insite T.M.

VOLUME II

INTEL SOFTWARE INDEX

AND TECHNOLOGY EXCHANGE

PROGRAM LIBRARY MANUAL

intel®

SECTION 5

MATH AND NUMERICAL MANIPULATION PROGRAM

SECTION 5

MATH AND NUMERICAL MANIPULATION PROGRAMS

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC1

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

FLOATING POINT MATH PACKAGE

Function

Package contains subroutines for Addition, Subtraction, Multiplication, Division, Negate, Absolute Value and Test of Floating Point numbers.

Required Hardware

None

Required Software

None

Input Parameters

Output Results

Program offered on diskette only.

Registers Modified:	**Assembler/Compiler Used:** MAC8
RAM Required: 63 bytes	**Programmer:** C.E. Ohme
ROM Required: 768 bytes	**Company:**
Maximum Subroutine Nesting Level:	**Address:** 44750 Winding Lane Fremont, Ca. 94538

8008 BINARY FLOATING POINT SYSTEM

THE 8008 BINARY FLOATING POINT SYSTEM CONSISTS OF A SET OF SUBROUTINES DESIGNED TO PERFORM ARITHMETIC OPERATIONS ON NUMERIC QUANTITIES REPRESENTED IN MEMORY.

EACH NUMERIC QUANTITY OCCUPIES FOUR CONSECUTIVE WORDS (32 BITS) OF MEMORY. THE LARGEST MAGNITUDE THAT CAN BE REPRESENTED IS APPROXIMATELY 3.6 TIMES TEN TO THE 38TH POWER. THE SMALLEST NON-ZERO MAGNITUDE THAT CAN BE REPRESENTED IS APPROXIMATELY 2.7 TIMES TEN TO THE MINUS 39TH POWER. EACH NUMERIC QUANTITY IS REPRESENTED WITH A PRECISION OF ONE PART IN APPROXIMATELY 16,000,000.

THE SOFTWARE CONSTITUTING THE FLOATING POINT SYSTEM IS DIVIDED INTO TWO SECTIONS, EACH OF WHICH OCCUPIES 3 BANKS OF ROM OR RAM. SECTION 1 IS INDEPENDENT OF OTHER SOFTWARE. SECTION 2 IS OPERABLE ONLY WHEN SECTION 1 IS AVAILABLE IN MEMORY. IN ADDITION TO MEMORY REQUIRED FOR PROGRAM, 63 WORDS OF RAM ARE USED AS SCRATCHPAD.

SOFTWARE SECTION 1 CONTAINS THE FOLLOWING SUBROUTINES:

- LOD - LOAD SPECIFIED DATA INTO THE FLOATING POINT ACCUMULATOR.
- ADD - ADD SPECIFIED DATA TO THE FLOATING POINT ACCUMULATOR.
- SUB - SUBTRACT SPECIFIED DATA FROM THE FLOATING POINT ACCUMULATOR.
- MUL - MULTIPLY SPECIFIED DATA TIMES THE FLOATING POINT ACCUMULATOR.
- DIV - DIVIDE SPECIFIED DATA INTO THE FLOATING POINT ACCUMULATOR.
- TST - SET CONTROL BITS TO INDICATE ATTRIBUTES OF THE FLOATING POINT ACCUMULATOR.
- CHS - CHANGE THE SIGN OF THE FLOATING POINT ACCUMULATOR.
- ABS - SET THE SIGN OF THE FLOATING POINT ACCUMULATOR POSITIVE.
- STR - STORE IN SPECIFIED MEMORY THE VALUE IN THE REGISTERS AS RETURNED BY OTHER SUBROUTINES.
- INIT - MOVE CODE FROM ROM TO RAM IN PREPARATION FOR EXECUTION OF THE MUL AND DIV SUBROUTINES.

SOFTWARE SECTION 2 CONTAINS SUBROUTINES WHICH ARE USED TO CONVERT DATA BETWEEN THE BINARY FLOATING POINT FORMAT AND A DECIMAL FORMAT SUITABLE FOR ENTRY OR DISPLAY ON INPUT/OUTPUT EQUIPMENT. THE DECIMAL FORMAT IS STORED IN MEMORY AS A SERIES OF CHARACTERS. RELATIVELY SIMPLE INPUT/OUTPUT ROUTINES MAY BE USED TO INTERFACE THE MEMORY-RESIDENT CHARACTER STRINGS WITH ANY TYPE OF PHYSICAL I/O DEVICE.

THE CHARACTER STRINGS CONSIST OF BCD REPRESENTATIONS OF DECIMAL DIGITS AND ARBITRARY REPRESENTATIONS OF +, -, ., AN EXPONENTIAL SIGN (LETTER E), AND SPACE. CHARACTER STRINGS MAY NOT CROSS MEMORY BANK BOUNDARIES. AN INPUT STRING IS THEREFORE LIMITED TO 256 CHARACTERS. AN OUTPUT STRING CONSISTS OF 13 CHARACTERS.

THE OUT SUBROUTINE GENERATES CHARACTER STRINGS IN 2 FORMATS; THE CHOICE OF FORMAT DEPENDS ON THE MAGNITUDE OF THE VALUE REPRESENTED.

MAGNITUDES BETWEEN .1000000 AND 9999999. ARE REPRESENTED BY A SPACE OR MINUS SIGN, SEVEN DECIMAL DIGITS AND AN APPROPRIATELY POSITIONED DECIMAL POINT, AND FOUR SPACES. MAGNITUDES OUTSIDE THE RANGE ARE REPRESENTED BY A SPACE OR MINUS SIGN, A VALUE BETWEEN 1.000000 AND 9.999999, AN EXPONENTIAL SIGN, AND A SIGNED TWO-DIGIT POWER OF TEN.

THE INP SUBROUTINE CONVERTS CHARACTER STRINGS IN EITHER OF THE ABOVE FORMATS, OR A MODIFIED VERSION OF THEM. THE LEADING SIGN MAY BE INCLUDED OR OMITTED. ANY NUMBER OF DIGITS MAY BE USED TO INDICATE THE VALUE, WITH OR WITHOUT AN INCLUDED DECIMAL POINT. IF A POWER-OF-TEN MULTIPLIER IS INDICATED IT MAY BE SIGNED OR UNSIGNED AND MAY CONTAIN ONE OR TWO DIGITS. AN INPUT STRING IS TERMINATED BY THE FIRST CHARACTER WHICH DEPARTS FROM THE FORMAT.

THE FOLLOWING ARE EXAMPLES OF INPUT AND CORRESPONDING OUTPUT CHARACTER STRINGS.

3.141593	3.141593
-.0000000000001	-1.000000E-13
+1.6E5	160000.0
123456789	1.234568E+08
54321E-10	5.432100E-06
-2718281828E-9	-2.718282

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC2

☐ 4004 ☒ 8008 ☐ 8080 (use additional sheets if necessary)

Program Title

FLOATING POINT FORMAT CONVERSION PACKAGE

Function

Provides subroutines for conversion between floating point format and ASCII or BCD. Functions contained are:

1. Floating point to BCD conversion
2. BCD to floating point conversion
3. Floating point to fixed point (integer) conversion
4. Fixed point to floating point conversion

Required Hardware

None.

Required Software

Floating Point Arithmetic and Utility Package

Input Parameters

Output Results

Registers Modified:	Maximum Subroutine Nesting Level:
RAM Required: 63 bytes	Assembler/Compiler Used: MAC 8
ROM Required: 512 bytes	Programmer: C.E. Ohme
	Company: 44750 Winding Lane Fremont, Ca. 94538

8008 BINARY FLOATING POINT SYSTEM

FORMAT CONVERSION PACKAGE

THE FORMAT CONVERSION PACKAGE OF THE 8008 BINARY FLOATING POINT SYSTEM CONTAINS SUBROUTINES FOR THE CONVERSION OF DATA BETWEEN THE FLOATING POINT SYSTEM NOTATION AND TWO OTHER FORMATS. THE NON-FLOATING-POINT FORMATS ARE FOUR WORD FIXED POINT FORMAT AND VARIABLE LENGTH CHARACTER STRING FORMAT.

THE FORMAT CONVERSION PACKAGE IS CONTAINED IN 512 CONSECUTIVE WORDS OF MEMORY(2 BANKS OF ROM) AND REQUIRES FOR ITS EXECUTION THAT THE ARITHMETIC AND UTILITY PACKAGE BE AVAILABLE IN MEMORY. THE COMBINATION OF THIS PACKAGE AND THE ARITHMETIC AND UTILITY PACKAGE USES THE FIRST 64 WORDS OF A BANK OF RAM AS SCRATCHPAD MEMORY.

THE FIXED POINT FORMAT DATA PROCESSED BY THIS PACKAGE CONSIST OF 32 BIT BINARY NUMBERS OCCUPYING FOUR WORDS. TWOS COMPLEMENT NOTATION IS USED TO REPRESENT NEGATIVE VALUES.
THE POSITION OF THE BINARY POINT RELATIVE TO THE BITS REPRESENTING THE VALUE IS DENOTED BY A BINARY SCALING FACTOR. THE BINARY SCALING FACTOR IS NOT NORMALLY RECORDED IN THE COMPUTER, BUT WHEN A FORMAT CONVERSION SUBROUTINE IS CALLED THE BINARY SCALING FACTOR MUST BE SPECIFIED(IN THE E REGISTER). A BINARY SCALING FACTOR OF ZERO INDICATES THE BINARY POINT IS IMMEDIATELY TO THE LEFT OF THE MOST SIGNIFICANT OF THE 32 BITS REPRESENTING THE VALUE. A BINARY SCALING FACTOR OF 32 INDICATES THE BINARY POINT IS IMMEDIATELY TO THE RIGHT OF THE LEAST SIGNIFICANT BIT. THE PERMISSIBLE RANGE OF THE BINARY SCALING FACTOR IS -128(200 OCTAL) TO +127(177 OCTAL).

THE CHARACTER STRING FORMAT DATA PROCESSED BY THIS PACKAGE CONSIST OF BINARY REPRESENTATIONS OF CHARACTERS OCCUPYING CONSECUTIVE WORDS OF MEMORY. A CHARACTER STRING MAY NOT CROSS A MEMORY BANK BOUNDARY. THE CHARACTERS WHICH MAY BE INCLUDED IN A CHARACTER STRING, AND THE CORRESPONDING OCTAL REPRESENTATIONS ARE LISTED BELOW.

