PDP-8/E INSTRUCTION SIMULATORS FOR OTHER PDP-8S

Guy L. Steele, Jr.

Brighton, Massachusetts

August 25, 1971

PAL III

Although this program has been tested by the contributor, 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 program material, and no responsibility is assumed by these parties in connection therewith.
PDP-8/E Instruction Simulators for Other PDP-8's

Guy L. Steele Jr.
110 Washington St.
Brighton, Mass. 02135

Introduction

It used to be that PDP-8's came in four flavors: straight 8, 8/S, 8/I, and 8/L. Now there is a most marvellous fifth flavor: 8/E. The 8/E unlike the other four 8's has some extra instructions in the standard (non-EAE) set. This makes for easier programming, but messes up compatibility. Thus, although other 8 programs will run on an 8/E, 8/E programs may not run on other 8's.

There are three solutions to this problem:
1) don't use 8/E programs on your eight.
2) get rid of your other 8 and get an 8/E.
3) use some nice simulation subroutines like these.

The subroutines listed herein provide simulation for the functions of the PDP-8/E's BSW (byte swap) and standard MQ (non-EAE) microinstructions.

Advantages of Simulation Subroutines
* They work.
* They are called by a JMS I through a page zero pointer; thus these subroutines are call able from anywhere in 4k of core.
* The subroutine calls are exactly as long as the microinstructions they replace: one location.
* They are virtually independent of one another; thus they may be tucked into any page where there are a few words to spare, scattered all over core.
* They're efficient (for simulation subroutines.)
* They may be called by the same mnemonics as the actual instructions; no extensive program revision is necessary.

Disadvantages of Simulation Subroutines
* They take up a little extra core.
* They tie up page zero locations.
* There is no provision for the use of multiple memory fields.
* They're slower than the real thing. (If you're that anxious to save execution time, see option 2) above and ignore this stuff.)

Methods

I was inspired to write these routines by Robert H. Nagel (Ref. 1), who wrote a BSW simulator. My BSW simulator is an improved version of his original subroutine. The other microinstructions are simulated by maintaining a pseudo-MQ register on page zero. All the MQ operations are performed with respect to this pseudo-MQ.

Each simulator is invoked by using the standard microinstruction mnemonics BSW, MQA, MQL, CM, ACL, SWP, and one I invented called ALC which is equivalent to CLA SWP which in turn equals CLA MQA MQL. These mnemonics are redefined to be JMS's to the appropriate simulator subroutines. Thus, assembling the subroutines with the user program causes the links to the simulators to be assembled into the program.
One location is required on page zero for the pseudo-MQ, plus one location for each subroutine to serve as a pointer for the JMS's.

The algorithms used for the simulations are relatively straightforward and are explained in comments in the listing. All algorithms are original with myself except for the exclusive OR algorithm (Ref. 2)

Restrictions

If the microinstructions are encountered in the forms MQA MQL, QA MQL, etc., instead of SWP, CAM, etc., the instructions will not execute properly. This applies especially to QA SWP which has no single standard mnemonic. I have provided one: AQC. Any coding containing such combinations should be rewritten in terms of single mnemonics.

No provision is made for the possible desire to microprogram BSW with CLL, QML, STL, or IAC. The user may either write an extra simulator or simply use two separate instructions.

References


/ PDQ-8/E MICROINSTRUCTION SIMULATION ROUTINES
/
/ EACH SUBROUTINE IS INDEPENDENT OF THE OTHERS;
/ HOWEVER, NOTE THAT TO CONSERVE CORE LOCATIONS
/ XXCAM AND XXACL ARE USED AS TEMPORARY STORAGE
/ BY OTHER SUBROUTINES. IF XXCM OR XXACL SHOULD
/ BE REMOVED A NEW LOCATION SHOULD BE DEFINED IF
/ NECESSARY FOR XXTMP1 OR XXTMP2.
/
/ NOTE THAT IF A SUBROUTINE IS REMOVED ITS PAGE
/ ZERO POINTER MAY ALSO BE REMOVED.
/
/ NOTE THAT THE MQL INSTRUCTION IS SIMULATED
/ SIMPLY BY A DCA RATHER THAN A JMS.
/
/////////////////////////////////////////////////////////////////////////////////////////////////////////
/ PAGE ZERO LOCATIONS
*170 / ANY PAGE ZERO LOCATIONS ARE OK
MQL=DCA . / MQL (LOAD MQ FROM AC, CLEAR AC)
XXCM, & / PSEUDO-MQ REGISTER
CAM=JMS I . / CAM (CLEAR AC AND MQ)
XXCAM
ACL=JMS I . / ACL (LOAD AC FROM MQ)
XXACL
SWP=JMS I . / SWP (SWAP AC AND MQ)
XXSWP
MQA=JMS I . / MQA (OR MQ INTO AC)
XXMQA
ALC=JMS I . / ALC (STANDS FOR AC LOAD
XXALC / AND CLEAR. THIS IS NOT A
/ STANDARD MNEMONIC AND SHOULD
XXTMP1=XXACL / BE DEFINED AS ALC=CLA SWP
XXTMP2=XXCAM / FOR A PDQ-8/E LOAD AC FROM
/ MQ, CLEAR MQ)
/
/ BSW=JMS I . / BSW (BYTE SWAP, SWAP BITS 0-5
XXBSW / OF AC WITH BITS 6-11)
/
/ THE SUBROUTINES THEMSELVES MAY BE LOCATED ON ANY PAGE;
/ IF XXTMP1-XXACL AND XXTMP2-XXCAM THEN THE SUBROUTINES
/ SHOULD BE ARRANGED SO THAT THESE LOCATIONS CAN BE
/ ACCESSED, BUT OTHER THAN THIS THEY MAY BE TUCKED INTO
/ WHATEVER FREE AREAS EXIST IN CORE.
*600 / CAM (CLEAR AC AND MQ)
/ ALGORITHM: QUITE SIMPLE AND STRAIGHTFORWARD
/ LENGTH: 4 LOCATIONS
/ EXECUTION TIME: 8 CYCLES
XXCAM, &
CLA / CLEAR AC
DCA XXMQ / CLEAR MQ
JMP I XXCAM
ACL (LOAD AC FROM MQ)
ALGORITHM: ABOUT AS STRAIGHTFORWARD AS FOR CAM
LENGTH: 4 LOCATIONS
EXECUTION TIME: 8 CYCLES
XXACL, ∅
CLA / CLEAR AC
TAD XMQ / GET MQ
JMP I XXACL

