F-64PX
8/64

PROGRAMMING EXAMPLES

PDP-6

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

F64-PX
8/64

PDP-6 PROGRAMMING EXAMPLES

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

INTRODUCTION

This manual contains examples of programming for the PDP-6 Type 166 Processor. They have been chosen to illustrate both the arithmetic and logical capabilities of the processor. For an explanation of the instructions shown see the PDP-6 Handbook (F-65). The examples use the same instruction mnemonics as the MACRO-6 assembler. The language is described in the MACRO-6 Manual (F-64MAS).

Times based on design estimates are shown in some of the examples. All of the instruction times have been conservatively calculated. For example, no attempt has been made to take advantage of speed gains due to memory overlapping. Two of the examples show how time may be saved by moving to fast memory a short program which executes a large number of iterations. One of these, Character Manipulation, is programmed in both a straightforward manner and by being moved to fast memory in order to show the break-even point between the time gained and the increased overhead time. The second, Two-bit Testing, was programmed for 500 iterations. In this case there is a considerable gain in time by moving the program to fast memory. The last example, Any Radix Print, demonstrates the use of recursion to shorten programs.

Sixteen examples are contained in this booklet:

1. Single Precision Integer Arithmetic
2. Double Precision Integer Arithmetic
3. Floating Point Arithmetic
4. Fix a Floating Point Number
5. Float a Fixed Point Integer
6. Repetitive Calculation
7. Subscripts
8. Exponentiation
9. Character Manipulation
10. Character Translation
11. Character Addition
12. Fifteenth Degree Polynomial
13. Evaluation of Complex Polynomial

14. Matrix Inversion
15. Two-Bit Testing and Depositing of Data
16. Any Radix Print

SINGLE PRECISION INTEGER ARITHMETIC

Assume: 1) A, B, C, D, E, F, G, H, J, K, L, M, N, and P are arbitrary memory locations.

2) Arguments and instructions are in the same memory module.

3) No scaling is required.

4) Y indicates 1 of the 16 accumulators

			Time in microseconds
	A=B+C		
MOVE Y, B			4
ADD Y, C			4
MOVEM Y, A			4
		Total	12
	D=E+F+G		
MOVE Y, E			4
ADD Y, F			4
ADD Y, G			4
MOVEM Y, D			4
		Total	16
	H=JxK		
MOVE Y, J			4
IMUL Y, K			13.6
MOVEM Y, H			4
		Total	21.6
	L=MxN+P		
MOVE Y, M			4
IMUL Y, N			13.6
ADD Y, P			4
MOVEM Y, L			4
		Total	25.6

DOUBLE PRECISION INTEGER ARITHMETIC

Assume:

1) A, B, C, D, E, F, G, H, J, K, L, M, N, and P are arbitrary memory locations, each denoting a block of two consecutive memory registers.

2) Each integer is stored in two consecutive memory locations with the high order integer in the first location and the low order integer in the second.

3) Instructions and arguments are in same memory module.

```
                                                            Time in microseconds

                          A=B+C

    JFCL 16, . + 1,             CLEAR STRAY FLAGS                    2.1
    MOVE 0, B                                                        4.0
    MOVE 1, B + 1                                                    4.0
    ADD 1, C+1,                 ADD LOW ORDER PARTS                  4.0
    JFCL 2, D1,                 DID LOW ORDER PARTS OFLO             2.1
D2: ADD 0, C,                   ADD HIGH ORDER PARTS                 4.0
    MOVEM 0, A,                 STORE RESULTS                        4.0
    MOVEM 1, A+1                                                     4.0
      .
      .
      .
D1: AOJA 0, D2                  COMPENSATE FOR OVERFLOW              3.3
                                                           Total 28.2-31.5

                          D=E+F+G

    JFCL 16, . + 1,             CLEAR STRAY FLAGS                    2.1
    MOVE 0, E                                                        4.0
    MOVE 1, E + 1                                                    4.0
    ADD 1, F + 1,               ADD LOW ORDER PARTS                  4.0
    JFCL 2, DD1,                DID LOW ORDER PARTS OFLO             2.1
DD1A: ADD 1, G + 1,             ADD LOW ORDER PARTS                  4.0
    JFCL 2, DD2,                OVERFLOW?                            2.1
DD2A: ADD 0, F                  ADD HIGH ORDER PARTS                 4.0
    ADD 0, G                                                         4.0
    MOVEM 0, D,                 STORE ANSWERS                        4.0
    MOVEM 1, D+1                                                     4.0
      .
      .
      .
DD1: AOJA 0, DD1A,              COMPENSATE FOR OVERFLOW              3.3
DD2: AOJA 0, DD2A,              COMPENSATE FOR OVERFLOW              3.3
                                                           Total 38.3-44.5
```

