXXXXXE±XX denotes the contents of the floating accumulator on entry.

**Floating-point Memory Reference Instructions**

The output format for these instructions is very similar to that for fixed-point memory reference instructions, viz:

```
±φ.XXXXXXXXXE±XX
PROG INST=FMNE ADDR ±φ.yyyyyyyE±yy > ±φ.YYYYYYYE±YY
±φ.XXXXXXXXXE±XX
```

where ±φ.XXXXXXXXXE±XX and ±φ.XXXXXXXXXE±XX represent the contents of the floating accumulator before and after the floating-point operation. FMNE is the floating-point instruction mnemonic (FADD, FSUB, FMPY, FDIV, FGET, FPUT, or FNOR) and ADDR is the effective memory reference address. The contents of ADDR before and after execution of the floating-point instruction are ±φ.yyyyyyyE±yy and ±φ.YYYYYYYE±YY.
Expansible Floating-point Instructions

These instructions are the commands 0001 through 0017 which the floating-point interpreter decodes according to the contents of the "look up" table. The output of Diagnose on encountering such an instruction is illustrated by the example of the square root operation:

\[
\begin{align*}
+0.12100000E+09 & \quad 2311 \quad 0002-F02 \\
+0.10999999E+05 & \quad 2312 \quad \ldots
\end{align*}
\]

FEXT Instruction

This floating-point exit instruction produces an output format similar to that for an expandable instruction. The FEXT mnemonic is outprinted. Fixed-point operation is then resumed until the next encounter of a JMS 17 instruction.

Restrictions

1. Diagnose cannot commence operation within a floating-point sequence.

2. Indirectly addressed autoindices are incorrectly traced.

3. Input/output sequences other than those for which Diagnose was specifically designed may encounter flagging difficulty.

4. A traced subroutine must be exited normally (i.e., return to the instruction following the JMS instruction) or suffer not more than three advances (i.e., return to up to the fourth instruction following JMS).

5. An untraced subroutine must be exited normally.

6. ORIGIN must be the first line of a memory page.