SWP (SWAP AC AND MQ)
ALGORITHM: MOVE AC TO T1; MOVE MQ TO T2; MOVE T1 TO MQ; MOVE T2 TO AC
LENGTH: 8 LOCATIONS
EXECUTION TIME: 17 CYCLES
XXSWP, ∅
DQA XXTMP1 / MOVE AC TO T1
TAD XXMQ
DQA XXTMP2 / MOVE MQ TO T2
TAD XXTMP1
DQA XXMQ / MOVE T1 TO MQ
TAD XXTMP2 / MOVE T2 TO AC
JMP I XXSWP

MQA (INCLUSIVE OR MQ INTO AC)
ALGORITHM: (AC IOR MQ) EQUALS (((NOT AC) AND MQ) + AC)
WHERE IOR, NOT, AND ARE LOGICAL OPERATORS (SEE 1970 DEC SMALL COMPUTER HANDBOOK, PAGE 28.)
LENGTH: 7 LOCATIONS
EXECUTION TIME: 14 CYCLES
XXMQA, ∅
DQA XXTMP1 / AC
TAD XXTMP1
CMA / (NOT AC)
AND XXMQ / (((NOT AC) AND MQ)
TAD XXTMP1 / (((NOT AC) AND MQ) + AC)
JMP I XXMQA

ALC (AC LOAD AND CLEAR; LOAD AC FROM MQ, CLEAR MQ)
ALGORITHM: MOVE MQ TO T1, CLEAR MQ, MOVE T1 TO AC
(This subroutine could be shortened by using XXSWP at a cost of execution time increase.)
LENGTH: 7 LOCATIONS
EXECUTION TIME: 14 CYCLES
XXALC, ∅
CLA
TAD XXMQ
DQA XXTMP1 / MOVE MQ TO T1
DCA XXMQ / CLEAR MQ
TAD XXTMP1 / MOVE T1 TO AC
JMP I XXALC
BSW (BYTE SWAP: SWAP BITS 0-5 OF AC WITH BITS 6-11)
ALGORITHM: QUITE INGENIOUS (I'M PROUD OF THIS ONE.)
THIS ONE IS HARD TO DESCRIBE VERBALY. THE
COMMENTS ON THE SUBROUTINE ILLUSTRATE WHERE
THE CONTENTS OF THE ORIGINAL AC AND LINK
APPEAR IN THE AC AND LINK AT EACH STEP.
LENGTH: 13 LOCATIONS
EXECUTION TIME: 21 CYCLES
NOTE THAT THE CONSTANT IS OFTEN USEFUL FOR OTHER
PARTS OF A USER PROGRAM.

<table>
<thead>
<tr>
<th>LINK</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>L XXX XXX YYY YYY</td>
<td></td>
</tr>
<tr>
<td>DCA XXTMP1</td>
<td></td>
</tr>
<tr>
<td>RTR</td>
<td></td>
</tr>
<tr>
<td>DCA XXTMP2</td>
<td></td>
</tr>
<tr>
<td>TAD XXTMP1</td>
<td></td>
</tr>
<tr>
<td>AND XXTMP</td>
<td></td>
</tr>
<tr>
<td>TAD XXTMP1</td>
<td></td>
</tr>
<tr>
<td>RTL</td>
<td></td>
</tr>
<tr>
<td>RTL</td>
<td></td>
</tr>
<tr>
<td>TAD XXTMP2</td>
<td></td>
</tr>
<tr>
<td>REL</td>
<td></td>
</tr>
<tr>
<td>JMP I XBSW</td>
<td></td>
</tr>
</tbody>
</table>

XX78, 2780 

<table>
<thead>
<tr>
<th>LINK</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>L XXX XXX YYY YYY</td>
<td></td>
</tr>
</tbody>
</table>

ALL DONE AT LAST!

XX78, 2780 

<table>
<thead>
<tr>
<th>LINK</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>L XXX XXX YYY YYY</td>
<td></td>
</tr>
</tbody>
</table>

NIPTY CONSTANT

THIS CONCLUDES THIS SET OF WONDERFUL PDP-8/E
INSTRUCTION SIMULATORS. I HOPE YOU HAVE
ENJOYED THEM.        - GUY L. STEELE JR.