DOUBLE PRECISION INTEGER ARITHMETIC (continued)

```
                                                           Time in microseconds

                      K=JxK

      MOVE 0, J                                                  4.0
      MUL 0, K,             MULTIPLY HIGH ORDER PARTS           14.0
      MOVE 2, J + 1                                              4.0
      MUL 2, K,             MULTIPLY LOW (J); HIGH (K)          14.0
      JFCL 16, . + 1,       CLEAR STRAY FLAGS                    2.1
      ADD 1, 2,             SUM PRODUCTS                         4.0
      JFCL 2, M1            OVERFLOW?                            2.1
  M1A: MOVE 2, J,           MULTIPLY HIGH (J), LOW (K)           4.0
      MUL 2, K + 1                                              14.0
      ADD 1, 2,             SUM PRODUCTS                         4.0
      JFCL 2, M2            OVERFLOW?                            2.1
  M2A: MOVEM 1, H + 1,      STORE RESULTS                        4.0
      MOVEM, H                                                   4.0
        .
        .
        .
  M1: AOJA 0, M1A           COMPENSATE OVERFLOW                  3.3
  M2: AOJA 0, M2A           COMPENSATE OVERFLOW                  3.3
                                                       Total 76.3-82.9
```

FLOATING POINT ARITHMETIC

Assume: 1) A, B, C, D, E, F, G, H, J, K, L, M, N, and P are arbitrary memory locations.

2) Arguments and instructions are in the same memory module.

3) Y indicates 1 of the 16 accumulators.

A=B+C

```
MOVE Y, B                 4.0
FAD Y, C                  5.8
MOVEM Y, A                4.0
                   Total 13.8
```

D=E+F+G

```
MOVE Y, E                 4.0
FAD Y, F                  5.8
FAD Y, G                  5.8
MOVEM Y, D                4.0
                   Total 19.6
```

H=JxK

```
MOVE Y, J                 4.0
FMP Y, K                 12.6
MOVEM Y, H                4.0
                   Total 20.6
```

L=MxN+P

```
MOVE Y, M                 4.0
FMP Y, N                 12.6
FAD Y, P                  5.8
MOVEM Y, L                4.0
                   Total 26.4
```

PROBLEM: Consider an eight block table with 100 entries in each block. Let A, B, C, D, E, F, G, and H denote the first location of each block. For each entry find:

$$G = (A-B)^2 + (C-D)^2$$

$$H = (G/E) \times F$$

Assume: 1) All entries in normalized floating point.

2) Argument and instructions are in the same memory module.

FLOATING POINT ARITHMETIC (continued)

```
Program                                                 Time in microseconds

BEGIN:  MOVSI 1,↑D100                                           2
        MOVE 2, A(1)                                            4
        FSB 2, B(1)                                             6.2
        FMP 2, 2                                               12.3
        MOVE 3, C(1)                                            4
        FSB 3, D(1)                                             6.2
        FMP 3, 3                                               12.3
        FAD 2, 3                                                5.5
        MOVEM 2, G(1)                                           4
        FDV 2, E(1)                                            18.0
        FMP 2, F(1)                                            12.8
        MOVEM 2, H(1)                                           4.0
        AOBJN 1, BEGIN+1                                        3.3
                                                               ----
```