DECIMAL DIGITS	000B-011B BCD DIGITS
SPACE	360B
+	373B PLUS
-	375B MINUS
.	376B DECIMAL POINT
EXPONENTIAL SIGN	025B LETTER E

(THESE OCTAL REPRESENTATIONS CAN BE CONVERTED TO THE CORRESPONDING ASCII CHARACTERS BY ADDING 060B TO EACH)

THE OUT SUBROUTINE GENERATES CHARACTER STRINGS IN TWO FORMATS, EACH CONSISTING OF 13 CHARACTERS. THE FORMAT USED IN A SPECIFIC CASE IS DEPENDENT UPON THE MAGNITUDE OF THE VALUE REPRESENTED.

ZERO AND MAGNITUDES BETWEEN .1000000 AND 9999999. ARE REPRESENTED BY A SPACE OR MINUS SIGN, SEVEN DECIMAL DIGITS AND AN APPROPRIATELY POSITIONED DECIMAL POINT, AND FOUR SPACES.

MAGNITUDES OUTSIDE THE ABOVE RANGE ARE REPRESENTED BY A SPACE OR MINUS SIGN, A VALUE BETWEEN 1.000000 AND 9.999999, AN EXPONENTIAL SIGN, AND A SIGNED TWO DIGIT POWER OF TEN.

THE INP SUBROUTINE CONVERTS CHARACTER STRINGS IN EITHER OF THE ABOVE FORMATS, OR A MODIFIED VERSION OF THEM. THE LEADING SIGN CHARACTER MAY BE INCLUDED OR OMITTED. UP TO 37 DIGITS MAY BE USED TO INDICATE THE VALUE, WITH OR WITHOUT AN INCLUDED DECIMAL POINT. IF A POWER-OF-TEN MULTIPLIER IS INDICATED IT MAY BE SIGNED OR UNSIGNED AND MAY CONTAIN ONE OR TWO DIGITS. AN INPUT CHARACTER STRING IS TERMINATED BY THE FIRST CHARACTER WHICH DEPARTS FROM THE SPECIFIED FORMAT.

THE FOLLOWING ARE EXAMPLES OF INPUT AND CORRESPONDING OUTPUT CHARACTER STRINGS.

3.141593	3.141593
-.0000000000001	-1.000000E-13
+1.6E5	160000.0
123456789	1.234568E+08
54321E-10	5.432100E-06
-2718281828	-2.718282E+09

THE INDIVIDUAL SUBROUTINES INCLUDED IN THE FORMAT CONVERSION PACKAGE OF THE FLOATING POINT SYSTEM ARE DESCRIBED IN DETAIL BELOW.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB1

☐ 4004 ☐ 8008 ☒ 8080 ☐ 4040 (use additional sheets if necessary)

Program Title

16 BIT MULTIPLY - 32 BIT RESULT

Function

MULTIPLICATION OF TWO 16 BIT POSITIVE NUMBERS

Required Hardware

BASIC 8080 SYSTEM

Required Software

23 INSTRUCTIONS

Input Parameters

2 BYTE MULTIPLIER X_1 Xo IN MEMORY
2 BYTE MULTIPLICAND Y_1 Yo IN D&E REGISTERS

Output Results

STARTING WITH MOST SIGNIFICANT BYTE:
A, H, L STACK

Registers Modified: ALL	**Maximum Subroutine Nesting Level:** NONE
RAM Required: 6 BYTES	**Assembler/Compiler Used:**
ROM Required: 42 BYTES	**Programmer:** DAN SOLTZ
	Company: FISCHER & PORTER COUNTY LINE ROAD,WARMINSTER PA. 18974

```
                        ;REF. NO. BB1
                        ;PROGRAM TITLE 16 BIT MULTIPLY (32 BIT RESULTS)
                        ;
                        ;
                        ; USER MUST ENTER PROGRAM WITH MULTIPLICAND
                        ; IN  THE D AND E REGISTERS AND THE MULTIPLIER
                        ; IN THE LOCATIONS SPECIFIED BY AX0 AND AX1 BELOW
0100                    ORG     100H
0100 3AFE00             LDA     AX0     ; LOAD A WITH CONTENTS OF LOC. AX0
0103 0E02               MVI     C,2     ; LOAD PASS COUNTER C WITH 2
0105 0608       PASS2:  MVI     B,8     ; LOAD LOOP COUNTER B WITH 8
0107 210000             LXI     H,0     ; CLEAR H AND L
010A 29         LOOP:   DAD     H       ; SHIFT PARTIAL PRODUCT LEFT INTO CARRY
010B 17                 RAL             ; ROTATE MULTIPLIER BIT INTO CARRY
010C D21201             JNC     DEC     ; TEST MULTIPLIER AT CARRY
010F 19                 DAD     D       ; ADD MULTIPLICAND TO PARTIAL PRODUCT
0110 CE00               ACI     0       ; ADD CARRY TO RESULTS MS BYTE
0112 05         DEC:    DCR     B       ; DECREMENT LOOP COUNTER
0113 C20A01             JNZ     LOOP    ; DO LOOP 8 TIMES
0116 0D                 DCR     C       ; DECREMENT PASS COUNTER
0117 CA2401             JZ      DONE    ; AFTER 2ND PASS JUMP TO DONE
011A E5                 PUSH    H       ; STORE LS RESULT BYTE
011B 6C                 MOV     L,H     ; MOV PARTIAL PRODUCT
011C 67                 MOV     H,A     ;  TO H AND L
011D E5                 PUSH    H       ;   AND STORE IN STACK
011E 3AFF00             LDA     AX1     ; LOAD A WITH X1
0121 C30501             JMP     PASS2   ; GO TO SECOND PASS ROUTINE
0124 D1         DONE:   POP     D       ; RETURN PARTIAL PRODUCT FROM STACK
0125 19                 DAD     D       ; ADD PART. PRODUCTS FROM TWO PASSES
0126 CE00               ACI     0       ; ADD CARRY TO MS BYTE
0128 76                 HLT
00FE            AX0     EQU     0FEH
00FF            AX1     EQU     0FFH
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB2

☐ 4004 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

16-BIT MULTIPLY - 16-BIT RESULT - MPY16

Function

Performs a 16 x 16 bit multiply giving a 16 bit result

Required Hardware

NONE

Required Software

NONE

Input Parameters

D,E registers contain multiplier
H,L registers contain multiplicand

Output Results

B,C contain result

Registers Modified: All	**Maximum Subroutine Nesting Level:** Ø
RAM Required: 3 bytes	**Assembler/Compiler Used:** MAC 30
ROM Required: 25H bytes	**Programmer:**
	Company:

```
                        ;REF NO. BB2
                        ;PROGRAM TITLE MPY 16
                        ;
                        ;
                        ;
                        MPY80:
0000 222800                     SHLD    TEMP    ;STORE MULTIPLICAND IN TEMPORARY
0003 212A00                     LXI     H,BNUM  ;STORE
0006 3611                       MVI     M,11H   ;  BIT COUNT
                                LXI     B,0     ;INITIALIZE RESULT
                        LOOP:
000B 7A                         MOV     A,D     ;ROTATE
000C 1F                         RAR
000D 57                         MOV     D,A     ;  MULTIPLIER
000E 7B                         MOV     A,E
000F 1F                         RAR             ;   RIGHT
0010 5F                         MOV     E,A
0011 35                         DCR     M       ;DECREMENT BIT COUNT
0012 C8                         RZ              ;DONE?  THEN RETURN
0013 D21F00                     JNC     SKIP    ;NO CARRY FROM ROTATE
  )16 2A2800                    LHLD    TEMP    ;  OTHERWISE
  )19 09                        DAD     B       ;   ADD
001A 44                         MOV     B,H     ;    MULTIPLICAND
001B 4D                         MOV     C,L     ;SAVE RESULT
001C 212A00                     LXI     H,BNUM  ;RESTORE H AND L REGISTERS
                        SKIP:
                                MOV     A,B     ;ROTATE
0020 1F                         RAR             ;  TEMP
0021 47                         MOV     B,A     ;   RESULT
0022 79                         MOV     A,C     ;    RIGHT
0023 1F                         RAR
0024 4F                         MOV     C,A
0025 C30B00                     JMP     LOOP    ;REPEAT LOOP
                        TEMP:
0028                            DS 2
                        BNUM:
002A                            DS 1
0000                            END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB3

☐ 4004 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

16-BIT MULTIPLY - 16-BIT RESULT

Function

Performs 16-bit x 16-bit multiplication giving a 16-bit result.

Required Hardware

None.

Required Software

None.

Input Parameters

B,C contain multiplicand.
D,E contain multiplier.

Output Results

B,C contain result.

Registers Modified: A, B, C, D, E, H, L	**Maximum Subroutine Nesting Level:** 0
RAM Required: 3 bytes	**Assembler/Compiler Used:** MAC-8
ROM Required: 35H bytes	**Programmer:**
	Company:

```
                        ;REF. NO. BB3
                        ;PROGRAM TITLE 16-BIT MULTIPLY
                        ;
                        ;
                        ;
                        ;
                        ;THIS SUBROUTINE PERFORMS A 16 BIT MULTIPLICATION
                        ;GIVING A 16 BIT RESULT.  THE NUMBERS MUST BE 16 BIT
                        ;UNSIGNED QUANTITIES.  THE B AND C REGISTERS SHOULD
                        ;CONTAIN THE MULTIPLICAND AND THE D AND E REGISTERS
                        ;SHOULD CONTAIN THE MULTIPLIER.  THE RESULT WILL BE
                        ;FOUND IN THE D AND E REGISTERS.

                        MPY16:
0000 213100                     LXI H,MPCD      ;TEMP  ADDRESS OF MULTIPLICAND
0003 71                         MOV M,C         ;SAVE MULTIPLICAND
0004 2C                         INR L
0005 70                         MOV M,B
0006 213300                     LXI H,BNUM      ;TEMP ADDRESS OF BIT COUNTER
  09 3611                       MVI M,11H       ;INITIALIZE BIT COUNTER
000B 0600                       MVI B,0H        ;B AND C REGS TO ZERO
000D 48                         MOV C,B
                        LOOP:
000E 7A                         MOV A,D         ;SHIFT
000F 1F                         RAR             ;  MULTIPLIER
0010 57                         MOV D,A         ;    RIGHT
0011 7B                         MOV A,E         ;      ONE
0012 1F                         RAR             ;       BIT
0013 5E                         MOV E,M         ;DECREMENT
0014 1D                         DCR E           ;  BIT
0015 73                         MOV M,E         ;    COUNTER
0016 5F                         MOV E,A
0017 C8                         RZ              ;DONE IF ALL BITS USED
0018 D22800                     JNC SKIP        ;NO CARRY FROM SHIFT SO NO ADD
001B 213100                     LXI H,MPCD      ;GET MULTIPLICAND ADDRESS
001E 79                         MOV A,C         ;ADD LOW ORDER
001F 86                         ADD M           ;  OF MULTIPLICAND TO RESULT
0020 4F                         MOV C,A         ;SAVE RESULT
0021 2C                         INR L           ;HIGH ORDER MULTIPLICAND ADDRESS
0022 78                         MOV A,B         ;ADD  HIGH ORDER
0023 8E                         ADC M           ;  OF MULTIPLICAND WITH CARRY
0024 47                         MOV B,A         ;SAVE RESULT
0025 213300                     LXI H,BNUM      ;RESTORE BIT COUNTER ADDRESS
                        SKIP:
0028 78                         MOV A,B         ;SHIFT
  29 1F                         RAR             ;  RESULT
002A 47                         MOV B,A         ;    RIGHT
002B 79                         MOV A,C         ;      ONE
```

```
002C 1F                 RAR             ;     BIT
002D 4F                 MOV C,A
002E C30E00             JMP LOOP        ;REPEAT STEPS IN LOOP
                MPCD:
0031                    DS 2            ;SPACE FOR TEMP MULTIPLICAND
                BNUM:
0033                    DS 1            ;SPACE FOR BIT COUNTER
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB4

☐ 4004 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

16-BIT 2'S COMPLEMENT SIGNED MULTIPLICATION - SMPY16

Function

16-Bit Binary Multiplication - 32-Bit Result

Required Hardware

None

Required Software

None

Input Parameters

B,C contain multiplicand
H,L contain address of multiplier

Output Results

32-bit product returned at
(H,L) to (H,L+3)

Registers Modified:	Maximum Subroutine Nesting Level:
All	1
RAM Required:	**Assembler/Compiler Used:**
4 bytes	MAC8
ROM Required:	**Programmer:** R.M. Gabrielson General Electric
68H	
	Company: Aerospace Controls Box 5000, Binghamton NY 13902

```
                ;REF.  NO. BB4.
                ;PROGRAM TITLE SMPY16
                ;
                ;
                ;
                ;
                ;
0400                    ORG     1024
                ;  DOUBLE PRECISION 16 BIT ADD
                ;  LO 8 BITS AUGEND IN B
                ;  HI 8 BITS AUGEND IN C
                ;  LO 8 BITS ADDEND AT (H,L)
                ;  HI 8 BITS ADDEND AT (H,L+1)
                ;  RESULT RETURNED IN B(LO SUM) AND C(HI SUM)
                ADBL:
0400 78                 MOV     A,B
0401 86                 ADD     M       ;LO AUG + LO ADD
0402 47                 MOV     B,A     ;B=LO SUM
0403 79                 MOV     A,C
0404 2C                 INR     L
0405 8E                 ADC     M
0406 8E                 ADC     M       ;HI AUFG + HI ADD + LO CARRY
0407 4F                 MOV     C,A
0408 C9                 RET
0564                    ORG     25440
                ;  16-BIT 2'S COMPLEMENT MULTIPLY
                ;  MULTIPLICAND IN B,C
                ;  MULTIPLIER AT (H,L)
                ;  PRODUCT RETURNED AT (H,L) - (H,L+3)
                ;  AFFECTS ALL REGISTERS
                ;
                MDBL:
0564 56                 MOV     D,M     ;MULTIPLIER TO D,E
0565 2C                 INR     L
0566 5E                 MOV     E,M
0567 AF                 XRA     A
0568 83                 ADD     E
0569 F27905             JP      SETT    ;IF MPLR POSITIVE
                ;COMPLEMENT MPLR & MPCD IF MPLR < 0
056C AF                 XRA     A
056D 6F                 MOV     L,A
056E 90                 SUB     B
056F 47                 MOV     B,A
0570 7D                 MOV     A,L
0571 99                 SBB     C
0572 4F                 MOV     C,A
 573 7D                 MOV     A,L
0574 92                 SUB     D
0575 57                 MOV     D,A
```

```
0576 7D                 MOV     A,L
0577 9B                 SBB     E
0578 5F                 MOV     E,A
                    ;SET UP FOR MULTIPLY
                    SETT:
0579 21E70B             LXI     H,MPCD
057C 70                 MOV     M,B
057D 2C                 INR     L
057E 71                 MOV     M,C
057F AF                 XRA     A
0580 47                 MOV     B,A
0581 4F                 MOV     C,A
0582 21E90B             LXI     H,TEST
0585 360F               MVI     M,15
                    ;BEGIN MULTIPLY
                    NEXT:
0587 CDB105             CALL    SHIFT
058A D29305             JNC     DECR        IF MPLR BIT = 0
058D 21E70B             LXI     H,MPCD
0590 CD0004             CALL    ADBL
                    DECR:
 593 21E90B             LXI     H,TEST
 596 3EFF               MVI     A,-1
0598 86                 ADD     M
0599 77                 MOV     M,A
059A C28705             JNZ     NEXT        IF NOT DONE
059D AF                 XRA     A
059E CDB105             CALL    SHIFT
05A1 AF                 XRA     A
05A2 CDB105             CALL    SHIFT       REPEAT SIGN
                     STORE PRODUCT IN MEM.
05A5 2C                 INR     L
05A6 71                 MOV     M,C
05A7 2D                 DCR     L
05A8 70                 MOV     M,B
05A9 2D                 DCR     L
05AA 73                 MOV     M,E
05AB 2D                 DCR     L
05AC 72                 MOV     M,D
05AD 21E70B             LXI     H,MPCD      ADDRESS OF PRODUCT
05B0 C9                 RET
                    ;4-BYTE ARITHMETIC RIGHT SHIFT
                    SHIFT:
05B1 79                 MOV     A,C
05B2 07                 RLC                 ;REPEAT SIGN
05B3 1F                 RAR
05B4 1F                 RAR
 5B5 4F                 MOV     C,A
05B6 78                 MOV     A,B
05B7 1F                 RAR
```

```
05B8 47                 MOV     B,A
05B9 7B                 MOV     A,E
05BA 1F                 RAR
05BB 5F                 MOV     E,A
05BC 7A                 MOV     A,D
05BD 1F                 RAR
05BE 57                 MOV     D,A
05BF C9                 RET
0BE7            MPCD    EQU     5747Q
0BE9            TEST    EQU     5751Q
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB5

☐ 4004 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

BINARY MULTIPLICATION - 24-BIT

Function

Multiplies two binary numbers:
Multiplicand - 24 bits
Multiplier - 1 to 24 bits

Required Hardware

NONE

Required Software

NONE

Input Parameters

H and L registers pointing to Least Significant byte of multiplier
D and E registers pointing to Least Significant byte of multiplicand
A register has the number of multiplier bits plus 1

Output Results

H and L registers pointing to Least Significant byte of Product
(see program for memory location)