The total time for 100 repetitions:
92.6 x 100 + 2 = 9.26 milliseconds

FIX A FLOATING POINT NUMBER

Assume: 1) A floating point number in any accumulator from 1-15 designated by F. The result is returned in F+1 module 16.

```
MULI F, 400,         EXPONENT IN F, FRACTION IN F+1
TSC F, F,            COMPLEMENT EXPONENT IF NEGATIVE
ASH F+1, -243(F),    TRUNCATE TO GREATEST INTEGER
```

FLOAT A FIXED POINT NUMBER

Assume: 1) A fixed point integer less than 2^{27} in magnitude in accumulator C.

```
TLC C,233000,   XOR INTO WORD
FAD C, 0,       FLOATING ADD ZERO TO NORMALIZE
```

2) A fixed point integer I, $-2^{35} \leq I < 2^{35}$, in accumulator F.

Note: Accumulator F+1 is used in the calculation.

```
IDIVI    F,400,          DIVIDE WORD INTO TWO PIECES
SKIPE    F;              SKIP IF NORMALIZED ZERO
TLC      F,243000,       XOR EXPONENT INTO F
TLC      F+1,233000,     XOR EXPONENT INTO F+1
FAD      F, F+1,         COMBINE AND NORMALIZE
```

REPETITIVE CALCULATION

The following are repeated 10000 times:
A=B+C, D=E+F+G, H=JxK

Assume: 1) A, B, C, D, E, F, G, H, J, K, L, M, N, and P are arbitrary memory locations.

2) Arguments are in floating point.

3) Arguments and instructions are in the same memory module.

```
                                                                  Time in microseconds

       MOVEI 2,↑D10000,           INITIALIZE COUNTER                       2
B1:    MOVE 0, B                                                           4
       FAD 0, C                                                            5.2
       MOVEM 0, A                                                          4
       MOVE 0, E                                                           4
       FAD 0, F                                                            5.2
       FAD 0, G                                                            5.2
       MOVEM 0, D                                                          4
       MOVE 0, J                                                           4
       FMP 0, K                                                           12.6
       MOVEM 0, H                                                          4
       SOJN 2, B1,                COUNT                                    3
                                                                         ____
                                                                          55.2
```

The total time for 10000
repetitions:
55.2 x 10000 + 2 = 0.552 sec.

SUBSCRIPTS

```
Compute for I=1, 100
A(I)=B(I)+C(I)
D(I)=E(I)+F(I)+G(I)
H(I)=A(I)xD(I)
```

Assume: 1) The data is arranged in memory as follows:

B1, B2, ---B100, C1, C2, ---C100, E1, F1, G1,
E2, F2, G2 ---E100, F100, G100, A1, D1, A2,
D2 ---A100, D100, H1, H2, ---H100

```
                                                                  Time in microseconds
                                                                  ____________________

       CLEARB 3, 2,            INITIALIZE INDEX REGISTERS                4.0
       HRLZI 1, -↑D100,        INITIALIZE INDEX COUNTER                  2.0
C1:    MOVE 4, E(3)                                                      4.0
       FAD 4, F(3)                                                       6.0
       FAD 4, G(3)                                                       6.0
       MOVEM 4, D(2),          D=E+F+G                                   4.2
       MOVE 0, B(1)                                                      4.0
       FAD 0, C(1)                                                       6.0
       MOVEM 0, A(2)                                                     4.2
       FMP 0, 4                                                         12.3
       MOVEM 0, H(1)                                                     4.2
       ADDI 2, 2,              INCREMENT 2 STEP INDEX                    3.0
       ADDI 3, 3,              INCREMENT 3 STEP INDEX                    3.0
       AOBJN 1, C1,            INCREMENT 1 STEP INDEX
                               AND COUNT                                 3.3
                                                                        ____

                                  Total Time required is 0.062 seconds
```