Registers Modified: All except H and L	**Maximum Subroutine Nesting Level:** 1
RAM Required: 7 bytes	**Assembler/Compiler Used:** MAC80
ROM Required: 110 bytes	**Programmer:** Chon Hock Leow Tektronix
	Company: Box 500, Dept. 50-447 Beaverton, Oregon 97077

```
                ;REF. NO. BB5.
                ;PROGRAM NAME BINARY MULTIPLICATION
                ;
                ;
                ;
                ;
                ;
                ;*****************************
                ;BINARY MULTIPLICATION
                ;*****************************
                ;
                ;
                ;       ON ENTERING THIS SUBROUTINE
                ;HL REG.  POINTS TO LS BYTE OF MULTIPLIER
                ;DE REG.  POINTS TO LS BYTE OF MULTIPLICAND
                ;A REG.  HAS THE NUMBER OF MULTIPLEIER BITS +1 (2 TO "25)
                ;
                ;       ON EXIT
                ;HL REG. POINTS TO LS BYTE OF PRODUCT
                ;NOT:  MEMORY ALLOCATION FOR PRODUCT
                ;
                ;ENTER:XXXMMM
                ;EXIT  :PPPPPP
                ;      X IS DONI'T CARE
                ;      M IS MULTIPLIER
                ;      P IS PRODUCT
                ;
                ;
                ;
0003            BY3     EQU     3       ;3 BYTES
0010            LSR     EQU     10H     ;LS PARTIAL PRODUCT POINTER SAVED LOCATI
0012            MSR     EQU     12H     ;MS PRODUCT POINTER SAVED LOCATION
0014            LSMCD   EQU     14H     ;LS MULTIPLICAND POINTER SAVED LOCATION
0016            NBIT    EQU     16H     ;NUMBER OF BITS SAVED LOCATION
                ;
0100                    ORG     100H
                ;
                ;INITIALIZATION OF RESULT BUFFER
                ;
                MULT:
0100 E5                 PUSH    H       ;SAVE MULTIPLIER POINTER
0101 2B                 DCX     H
0102 2B                 DCX     H
0103 2B                 DCX     H
0104 221000             SHLD    LSR     ;SAVE LS PARTIAL PRODUCT POINTER
 107 321600             STA     NBIT    ;SAVE NUMBER OF MULTIPLIER BITS IN NBIT
010A AF                 XRA     A       ;CLEAR A
010B 77                 MOV     M,A
```

```
010C 2B             DCX     H
010D 77             MOV     M,A
010E 2B             DCX     H
010F 77             MOV     M,A
0110 221200         SHLD    MSR         ;SAVE MS PRODUCT POINTER
0113 EB             XCHG
0114 221400         SHLD    LSMCD       ;SAVE LS MULTIPLICAND POINTER
                    ;
                    ;TEST FOR ZERO
                    ;
0117 0600           MVI     B,0
0119 E1             POP     H           ;PICK UP LS MULTIPLIER POINTER
011A E5             PUSH    H
011B 7E             MOV     A,M
011C 2B             DCX     H
011D 8E             ADC     M
011E 2B             DCX     H
011F 8E             ADC     M
0120 B8             CMP     B           ;COMPARE WITH B REG.  (0)
0121 CA5E01         JZ      EXIT        ;EXIT IF MULTIPLIER = 0
                    ;
                    ;ROTATE MULTIPLIER ROUND AND TEST
                    ;
0124 3A1600         LDA     NBIT
0127 47             MOV     B,A         ;B REG.  HAS NUMBER OF MULTIPLER BITS
0128 AF             XRA     A           ;CLEAR CARRY BIT
                    LOOP:
0129 CD6001         CALL    ROT1        ;CALL ROTATE ROUTINE
012C 05             DCR     B           ;DECREMENT COUNT FOR MULTIPLIER BITS
012D CA4901         JZ      FINIS       ;IF ZERO -> FINISH
0130 D22901         JNC     LOOP        ;IF CARRY=0 -> JUST ROTATE
                    ;
                    ;ADDTION ROUTINE
                    ;
0133 2A1400         LHLD    LSMCD       ;PICK UP LS MULTIPLICAND POINTER
0136 EB             XCHG
0137 2A1000         LHLD    LSR         ;PICK UP LS PARTIAL PRODUCT POINTER
013A 0E03           MVI     C,BY3       ;COUNT FOR 3 BYTES
013C AF             XRA     A           ;CLEAR CARRY
                    ADD1:
013D 1A             LDAX    D           ;PICK UP MULTIPLICAND
013E 8E             ADC     M           ;ADD IN TO RESULT
013F 77             MOV     M,A
0140 1B             DCX     D
0141 2B             DCX     H
0142 0D             DCR     C
0143 C23D01         JNZ     ADD1        ;IF ZERO -> FINISH ADDING
 146 C32901         JMP     LOOP
                    ;
                    ;ALIGN PRODUCT TO WHERE ILS MULTIPLIER WAS
```

```
                        ;
                        FINIS:
0149 3A1600             LDA     NBIT    ;COMPUTER BY3*8 -NBIT + 1
014C 47                 MOV     B,A
014D 3E18               MVI     A,BY3*8
014F 90                 SUB     B
0150 C601               ADI     1
0152 CA5E01             JZ      EXIT    ;DON'T NEED ATO ALIGN IF NBIT=25
0155 47                 MOV     B,A     ;B REG. HAS THE COUNT OF ALIGNMENT
0156 AF                 XRA     A       ;CLEAR CARRY
                        ADJ:
0157 CD6001             CALL    ROT1    ;CALL ROTATE ROUTINE
015A 05                 DCR     B
015B C25701             JNZ     ADJ
                        EXIT:
015E E1                 POP     H       ;RETURN WITH HL POINTER TO LS RESULT
015F C9                 RET             ;RETURN TO CALLING POINT
                        ;
                        ; ROTATE ROUTINE
                        ;
                        ROT1:
0160 0E06               MVI     C,BY3*2 ;COUNT FOR 6 BYTES
0162 2A1200             LHLD    MSR     ;PICKUP MS RESULT POINTER
                        ROT2:
0165 7E                 MOV     A,M
0166 1F                 RAR
0167 77                 MOV     M,A
0168 23                 INX     H
0169 0D                 DCR     C
016A 0D                 DCR     C       ;DECREMENT COUNT OF BYTES OF ROTATION
016B C26501             JNZ     ROT2    ;IF NOT ZERO,KEEP ROTATING
016E C9                 RET

0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB8

☐ 4004 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

16-BIT DIVISION - 16-BIT RESULT

Function

Performs integer division of 16-bit quantities. Gives a 16-bit result.

Required Hardware

None.

Required Software

None.

Input Parameters

B,C contain divisor.
D,E contain the dividend.

Output Results

D,E contain the result.

Registers Modified: All	**Maximum Subroutine Nesting Level:** 0
RAM Required: 3 bytes	**Assembler/Compiler Used:** MAC 8
ROM Required: 40H bytes	**Programmer:**
	Company:

```
                        ;REF. NO. BB8
                        ;PROGRAM TITLE 16-BIT DIVISION
                        ;
                        ;
                        ;
                        ;THIS SUBROUTINE PERFORMS 16 BIT DIVISION GIVING A 16
                        ;BIT RESULT. THE B AND C REGISTERS CONTAIN THE
                        ;DIVISOR AND D AND E REGISTERS CONTAIN THE DIVIDEND.
                        ;THE RESULT IS RETURNED IN THE D AND E REGISTERS.
                        ;NOTE: BOTH QUANTITIES ARE 16 BIT UNSIGNED NUMBERS.

                        DIV16:
0000 213C00                     LXI H,TEMP      ;ADDRESS OF TEMP CELL
0003 71                         MOV M,C         ;SAVE DIVIDEND
0004 2C                         INR L
0005 70                         MOV M,B
0006 213E00                     LXI H,BNUM      ;ADDRESS OF BIT COUNTER
0009 3611                       MVI M,11H       ;INITIALIZE BIT COUNTER
  0B 0600                       MVI B,0H        ;SET B AND C TO ZERO
000D 48                         MOV C,B
                        LOOP:
000E 7B                         MOV A,E         ;ROTATE
000F 17                         RAL             ; DIVISOR
0010 5F                         MOV E,A         ;  LEFT
0011 7A                         MOV A,D         ;   ONE
0012 17                         RAL             ;    BIT
0013 56                         MOV D,M         ;DECREMENT
0014 15                         DCR D           ; BIT
0015 72                         MOV M,D         ;  COUNTER
0016 57                         MOV D,A
0017 C8                         RZ              ;RETURN IF BIT COUNTER ZERO
0018 79                         MOV A,C         ;ROTATE
0019 17                         RAL             ; RESULT
001A 4F                         MOV C,A         ;  LEFT
001B 78                         MOV A,B         ;   ONE
001C 17                         RAL             ;    BIT
001D 47                         MOV B,A
001E 213C00                     LXI H,TEMP      ;ADDRESS OF DIVIDEND
0021 79                         MOV A,C         ;SUBTRACT
0022 96                         SUB M           ; LOW ORDER
0023 4F                         MOV C,A         ;  OF DIVIDEND
0024 2C                         INR L
0025 78                         MOV A,B         ;SUBTRACT
0026 9E                         SBB M           ; HIGH ORDER
0027 47                         MOV B,A         ;  OF DIVIDEND
0028 D22B00                     JNC SKIP        ;NO BORROW SO SKIP ADD
002B 2D                         DCR L           ;ADDRESS OF LOW ORDER
```

```
002C 79                 MOV A,C         ;ADD
002D 86                 ADD M           ; DIVIDEND
002E 4F                 MOV C,A         ;   BACK
002F 2C                 INR L           ;    IN
0030 78                 MOV A,B
0031 8E                 ADC M
0032 47                 MOV B,A
                SKIP:
0033 213E00             LXI H,BNUM      ;RESTORE ADDRESS OF BIT COUNTER
0036 9F                 SBB A           ;COMPLEMENT
0037 DE80               SBI 80H         ; CARRY
0039 C30E00             JMP LOOP        ;REPEAT LOOP
                TEMP:
003C                    DS 2            ;CELL FOR DIVIDEND
                BNUM:
003E                    DS 1            ;CELL FOR BIT COUNTER
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB9

☐ 4004 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

16-BIT DIVISION - 16-BIT RESULT - DIV16

Function

Performs a 16 bit x 16 bit division giving a 16 bit quotient and a 16 bit remainder

Required Hardware

NONE

Required Software

NONE

Input Parameters

D,E contain dividend
H,L contain divisor

Output Results

D,E contain result
H,L contain remainder

Registers Modified: All	**Maximum Subroutine Nesting Level:** 0
RAM Required: 3 bytes	**Assembler/Compiler Used:** MAC 80
ROM Required: 31H bytes	**Programmer:**
	Company:

```
                        ;REF. NO. BB9
                        ;PROGRAM TITLE DIV16
                        ;
                        ;
                        ;
                        DIV80:
0000 223100                     SHLD TEMP       ;SAVE DIVIDEND IN TEMPORARY
0003 213300                     LXI H,BNUM      ;STORE
0006 3611                       MVI M,11H       ;  BIT COUNT
0008 010000                     LXI B,0         ;INITIALIZE RESULT
000B C5                         PUSH B          ;SAVE RESULT ON STACK
                        LOOP:
000C 7B                         MOV A,E         ;GET LOW DIVISOR BYTE
000D 17                         RAL
000E 5F                         MOV E,A         ;SHIFT DIVISOR LEFT ONE BIT
000F 7A                         MOV A,D
0010 17                         RAL             ;RETURN DIVISOR TO D & E
0011 57                         MOV D,A
0012 35                         DCR M           ;DECREMENT BIT COUNT
  13 E1                         POP H           ;RESTORE TEMP RESULT
0014 C8                         RZ              ;ZERO BIT COUNT MEANS ALL DONE
0015 3E00                       MVI A,0         ;ADD IN
0017 CE00                       ACI 0           ;  CARRY
0019 29                         DAD H           ;SHIFT TEMP RESULT LEFT ONE BIT
001A 44                         MOV B,H         ;COPY H& L TO A & C.
001B 85                         ADD L
001C 2A3100                     LHLD TEMP       ;GET ADDRESS OF DIVIDEND
001F 95                         SUB L           ;SUBTRACT FROM
0020 4F                         MOV C,A
0021 78                         MOV A,B
0022 9C                         SBB H              TEMPORARY RESULT
0023 47                         MOV B,A
0024 C5                         PUSH B          ;SAVE TEMP RESULT ON STACK
0025 D22A00                     JNC SKIP        ;NO BORROW FROM SUBTRACT
0028 09                         DAD B           ;  ADD DIVIDEND BACK IN
0029 E3                         XTHL            ;REPLACE TEMP RESULT ON STACK
                        SKIP:
002A 213300                     LXI H,BNUM      ;RESTORE H & L
002D 3F                         CMC             ;COMPLEMENT CARRY
002E C30C00                     JMP LOOP        ;REPEAT LOOP STEPS
                        TEMP:
0031                            DS 2
                        BNUM:
0033                            DS 1
0000                            END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB10

☐ 4004 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title: BINARY TO BCD CONVERSION ROUTINE

Function: Converts binary value (1-24 bits) to its BCD value (1-8 digits)

Required Hardware: NONE

Required Software: NONE

Input Parameters: H and L registers pointing to Most Significant BCD digits buffer
D and E registers pointing to Least Significant byte of binary value
A register has the number of bits to be converted.

Output Results: H and L registers pointing to Most Significant BCD digit buffer

Registers Modified: All but H,L and D,E	**Maximum Subroutine Nesting Level:** 1
RAM Required: 3 bytes	**Assembler/Compiler Used:** MAC80
ROM Required: 176 bytes	**Programmer:** Chon Hock Leow Tektronix
	Company: Box 500, Dept. 50-447 Beaverton, Oregon 97077

```
                ;REF NO.  BB10
                ;PROGRAM NAME BIN TO BCD ROUTINE
                ;
                ;
                ;
                ;
                ;**********************
                ;BINARY TO BCD ROUTINE
                ;**********************
                ;
                ;
                ;
                ;        ON ENTERING THIS PROGRAM
                ;HL REG.  POINTS TO MS BCD DIGIT BUFFER
                ;        (DS=8 FOR MAXIMUM CASE I.E, 24 BITS)
                ;DE REG.  POINTS TO LS BYTE OF BINARY VALUE
                ;A REG.  HAS THE NUMBER OF BITS TO BE CONVERTED
                ;
                ;        ON EXIT
                ;HL REG.  POINTERS TO MS BCD DIGIT BUFFER
                ;NOTE: 1 BCD DIGIT PER BYTE FOR EASY CONVERSION TO ASCII
                ;EG.  A REG,=10 -> BCD BUFFER NEEDS ONLY 4 BYTES
                ;
                ;IF BINARY VALUE IS 0, BCD BUFFER IS NOT CLEARED
                ;
                ;
                ;
0010            SBIN    EQU     10H     ;ADDRESS FOR SAVING DE REG.
0012            NBIT    EQU     12H     ;NUMBER OF BITS SAVED LOCATION
                ;
0200                    ORG     200H
                ;
                BINBCD:
0200 321200             STA     NBIT    ;SAVE NUMBER OF BITS FOR CONVERSION
0203 D5                 PUSH    D       ;SAVE DE,POINTER FOR LSB OF BINARY
0204 E5                 PUSH    H       ;SAVE HL,POINTER FOR MS BCD
0205 EB                 XCHG
0206 221000             SHLD    SBIN    ;SAVE BIN POINTER FOR DIGIT ROUTINE USAG
                ;
                ;TEST IF BINARY VALUE IS ZERO
                ;
0209 AF                 XRA     A       ;CLEAR A REG.
020A 7E                 MOV     A,M     ;PICK UP LS BYTE
020B 2B                 DCX     H
020C 8E                 ADC     M       ;ADD IN NEXT HIGHER BYTE
020D 2B                 DCX     H
020E 8E                 ADC     M       ;ADD IN MS BYTE
020F CA7F02             JZ      EXIT    ;IF BINARY VALUE = 0 -> EXIT
                ;
```

```
0212 EB                 XCHG            ;RESTORE BCD POINTER
0213 3A1200             LDA     NBIT    ;PICK UP NUMBER OF BITS OF CONVERSION
0216 FE18               CPI     24      ;2**24 = 1.6777216 * 10**7
0218 CA3C02             JZ      TEN7
021B FE12               CPI     18      ;2**18 = 2.62144 * 10**5
021D D24402             JNC     TEN6    ;NO CARRY => G.T. 2**18
0220 FE0F               CPI     15      ;2**15 = 3.2768 * 10**4
0222 D24C02             JNC     TEN5    ;NO CARRY => G.T. 2**15
0225 FE0C               CPI     12      ;2**12 = 4.096 * 10**3
0227 D25402             JNC     TEN4    ;NO CARRY => G.T. 2**12
022A FE08               CPI     8
022C D25C02             JNC     TEN3
022F FE06               CPI     6
0231 D26402             JNC     TEN2
0234 FE03               CPI     3
0236 D26C02             JNC     TEN1
0239 C37402             JMP     TEN0    ;REACHED IF NBIT <= 3
                        ;
                        ;BEGIN CONVERION
                        ;
023C 1698       TEN7:   MVI     D,98H   ;989680H = 10**7
023E 018096             LXI     B,9680H
0241 CD8202             CALL    DIGIT
0244 160F       TEN6:   MVI     D,0FH   ;0F4240H = 10**6
0246 014042             LXI     B,4240H
0249 CD8202             CALL    DIGIT
024C 1601       TEN5:   MVI     D,1     ;0186A0H = 10**5
024E 01A086             LXI     B,86A0H
0251 CD8202             CALL    DIGIT
0254 1600       TEN4:   MVI     D,0
0256 011027             LXI     B,10000
0259 CD8202             CALL    DIGIT
025C 1600       TEN3:   MVI     D,0
025E 01E803             LXI     B,1000
0261 CD8202             CALL    DIGIT
0264 1600       TEN2:   MVI     D,0
0266 016400             LXI     B,100
0269 CD8202             CALL    DIGIT
026C 1600       TEN1:   MVI     D,0
026E 010A00             LXI     B,10
0271 CD8202             CALL    DIGIT
0274 1600       TEN0:   MVI     D,0
0276 010100             LXI     B,1
0279 CD8202             CALL    DIGIT
027C C37F02             JMP     EXIT

                        EXIT:
027F E1                 POP     H       ;RETURN HL POINTING TO MS BCD
0280 D1                 POP     D       ;RETURN DE POINTING FO LSB OF BINARY
0281 C9                 RET
```

```
                        ;
                        ;SUBRACTION ROUTINE
                        ;
                        DIGIT:
0282 3600                       MVI     M,0     ;INITIALIZE DIGIT
                                                ;FOR BCD, DIGIT=0
                                                ;FOR ASCII, DIGIT=30H
0284 E5                         PUSH    H       ;SAVE BCD POINTER
                        SUB1:
0285 2A1000                     LHLD    SBIN    ;PICK UP BIN POINTER
0288 7E                         MOV     A,M     ;PICK UP LS BYTE
0289 91                         SUB     C
028A 77                         MOV     M,A     ;PUTS IT BACK
028B 2B                         DCX     H       ;POINTS TO 2ND LS BYTE
028C 7E                         MOV     A,M
028D 98                         SBB     B
028E 77                         MOV     M,A
028F 2B                         DCX     H       ;POINTS TO MS BYTE
0290 7E                         MOV     A,M
0291 9A                         SBB     D
0292 77                         MOV     M,A
0293 7A                         MOV     A,D     ;SAVE D REG.
0294 DA9E02                     JC      RSTR    ;IF CARRY RESTORE
0297 E1                         POP     H       ;PICK BCD POINTER
0298 34                         INR     M       ;INCREMENT BCD POINTER
0299 E5                         PUSH    H       ;SAVE BCD POINTER
029A 57                         MOV     D,A     ;RESTORE D REG.
029B C38502                     JMP     SUB1
                        ;
                        ;RESTORE ROUTINE
                        ;
                        RSTR:

029E 57                         MOV     D,A     ;RESTORE D REG.
029F 2A1000                     LHLD    SBIN    ;PICK UP BIN POINTER
02A2 7E                         MOV     A,M
02A3 81                         ADD     C
02A4 77                         MOV     M,A
02A5 2B                         DCX     H       ;POINTS TO 2ND LS BYTE
02A6 7E                         MOV     A,M
02A7 88                         ADC     B
02A8 77                         MOV     M,A
02A9 2B                         DCX     H
02AA 7E                         MOV     A,M     ;PICKS UP MS BYTE
02AB 8A                         ADC     D
02AC 77                         MOV     M,A
02AD E1                         POP     H       ;PICKS UP BCD POINTER
02AE 23                         INX     H       ;POINTS TO NEXT BCD DIGIT
02AF C9                         RET
0000                            END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB11

☐ 4004 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

BCD TO BINARY CONVERSION ROUTINE

Function

Converts BCD value (1-8 digits) to Binary value (max. 24 bits)

Required Hardware

NONE

Required Software

NONE

Input Parameters

DE register pointing to Most Significant BCD digit
A register has the number of BCD digits to be converted

Output Results

A register has the Most Significant 8-bits
H and L register has the lower 16 bits

Registers Modified: All	**Maximum Subroutine Nesting Level:** 1
RAM Required: NONE	**Assembler/Compiler Used:** MAC 80
ROM Required: 38 bytes	**Programmer:** Chon Hock Leow Tektronix
	Company: Box 500, Dept. 50-447 Beaverton, Oregon 97077

```
                ;REF. NO. BB11.
                ;PROGRAM NAME BCD TO BIN ROUTINE
                ;
                ;
                ;
                ;
                ;
                ;       ***************************
                ;       BCD TO BIN ROUTINE
                ;       ***************************
                ;
                ;
                ;       ON ENTERING THIS PROGRAM
                ;DE REG.  POINTS TO MS BCD DIGIT
                ;A REG.  HAS THE NUMBER OF BCD DIGITS TO BE CONVERTED
                ;
                ;       ON EXIT
                ;THE BINARY VALUE IS IN A,H,L REG, WITH A REG. HAVING
                ;THE MS 8 BITS AND HL REG, HAVING THE LOWER 16 BITS
                ;
                ;
;300                     ORG     300H
                ;
                BCDBIN:
0300 4F                 MOV     C,A     ;MOVE NUMBER OF BCD DIGITS TO C REG.
0301 210000             LXI     H,0     ;INITIALIZE H & L RTO 0
0304 AF                 XRA     A       ;AND A TO 0
                LOOP:
0305 CD0E03             CALL    BCD     ;CALL *10 ROUTINE
0308 13                 INX     D       ;POINT TO NEXT LOWER BCD DIGIT
0309 0D                 DCR     C
030A C20503             JNZ     LOOP    ;IF CONVERSION IS NOT FINISHED,KEEP GOIN
030D C9                 RET
                ;
                ;*10 ROUTINE
                ;
                BCD:
030E D5                 PUSH    D       ;SAVE DE POINTER IN STACK
030F E5                 PUSH    H       ;SAVE ACCUMULATED LOWER 16 BITS
0310 47                 MOV     B,A     ;SAVE A IN B REG,
0311 29                 DAD     H       ;AHL <- AHL * 2
0312 8F                 ADC     A
0313 29                 DAD     H       ;AHL <- AHL * 4
0314 8F                 ADC     A
0315 D1                 POP     D       ;RESTORE OLD LOWER 16 BITS TO DE REG,
0316 19                 DAD     D       ;AHL <- AHL * 5
 317 88                 ADC     B
0318 29                 DAD     H       ;AHL <- AHL *1 10
0319 8F                 ADC     A
```

```
031A 47              MOV     B,A     ;SAVE A IN B REG.
031B D1              POP     D       ;RESTORE BCD DIGIT POINTER
031C 1A              LDAX    D       ;GET NEW DIGIT
031D 85              ADD     L       ;ADD IN NEW DIGIT
031E 6F              MOV     L,A
031F 7C              MOV     A,H
0320 CE00            ACI     0       ;ADD IN CARRY
0322 67              MOV     H,A
0323 78              MOV     A,B
0324 CE00            ACI     0       ;ADD IN CARRY
0326 C9              RET
0000                 END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB12

☐ 4004 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

BCD TO/FROM BINARY CONVERSION

Function

Subroutines are provided for:
1. BCD to binary conversion
2. Binary to BCD conversion

Required Hardware

None.

Required Software

None.

Input Parameters

BCD to Binary
1. D,E accumulated binary value
2. H,L pointer to BCD string

Binary to BCD
1. D,E contain binary value
2. H,L point to location to contain BCD string

Output Results

D,E contain accumulated binary equivalent.
H,L contain address of BCD string.

Registers Modified: All	**Maximum Subroutine Nesting Level:** 1
RAM Required: None	**Assembler/Compiler Used:** MAC 8
ROM Required: BCD to binary 32 bytes Binary to BCD 67 bytes	**Programmer:**
	Company:

```
                ; REF. NO. BB12
                ; PROGRAM TITLE BCD TO/FROM BINARY CONVERSION
                ;
                ;
                ;
                ; BCD TO BINARY ROUTINE
                ;
                ; REGISTER USAGE
                ; DE - ACCUMULATED BINARY VALUE , SHOULD BE INITIALIZED TO 0
                ; HL - POINTER TO BCD STRING IN MEMORY
                ; A,B,C - USED AS TEMPORARIES
                ;
                BCDBIN:
0000 42                 MOV B,D         ;SAVE INITIAL VALUE OF ACCUMULATOR
0001 4B                 MOV C,E
0002 CD1900             CALL DOUBLE     ;DE*2 -> DE
0005 CD1900             CALL DOUBLE     ; *4
0008 7B                 MOV A,E
0009 81                 ADD C
000A 5F                 MOV E,A
000B 7A                 MOV A,D
000C 88                 ADC B
000D 57                 MOV D,A         ; *5
000E CD1900             CALL DOUBLE     ; *10
0011 77                 MOV M,A         ;GET NEXT DIGIT
                        SUI 30H         ;FOR ASCII CONVERSION, REMOVE COMMENT SE
0012 83                 ADD E
0013 5F                 MOV E,A
0014 7A                 MOV A,D
0015 CE00               ACI 0
0017 57                 MOV D,A
0018 C9                 RET             ;ADD IN NEW DIGIT AND RETURN

                DOUBLE:
0019 7B                 MOV A,E
001A 87                 ADD A
001B 5F                 MOV E,A
001C 7A                 MOV A,D
001D 8F                 ADC A
001E 57                 MOV D,A
001F C9                 RET
0000                    END
```

```
                ; BINARY TO BCD ROUTINE
                ;
                ; REGISTER USAGE
                ; DE - BINARY VALUE
                ; HL - POINTER TO BCD BUFFER IN MEMORY
                ; A,B,C - USED AS TEMPORARIES
                ;
                BINBCD:
0000 011027             LXI B,10000
0003 CD1F00             CALL DIGIT
0006 01E803             LXI B,1000
0009 CD1F00             CALL DIGIT
000C 01E203             LXI B,1000
000F CD1F00             CALL DIGIT
0012 010A00             LXI B,10
0015 CD1F00             CALL DIGIT
0018 010100             LXI B,1
001B CD1F00             CALL DIGIT
001E C9                 RET

                DIGIT:
001F 3600               MVI M,0         ;INITIALIZE DIGIT
                                        ;FOR BCD, DIGIT=0,
                                        ;FOR ASCII, DIGIT=30H
                DI0:                    ;SUBTRACT LOOP
0021 7B                 MOV A,E
0022 91                 SUB C
0023 5F                 MOV E,A
0024 7A                 MOV A,D
0025 98                 SBB B
0026 57                 MOV D,A
0027 FA3100             JM DI1
002A 7E                 MOV A,M         ;INCREMENT BCD DIGIT
002B C601               ADI 1
002D 77                 MOV M,A
002E C32100             JMP DI0
                DI1:
0031 7B                 MOV A,E         ;ADJUST ACCUMULATOR FOR NEXT SEQUENCE
0032 81                 ADD C
0033 5F                 MOV E,A
0034 7A                 MOV A,D
0035 88                 ADC B
0036 57                 MOV D,A
0037 CD3B00             CALL INCHL      ; HL + 1 -> HL
003A C9                 RET

                INCHL:
003B 2C                 INR L
```

```
003C C0             RNZ
003D 24             INR H
003E C9             RET
0000                END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB14

☐ 4004 ☐ 8008 ☒ 8080 ☐ 4040 (use additional sheets if necessary)

Program Title

GRAY TO BINARY CONVERSION

Function

CONVERTS UP TO 16 BITS OF CYCLIN GRAY INTO BINARY DATA.
USED IN READING ENCODER OUTPUT WORDS.

Required Hardware

1 ROM
1 CPU

Required Software

NONE

Input Parameters

GRAY DATA IN D-E REGISTER, E_0 = LSB, UNUSED BITS = 0

Output Results

BINARY DATA IN D-E REGISTER, E_0 = LSB

Registers Modified: A,B,C,	**Maximum Subroutine Nesting Level:**
RAM Required: NONE	**Assembler/Compiler Used:** ISIS 8080 Macro Assembler, V1.0
ROM Required: 20 BYTES	**Programmer:** G. Mercola Data Works, Inc.
	Company: 18725 Bryant St. Northridge, Ca. 91324

ALGORITHM:

$$N_B = ((N_G \veebar \frac{N_G}{2}) \veebar \frac{N_G}{4}) \frac{N_G}{8} \text{ ----}$$

Until $N_G/_2n = 0$

Conversion Time = 36 D usec.

(at 2 usec/instruction cycle)

D = number of bits to the right of the most significant "1" in the Gray code input, +1.

Example

	Gray (N_G)	Binary (N_B)
15	1000	1111
14	1001	1110
13	1011	1101
12	1010	1100
11	1110	1011
10	1111	1010
9	1101	1001
8	1100	1000
7	0100	0111
6	0101	0110
5	0111	0101
4	0110	0100
3	0010	0011
2	0011	0010
1	0001	0001
0	0000	0000

```
                ;REF. NO.  BB14
                ;PROGRAM TITLE GRAY TO BINARY CONVERSION
                ;
                ;
                ;
                ;
                ;
                ; BINARY TO GRAY CODE CONVERSION (UP TO 16 BIT WORD)
                ;
                ;    1. LOAD GRAY WORD INTO D=E REGISTERS.
                ;       (RIGHT JUSTIFIED, UNUSED BITS=0)
                ;    2. CALL SUBROUTINE
                ;    3. ON RETURN, BINARY WORD IS IN D=E REGISTERS.



1000                    ORG     1000H
1000 CD0310             CALL    GRAY
1003 42         GRAY:   MOV     B,D     ;LOAD D TO B
1004 4B                 MOV     C,E     ;LOAD E TO C

1005 78         LOOP:   MOV     A,B     ;LOAD B TO A
1006 B7                 ORA     A       ;OR A WITH A
1007 1F                 RAR             ;SHIFT RIGHT
1008 47                 MOV     B,A     ;LOAD A TO B

1009 79                 MOV     A,C     ;LOAD C TO A
100A 1F                 RAR             ;SHIFT RIGHT
100B 4F                 MOV     C,A     ;LOAD A TO C

100C B0                 ORA     B       ;A OR B TO A
100D C8                 RZ              ;RETURN IF ZERO

100E 79                 MOV     A,C     ;LOAD C TO A
100F AB                 XRA     E       ;EXCL. OR A WITH E
1010 5F                 MOV     E,A     ;LOAD A TO E

1011 78                 MOV     A,B     ;LOAD B TO A
1012 AA                 XRA     D       ;EXCL. OR A WITH D
1013 57                 MOV     D,A     ;LOAD A TO D

1014 C30510             JMP     LOOP
1017 76                 HLT
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BA1

☐ 4004 ☒ 8008 ☐ 8080 ☐ 4040 (use additional sheets if necessary)

Program Title

BCD SUM FOR 8008

Function

PERFORMS THE SUM OF TWO BCD NUMBERS FOR THE 8008 MICROCOMPUTER. THE NUMBER OF DECADES IS A COMPILATION PARAMETER.

Required Hardware

MCS8 MICROCPT.

Required Software

PLM 8 COMPILER.

Input Parameters

THE TWO BCD NUMBERS ARE LOCATED IN MEMORY WITH DECADES ORDINATED IN ASCENDING ORDER IN CONTIGUOUS POSITIONS FROM NUM(0) TO NUM(N), ONE DECADE FOR EACH MEMORY POSITION. NOTE: "N" ALSO REPRESENTS THE POWER OF 10 OF THAT DECADE. THE ADDRESS OF THE TWO LEAST SIGNIFICANT DECADES OF THE TWO NUMBERS ARE SENT TO THE PROCEDURE:
PROCEDURE CALL = SUM(.ADD1,.ADD2)

Output Results

THE SUM OF THE TWO NUMBERS IS IN THE MEMORY POSITION ADD1.
NOTE: ADD1 IS THE FIRST PARAMETER PASSED TO THE PROCEDURE.
THE PROCEDURE RETURNS 0 FOR NO OVERFLOW OR 1 FOR OVERFLOW FROM THE MOST SIGNIFICANT DECADE.

Registers Modified: N/A	**Maximum Subroutine Nesting Level:** 0
RAM Required: 8 + 2n (number of decades. These are the positions of the 2 numbers.	**Assembler/Compiler Used:** PLM 8
ROM Required: 208	**Programmer:** E. Massetti Lab: Massetti
	Company: 20133 Milano via Ronchi 17, Italy