EXPONENTIATION

```
,FLOATING POINT NUMBER TO A FIXED POINT POWER
,COMPUTE X↑I  USING ACCUMULATORS A1, A2, AND T.
,STORE THE RESULT IN Y.  IF X IS ZERO, RETURN ZERO

FEXP:   MOVE A1, X              ;MOVE X TO A1
        MOVSI T, 201400         ;T1 = 1.0
        SKIPGE A2, I            ;MOVE I TO A2, SKIP IF NON-NEGATIVE
        FDVM T, A1              ;TAKE RECIPROCAL OF X (NEGATIVE POWER)
        MOVMS A2                ;TAKE ABSOLUTE VALUE OF I
        JUMPN A1, FEXP2         ;GO TO MAIN LOOP (IF NON ZERO BASE)
        CLEARB T, A2            ;ZERO EXPONENT AND RESULT FOR QUICK EXIT
FEXP1:  FMP A1, A1              ;SQUARE BASE TERM
        LSH A2, -1              ;SHIFT RIGHT FOR NEXT BIT OF EXPONENT
FEXP2:  TRZE A2, 1              ;IS POWER A FACTOR? TURN OFF BIT
        FMP T, A1               ;YES
        JUMPN A2, FEXP1         ;MORE FACTORS?
        MOVEM T, Y              ;NO, STORE RESULT
```

CHARACTER MANIPULATION

PROBLEM: There is a string of 7 bit ASCII characters beginning at memory location A and ending with a slash. Transfer the characters, excluding the slash, to a block beginning at location B. Count the number of characters and leave the result in an index register.

Assume: 1) The code for a slash is 74_8.

```
Program                                                     Time in microseconds

        MOVE 3, [POINT 7, B]                                         4
        MOVE 2, [POINT 7, A]                                         4
        MOVEI 1, 0                                                   2.0
C:      LDBI 0, 2                                                    5
        CAIN 0, "/"                                                  2.6
        JRST  EXIT                                                   2.1
        DPBI 0, 3                                                    5
        AOJA 1, C                                                    3.4

                                       Total time is 18 + 16 x N where
                                       N is the number of characters.

        MOVSI A1, CMOVP                                              2.0
        BLT A1, A1-1,                 MOVE TO FAST MEMORY           17.6
        JRST CMOV                                                    3.0
CMOVP: PHASE 0
B1      0
PTA:    POINT 7,A
PTB:    POINT 7,B
CMOV: LDBI A1, PTB                                                   5.0
        CAIN A1, 74                                                  2.0
        JRST  EXIT                                                   2.0
        DPBI A1, PTB                                                 5.0
CM:     AOJA B1, CMOV                                                2.0
        DEPHASE
A1 = CM + 1
```

The time for this case is 31 + 14 x N.

CHARACTER TRANSLATION

Assume: 1) That the number in accumulator A is a 6-bit code read from the card reader. The program must translate the card code into the equivalent 7-bit ASCII code. A translation table begins at location TAB consisting of 7-bit ASCII characters packed five to a word.

2) The characters in this table are in order of their appearance in the card code. Because characters are packed five to a word, the quotient of the card code divided by 5 gives the word in which the ASCII character is found. The remainder gives the character position. An auxilliary table of five byte pointers, one pointing to each character position, allows retrieval of the proper ASCII with a single LDB instruction.

```
TRANSL: IDIVI   A,5
        LDB     A, BTAB (A+1)
        JRST    EXIT

BTAB:   POINT   7, TAB (A), 6
        POINT   7, TAB (A),13
        POINT   7, TAB (A),20
        POINT   7, TAB (A),27
        POINT   7, TAB (A),34

TAB:    ASCII   .←1234.
        ASCII   .56789.
        ASCII   .Ø=@↑'.
        ASCII   .\ /ST.
        ASCII   .UVWXY.
        ASCII   .Z;,(".
        ASCII   .#%-JK.
        ASCII   .LMNOP.
        ASCII   .QR:$*.
        ASCII   .[>&+A.
        ASCII   .BCDEF.
        ASCII   /GHI?./
        ASCII   .) ]<!.
```

CHARACTER ADDITION

PROBLEM: Add two 5 digit numbers expressed at 7-bit ASCII characters.

Calling Sequence: JSP AC3, ASCIAD

```
,ASCIAD:    A ROUTINE TO ADD OR SUBTRACT FIVE DIGIT ASCII NUMBERS (7 BIT
,           CHARACTERS).
,
,           1. CHARACTERS MUST BE RIGHT JUSTIFIED
,           2. TO ENTER ROUTINE:
,              A. MOVE AC0, (ADDEND)
,              B. MOVE AC1, (AUGEND)/ MOVN AC1, (SUBTRAHEND)
,              C. JSP AC3, ASCIAD
,           3. ON RETURN THE SUM OR DIFFERENCE IS IN AC2 AND THE CONTENTS
,              OF AC0 AND AC1 ARE UNCHANGED
,           4. THE ROUTINE IS A RING COUNTER; FOR EXAMPLE
,              99999+2=00001 and 3-6=99997
,           5. NOTE THAT TWO NEGATIVE NUMBERS CANNOT BE COMBINED AND
,              THAT IF ONE IS NEGATIVE IT MUST APPEAR IN AC1 ON ENTRY.
,
,
,
ASCIAD:     AND     AC0,M2
            IOR     AC1,M4
            TLZN    AC1,400000
            ADD     AC1,M1
            ADD     AC0,AC1
            AND     AC0,M3
            MOVE    AC2, M4
            AND     AC2,AC0
            ASH     AC2,-3
            SUBM    AC0,AC2
            IOR     AC2,M4
            JRST    (AC3)

M1:         BYTE (1) (7) 6, 6, 6, 6, 6
M2:         BYTE (1) (7) 17, 17, 17, 17, 17
M3:         BYTE (1) (7) 77, 77, 77, 77, 77
M4:         BYTE (1) (7) 60, 60, 60, 60, 60
            AC0=0
            AC1=1
            AC2=2
            AC3=3
            END
```

FIFTEENTH DEGREE POLYNOMIAL

Assume: 1) P denotes a block of memory containing the 16 coefficients; $\underline{\overline{X}}$ is a memory location containing the argument; the answer is stored in location Z.

```
                                                      Time in microseconds

RADIX 10,             SET ASSEMBLER RADIX TO 10
MOVE 3, X,            MOVE ARG TO FAST MEMORY              4.0
MOVEI 2, 15           INITIALIZE INDEX COUNTER             2.0
MOVE 0, P + 15,       INITIALIZE VALUE                     4.0
IMUL 0, 3,            MULTIPLY BY ARGUMENT                13.2
ADD 0, P-1 (2)        ADD NEXT LOWER COEFFICIENT           4.0
SOJGE 2, .-2,         INCREMENT AND COUNT                  2.8
MOVEM 0, Z,           STORE ANSWER                         4.0
                                                          ____
                        Total time required is 314 microseconds
```

EVALUATION OF COMPLEX POLYNOMIAL

$Y = P_o + P_1 X^1 + P_2 X^2 + \ldots\ldots P_n X^n$

WHERE Y, X, and P are complex numbers.

The real parts of the coefficients, P, are stored in an array, the first location labeled P. The imaginary parts are stored in another array, PI. The argument is X (real part) and XI (imaginary part), the answer is placed in Y and YI, and the order is in N.

DATA STRUCTURE

```
P:  BLOCK 14,                REAL COEFFICIENT PARTS
PI: BLOCK 14,                IMAGINARY COEFFICIENT PART
X:  0                        REAL PART OF ARGUMENT
XI:0                         IMAG. PART OF ARGUMENT
Y:  0                        REAL PART OF ANSWER
YI:0                         IMAG. PART OF ANSWER
N:  0                        ORDER OF POLYNOMIAL
```

```
                                                            Time in microseconds

MOVEI 4, N,                  INITIALIZE INDEX COUNTER                  2.0
MOVE 0, P (4),               INITIALIZE ANSWER                         4.0
MOVE 1, PI(4),                                                         4.0
MOVE 2, X,                   MOVE ARGUMENT TO                          4.0
MOVE 3, XI,                  FAST MEMORY                               4.0

P13:  MOVE 5, 1                                                        2.4
      FMP 5, 3,              PI * XI                                  12.2
      MOVE 6, 0                                                        2.4
      FMP 6, 3,              P * XI                                   12.2
      FMP 0, 2,              P * X                                    12.2
      FSB 0, 5,              P * X - PI * XI = REAL PART               5.6
      FMP 1, 2,              PI * X                                   12.2
      FAD 1, 6,              P * XI + PI * X = IMAGINARY PART          5.4

      FAD 0, P-1 (4),        ADD NEXT LOWER COEFFICIENT                6.0
      FAD 1, PI-1 (4)                                                  6.2
      SOJGE 4, P13                                                     4.0

      MOVEM 0, Y,            STORE ANSWER                              4.0
      MOVEM 1, YI                                                      4.0
```