```
00001  1
00002  1   /*REF. NO. BA1 */
00003  1   /*PROGRAM TITLE PROCEDURE  BCD SUM FOR 8008  */
00004  1   /*
00005  1   /*
0   6  1   /* PROCEDURE FOR SUMMING TWO BCD NUMBER,  CIFRE  DECADES EACH.
00007  1   THE TWO ADDRESS OF THE FIRST DECADE OF EACH NUMBER, .ADD1 AND ADD2, ARE
00008  1   SENT TO THE PROCEDURE.
00009  1   THE TWO BCD NUMBER IN THE MEMORY ONE BCD DECADE IN EACH
00010  1   MEMORY POSITION ORDERED IN ASCENDING ORDER FROM NUM(0) TO
00011  1   NUM(N), WHERE N RAPRENSENTS ALSO THE POWER OF 10 OF THAT DECADE.
00012  1   RESULT IS SENT TO THE MEMORY POSITIONS OF THE FIRST ADDEND SENT
00013  1   TO THE PROCEDURE. */
00014  1
00015  1   DECLARE CIFRE LITERALLY '8'; /* THIS IS A COMPILER PARAMETER
00016  1                                   AND IS THE NUMBER OF DECADES FOR
00017  1                                   THAT COMPILATION. */
00018  1
00019  1   SUM : PROCEDURE (INDADD1,INDADD2) BYTE;
00020  2   DECLARE (INDADD1,INDADD2) ADDRESS;
00021  2   DECLARE ADD1 BASED INDADD1 BYTE,
00022  2           ADD2 BASED INDADD2 BYTE,
00023  2           (RIPORTO,N) BYTE;
00024  2   RIPORTO =0;
00025  2   DO N=0 TO (CIFRE-1);
00026  2           ADD1 (N)=(ADD1(N)+ADD2(N)+RIPORTO)AND 0FH;
00027  3   /* AND NOW BCD CORRECTION ROUTINE*/
00028  3           IF ADD1(N) >9 THEN DO;
00029  3                 ADD1 (N)=ADD1 (N)+6;
00030  4                 RIPORTO =1;
00031  4                 END;
00032  3           ELSE RIPORTO =0;
00033  3           END;
00034  2   RETURN RIPORTO;
00035  2   END SUM;
00036  1   /*END OF PROCEDURE. TO TEST IT THERE IS NOW A SAMPLE CALL OF SUM
00037  1   WITH THE NUMBERS TOTAL AND PAR, THE RESULT IS IN TOTAL*/
00038  1   VERIFY: DECLARE TOTAL (CIFRE) BYTE,
00039  1                   NOTOK BYTE,
00040  1                   PAR(CIFRE)BYTE;
00041  1   NOTOK=SUM(.TOTAL,.PAR);
00042  1   EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB15

☐ 4004 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

LOG2A

Function

To calculate the Base 2 Log of a number between 1 and 255.

Required Hardware

None

Required Software

None

Input Parameters

Value of number in A

Output Results

$LOG2(A) = B + \frac{C}{256}$ where B = characteristic and $\frac{C}{256}$ = Mantissa. If % error is defined as $100 \frac{A - 2^{B + C/256}}{A}$, then the maximum error is .311% which occurs for A = 133 (decimal).

Registers Modified:	Maximum Subroutine Nesting Level:
A,B,C,D,H&L	0
RAM Required:	**Assembler/Compiler Used:**
None	MAC8
ROM Required:	**Programmer:** Bill Baker Bell Telephone Labs
64 bytes	
	Company: Room 1G625 Holmdel, N.J. 07733

Explanation of LOG2A

The program calculates the LOG2(A) for any integer between 1 and 255 in the following manner:

Any integer A between 1 and 255 can be expressed in the form:

$A = 2^B * (1 + \frac{I}{256})$ where B and I are integers and

$0 \leq B \leq 7$ and $0 \leq I < 255$

The expression for LOG2(A) in terms of B and I is:

$$LOG2(A) = B + LOG2\ (1 + \frac{I}{256})$$

where B is the characteristic and $LOG2\ (1 + \frac{I}{256})$ is the mantissa.

B is determined by left shifting the accumulator (which contains A) until a carry bit is generated and subtracting the number of shifts from 8. The number remaining in the accumulator is I.

To determine the mantissa, I is expressed in the form:

$$I = 16 \cdot C_0 + C_1$$

The nearest integer values of

$$D(C_0) = 256 \quad LOG2\ (1 + \frac{16 \cdot C_0}{256})$$

for C_0 from 0 to 15 are stored in the program.

C_0 is used to look up the first approximation of

$$256 \cdot LOG2\ (1 + \frac{I}{256})$$

C_1 is used to interpolate between $D(C_0)$ and $D(C_0 + 1)$.

The maximum error between the log determined by this program and the true log (rounded to 8 bits) is 1/256, or a count of 1 in C. If this were the only source of error, the maximum error in the resulting antilog would be .25%. However, there is an additional error which results from rounding off the mantissa to an accuracy of 8 bits. Thus, the maximum error encountered in using this program is greater than .25%. The maximum error occurs for A = 133 and is .311%.

```
                ;REF. NO. BB15
                ;PROGRAM TITLE LOG 2A
                ;
                ;
0900                    ORG     44000Q
0031            LLL     EQU     0610
                ;LOG2(A) RESULT IN B,C
                LOG2A:
0900 0608               MVI     B,8
0902 A7                 ANA     A       ;SET CARRY TO 0
                LO1:
0903 05                 DCR     B       ;SHIFT MSB TO
0904 17                 RAL             ;CARRY POSITION
0905 D20309             JNC     LO1
0908 07                 RLC             ;A=16*C0+C1
0909 07                 RLC
090A 07                 RLC
090B 07                 RLC
090C 57                 MOV D,A         ;C=16*C1+C0
090D E60F               ANI     0170
  0F C631               ADI     LLL
  11 6F                 MOV     L,A
0912 2609               MVI     H,0110  ;=H VALUE OF D TABLE
0914 7E                 MOV     A,M
0915 2D                 DCR     L
0916 96                 SUB     M       ;A=D(C0+1)-D(C0)
0917 57                 MOV     D,A     ;D=D(C0+1)-D(C0)
0918 6E                 MOV     L,M     ;L=D(C0)
0919 79                 MOV     A,C
091A E6F0               ANI     3600    ;A=16*C0;CY=0
091C 2603               MVI     H,3     ;3 SINCE LSB OF C0 IS 0
                LO2:
091E 17                 RAL
091F D22409             JNC     LO3
0922 82                 ADD     D
0923 A7                 ANA     A
                LO3:
0924 25                 DCR     H
0925 C21E09             JNZ     LO2
0928 0F                 RRC             ;A=C0*[D(C0+1)-D(C0)]/2
0929 0F                 RRC
092A 0F                 RRC
092B E61F               ANI     0370    ;A=C0*[D(C0+1)-D(C0)]/16
092D 85                 ADD     L
092E 4F                 MOV     C,A
  2F C9                 RET
                DATA0:
0930 00162C40           DB      000,022,044,064
0934 53657686           DB      083,101,118,134
```

```
0938 96A5B3C1           DB      150,165,179,193
093C CFDCE8F4           DB      207,220,232,244
                        ;
                        ;
                        ;
                        ;TEST PROGRAM FOR LOG2A
                        ;FOR PROPER OPERATION
                        ;OUTPUT PORT 0 GOES TO
                        ;ALL ONES AND PROCESSOR
                        ;GOES INTO HALT STATE.
                        ;FOR IMPROPER OPERATION,
                        ;OUTPUT PORT 0 GOES TO ALL
                        ;ZEROES AND PROCESSOR
                        ;GOES INTO HALT STATE
                        ;
0940                    ORG     4500Q
0940 3E00               MVI     A,0
0942 D308               OUT     10Q
0944 3E08               MVI     A,8
0946 CD0009             CALL    LOG2A
0949 78                 MOV     A,B
094A FE03               CPI     3
094C C26A09             JNZ     ERROR
094F 79                 MOV     A,C
0950 FE00               CPI     0
0952 C26A09             JNZ     ERROR
0955 3E3C               MVI     A,74Q
0957 CD0009             CALL    LOG2A
095A 78                 MOV     A,B
095B FE06               CPI     6
095D C26A09             JNZ     ERROR
0960 79                 MOV     A,C
0961 FEE8               CPI     350Q
0963 C26A09             JNZ     ERROR
0966 3EFF               MVI     A,377Q
0968 D308               OUT     10Q
096A 76         ERROR:  HLT
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB16

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

DIGITAL TO ANALOG CONVERSION FOR EIGHT OUTPUTS

Function

The program processes a list of eight 16 bit values to generate eight pulse width modulated voltages, which can be filtered to provide inexpensive digital to analog conversion useful for process control or other low speed requirements. See the attached explanation of the program's operation.

Required Hardware

A clock interrupt generator (line frequency or faster) and an eight bit latch (Intel 8212) at output address 1, with voltage level conversion and filtering as required.

Required Software

To be useful, something should set the output values in the table.

Input Parameters

A table of eight double byte values and remainders is required, in the following order:

low order value #1
high order value #1
low order remainder #1
high order remainder #1
low order value #2
------
high order remainder #8

Output Results

Eight analog output voltages are produced at the filter outputs.

Registers Modified: A, B, D, E, H, L SP is used and restored.	Assembler/Compiler Used: 8080 MACRO ASSEMBLER, VER. 2.2
RAM Required: 34 bytes	Programmer: W. M. Hawkins
ROM Required: 32 bytes	Company: Hercules Inc., Res. Ctr.
Maximum Subroutine Nesting Level: none	Address: Wilmington, De, 19899

Digital to Analog Conversion for Eight Outputs

W. M. Hawkins 4/18/75

The program produces a PWM output by adding the present desired output value to the past remainder at a rate determined by the clock interrupt that starts it. When a carry results from the addition, the output is turned on. When no carry occurs, the output is turned off.

The desired output value in the table is a binary fraction of full output, so that a value of 0000 produces no output and a value of FFFFH produces full output. If the value is 8000H, a carry is generated on every other pass which produces a square wave at half the clock frequency whose average value is exactly half of full output.

The PWM output can be filtered with a simple R and C combination to obtain a smooth analog output voltage. The demands on the filter become greater as the value moves away from 8000H. At 5% output the carry occurs only once in every twenty clock interrupts.

The stack pointer and register pairs are used to process the list because time is essential. Execution time (and fraction of system loading) determines the upper interrupt rate limit. The lower limit is set by the desired filter time constant. An 8080 with fast memory can execute the program in about 330 microseconds.

Please call me at 302-995-3562 if further explanation would help.