TIME = 28 + 80.8N μsec.

Example: A 13th Degree Polynomial Requires 1.04 milliseconds.

MATRIX INVERSION

PROBLEM: To invert an NxM matrix, stored row-wise in sequential locations beginning with A.

```
,CALLING SEQUENCE:
,CALL:  JSP 17, INVER
,       EXP A
,       JRST ERROR
,THE ORDER OF THE MATRIX IS IN A, WITH THE NUMBER OF ROWS IN THE LEFT HALF,
,AND THE NUMBER OF COLUMNS IN THE RIGHT HALF.  THE ELEMENTS ARE STORED
,ROW-WISE BEGINNING IN A+1
,
,IF THE INVERSION WAS SUCCESSFUL IT WILL RETURN TO CALL +3, AND IF A ZERO
,PIVOT ELEMENT OR OVERFLOW OCCURRED, IT WILL RETURN TO CALL +2
,
,ACCUMULATOR ASSIGNMENTS

T=15      ,PIVOT ELEMENT
J=14      ,COLUMN SUBSCRIPT
K=13      ,ROW SUBSCRIPT
P=12      ,INDEX POINTING TO PIVOT ELEMENT
PT=11     ,MULTIPLIER
LC=10     ,STOP COUNTER
LCS=7     ,ROW COUNTER

INVERT:  HRRZ  @ (17)
         MOVEM, ROWS#                 ;GET COLUMN COUNT
         HLRZ  @ (17)
         MOVEM, COLS#                 ;GET ROW COUNT
         MOVE [XWD ROWPRG, ROW]       ;MOVE ROW SUBROUTINE
         BLT ROWL;                    INTO FAST MEMORY

         HRR ROW, (17)
         ADDI ROW,1                   ;SET UP PROGRAM ADDRESSES
         HRR ROW+2, (17)
         HRR ROW+3, ROW
         ADDI ROW+2, 2
         HRRM ROW, INZ1+1
         HRRM ROW, DIV+2
         HRRM ROW, DIV+6
         HRRM ROW, INZROW+1
         HRRM ROW+2, DIV

         MOVEI P, 0
         MOVN T, COLS
         MOVE T, 1 (T)
         HRRM T, INZ1
```

```
INZSTP:    MOVEI K, 0               ;INITIALIZE INVERSION STEP
           MOVE LCS, ROWS
           MOVE J, P

INZ1:      HRLI J, 0                ;GET PIVOT ELEMENT
           SKIPN T, A(P)            ;IF IT IS ZERO, EXIT AS ERROR
           JRST 1(17)

DIV:       MOVE A+1 (J)             ;DIVIDE PIVOT ROW THROUGH BY
           FDV T                    ;PIVOT ELEMENT
           MOVEM A(J)
           AOBJN J, DIV
           MOVSI 1.0B53             ;LAST ELEMENT OF PIVOT ROW
           FDV T
           MOVEM A(J)

INZROW:    MOVE J, P                ;INITIALIZE TO PROCESS A ROW
           MOVE PT, A(K)
           CAMN K, P                ;IF THE ROW IS THE PIVOT ROW,
           JRST ROWSKP              ;SKIP IT
           JRST ROW                 ;GO TO PROGRAM IN FAST MEMORY
ROWOUT:    MOVN@ ROW                ;HANDLE FINAL ELEMENT OF THE ROW
           FMP PT
           MOVEM@ROW+2

CTX:       SOJN LCS, INZROW         ;IS STEP FINISHED?
           ADD P, COLS
           SOJN LC, INZSTP          ;IS JOB FINISHED?

           JRST 2(17)               ;RETURN

ROWSKP:    ADD K, COLS              ;SKIP PIVOT ROW DURING
           JRST CTX                 ;INVERSION STEP
,THIS PROGRAM FOR PROCESSING THE ELEMENTS IN A ROW IS MOVED INTO FAST MEMORY

ROWPRG:
PHASE 1
ROW:       MOVN A(J)
           FMP PT
           FAD A+1 (K)
           MOVEM  A(K)
           ADDI J, 1
           AOBJN K, ROW
ROWL:      JRST ROWOUT
DEPHASE
END
```

TWO-BIT TESTING AND DEPOSITING OF DATA

PROBLEM: Consider four tables with 500 registers a table. The entries of the first table contain a 2-bit item, ITEM zeros, in bits 13 and 14. The entries of the second table contain ITEM ones in bits 1-6; the third table contains ITEM twos in bits 1-9; and the fourth table contains ITEM threes in bits 1-10.

For n = 1, 500

If ITEM $0_n = 0$ Set: ITEM $1_n = 10_8$

ITEM $2_n = 100_8$

ITEM $3_n = 300_8$

If ITEM $0_n = 1$ Set: ITEM $1_n = 20_8$

ITEM $2_n = 200_8$

ITEM $3_n = 400_8$

If ITEM $0_n = 2$ Set: ITEM $1_n = 30_8$

ITEM $2_n = 300_8$

ITEM $3_n = 500_8$

If ITEM $0_n = 3$ Set: ITEM $1_n = 40_8$

ITEM $2_n = 400_8$

ITEM $3_n = 600_8$

Program: For 500 cases, moving the program to fast memory results in a time saving of approximately 5000 microseconds.

```
      HRLZI 0, A
      BLT 0, 17
      JRST 2

A:    0
      XWD↑D-500,0
      LDB 0, 14
      ROT 0, 3
      ADDI 0, 10
      DPB 0, 15
      ROT 0, 3
```

TWO-BIT TESTING AND DEPOSITING OF DATA (continued)

```
DPB 0, 16
ADDI 0, 200
DPB 0, 17
AOBJN 1, 2
JRST EXIT
POINT 2, TAB0 (1), 14
POINT 6, TAB1 (1), 6
POINT 9, TAB2 (1), 9
POINT 10, TAB3 (1), 10
```

ANY RADIX PRINT

PROBLEM: To Print out a signed number in an arbitrary radix.

Assume: 1) TOUT is the first location of an I/O Routine which exits by POPJ P,0. The argument to tout is in accumulator B.

2) The output radix is stored in the address part of RADIX. The output radix in this example is R.

3) Place the number to be converted in accumulator A and call RADPT with PUSHJ P, RADPT. This routine suppresses leading zeros.

```
RADPT:   JUMPGE B, RADIX          ;IS NUMBER NEGATIVE?
         MOVEI B, "-"             ;YES, GET ASCII MINUS SIGN
         PUSHJ P, TOUT            ;OUTPUT THE MINUS SIGN
         MOVNS A                  ;TAKE ABSOLUTE VALUE OF ARGUMENT
RADIX:   IDIVI A, R               ;QUOTIENT GOES TO A, REMAINDER TO A+1
         HRLM A+1, (P)            ;SAVE REMAINDER IN LEFT SIDE OF LAST
                                  ;ITEM ON PUSH DOWN LIST
         SKIPE A                  ;IS QUOTIENT = 0?
         PUSHJ P, RADIX           ;NO, GO BACK FOR ANOTHER DIGIT
RADPT1:  HLRZ B, (P)              ;GET THE DIGIT OFF THE PUSHDOWN LIST
         ADDI B, 260              ;CONVERT THE DIGIT TO ASCII
         JRST TOUT                ;GO TO THE I/O ROUTINE. TOUT EXECUTES
                                  ;A POPJ P, BACK TO RADPT1 OR (FINALLY)
                                  ;TO THE PLACE WHERE RADPT WAS CALLED.
```