```
                        ;REF.  NO.  BB16
                        ;PROGRAM NAME DIGITAL TO ANALOG CONVERSION FOR EIGHT OUTPUTS
                        ;
                        ;
                        ;
                        ;
                        ;DIGITAL TO ANALOG CONVERSION FOR EIGHT OUTPUTS
                        ;
                        ;W.  M.  HAWKINS    4/9/75
                        ;
                        ;CONVERT 8 WORDS IN "TABLE" TO 8 BITS IN A LATCH
                        ;AT A RATE THAT CAN BE FILTERED TO GIVE AN
                        ;ACCEPTABLE ANALOG OUTPUT WITH 16 BIT RESOLUTION.
                        ;
0000                            ORG     0
0001                    DAC     EQU     1           ;THE 8212 LATCH ADDRESS
                        ;
0000 F3                 DAC8:   DI                  ;STOP USING THE SP
0001 210000                     LXI     H,0000H     ;ZERO H AN D L
0004 7C                         MOV     A,H         ;ZERO A
0005 39                         DAD     SP          ;MOVE SP TO HL
0006 220010                     SHLD    SAVSP       ;SAVE THE POINTER
0009 0608                       MVI     B,8         ;PROCESS 8 VALUES
000B 310210                     LXI     SP,TABLE            ;POINT TO DATA
                        ;
                        ;
000E D1                 LOOP:   POP     D           ;LOAD DE WITH A VALUE
000F E1                         POP     H           ;LPOAD HL WITH REMAINDER
0010 19                         DAD     D           ;ADD VALUE TO REMAINDER
0011 1F                         RAR                 ;PUT C BIT INTO A
0012 E5                         PUSH    H           ;NEW REMAINDER
0013 33                         INX     SP          ;PASS REMAINDER
0014 33                         INX     SP          ;(SAVE 8 CYCLES)
0015 05                         DCR     B           ;COUNT PASSES
0016 C20E00                     JNZ     LOOP        ;GO UNTIL DONE
                        ;
                        ;
0019 D301                       OUT     DAC         ;SEND BITS TO LATCH
001B 2A0010                     LHLD    SAVSP       ;GET POINTER TO HL
001E F9                         SPHL                ;RESTORE SP
001F FB                         EI                  ;ENABLE , IF REQUIRED
0020 76                         HLT                 ;FOR TESTING      ONLY *
                        ;
                        ;SET UP THE RAM STORAGE
                        ;
1000                            ORG     1000H       ;OR WHEREVER RAM IS AT
                        ;
1000 0000               SAVSP:  DW      0           ;PLACE TO PUT SP
1002                    TABLE.  DS      32          ;8 VALUE ,REMAINDER PAIRS
```

```
                        ;
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB17

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Binary to HEX Routine

Function

To read a paper tape in binary (EBCDIC) format from the Intel HSPTR to the MCS-80 System

Required Hardware

TTY on port 0 and 1
HSPTR on port 1 and 3

Required Software

Input Parameters

Output Results

SEE WRITEUP

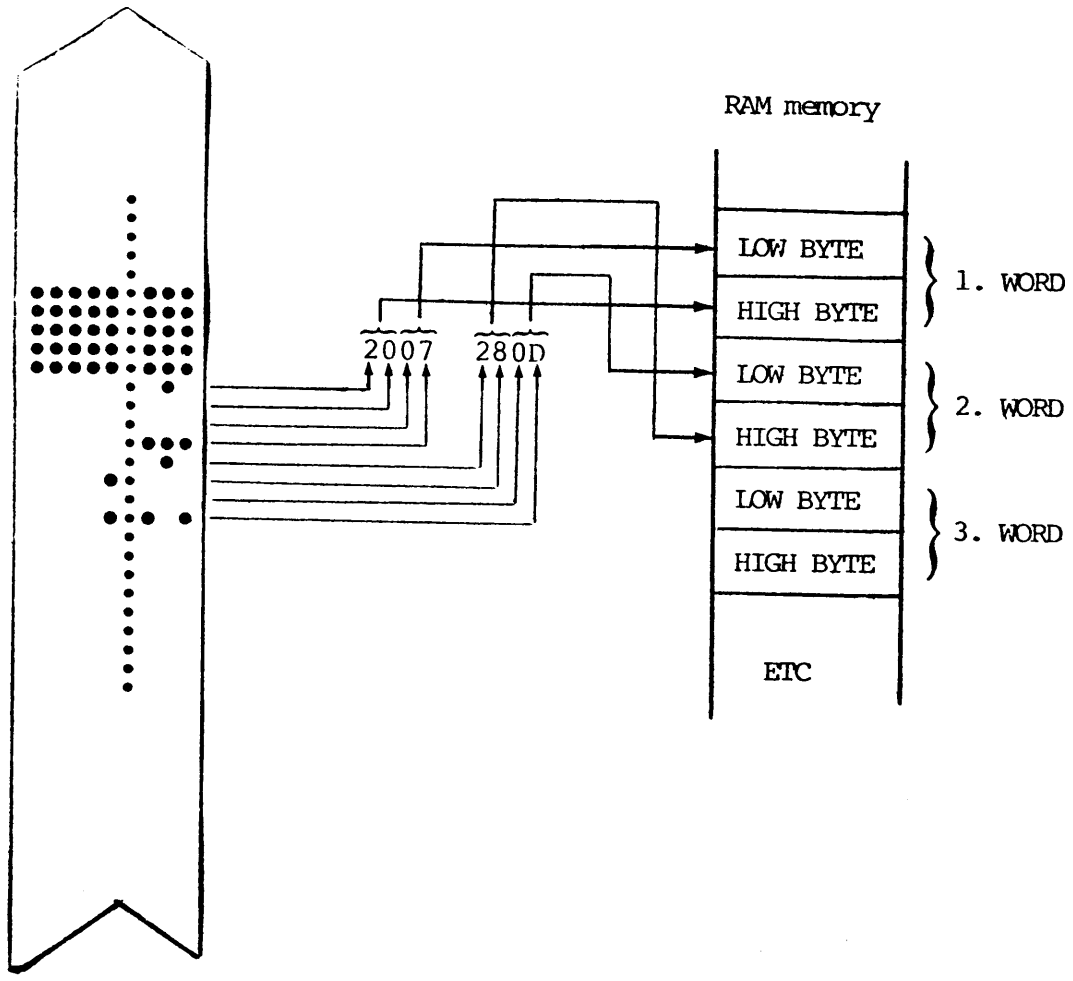
Registers Modified:	**Assembler/Compiler Used:**
RAM Required:	**Programmer:** Sigmund Hjerde
ROM Required:	**Company:** National Institute of Technology
Maximum Subroutine Nesting Level:	**Address:** Akerson. 24C OSLO 1, Norway

MICROCOPUTER USER'S LIBRARY SUMITTAL FORM

8080 Microprosessor.

PROGRAM TITLE: BINARY to HEX Routine.

The program perform loading of paper-tape in binary format (EBCDIC) from HSPTR (INTEL) to the MCS-80 system as shown in figure below.



After data is loaded in RAM the program will exchange the contents of high byte and low byte for each 16 bit word in memory.
The program is able to read an exhange 2 K bytes.

FUNCTION: The program store data from papertape in memory from location 1000 to 17 FF.

To execute the READ routine of one character from papertape, a CALL to subroutine in monitor location 3EAO is used.

When the first character (4-bits) is read to accumulater a RLC moves the contents four times left and store it in D-register.

Next character is read to accumulator and a XRA D command masks the content of D-register to the content of accumulator.

8-bit data in accumulator (two characters from papertape) is moved to memory location addressed by the contents of registers H and L.

A CALL to subroutine TEST is a test on memory location 17 FF. When 17 FF is reached the 2 K byte of data is read from papertape and the program is ready to exhange the content of location 1000 with 1001, 1002 with 1003 ..... and so on.

Required handware:

TTY on port 0 and 1
HSPTR on port 1 and 3

Programmer: Sigmund Hjerde

Company: National Institute of Technology
Akersvn. 24 C,
OSLO 1
NORWAY

```
                ;REF. NO. BB17.
                ;PROGRAM TITLE BINARY TO HEX ROUTINE
                ;
                ;
                ;
                ;
                ;
1E00                    ORG     1E00H   ; BIN - HEX (HSPTR)
1E00 210010             LXI     H,1000H ; DATA ADDRESS FROM HSPTR
1E03 0602       NEW:    MVI     B,02    ; CHARACTER LOOP COUNTER
1E05 1E04               MVI     E,04    ; BIT LOOP COUNTER
1E07 CDA03E     READ:   CALL    3EA0H   ; RI SUBROUTINE IN MONITOR
1E0A 05                 DCR     B       ; CHARACTER LOOP
1E0B CA171E             JZ      ACC     ; TWO CHARACTERS IS READ
1E0E 07         ROTA:   RLC             ; ROTATE LEFT
1E0F 1D                 DCR     E       ; BIT LOOP
1E10 C20E1E             JNZ     ROTA    ; CONTENT IN ACC MOVED LEFT 4 TIMES
1E13 57                 MOV     D,A     ; SAVE ACC IN D-REG
1E14 C3071E             JMP     READ    ; READ NEXT CHARACTER
1E17 AA         ACC:    XRA     D       ; MASK ACC AND D-REG IN ACC
 E18 77                 MOV     M,A     ; 8-BIT DATA TO MEMORY
1E19 23                 INX     H       ; NEXT ADDRESS IN MEMORY
1E1A CD321E             CALL    TEST    ; CHECK SIZE OF MEMORY, STOP AT LOC
1E1D D2031E             JNC     NEW     ; READ NEXT TWO CHARACTERS
1E20 210010             LXI     H,1000H ; START OF "EXCHANGE HIGH-LOW BYTE"
1E23 56         VEND:   MOV     D,M     ; FIRST LOC TO D-REG
1E24 23                 INX     H
1E25 5E                 MOV     E,M     ; NEXT LOC TO E-REG
1E26 2B                 DCX     H
1E27 73                 MOV     M,E     ; E-REG TO FIRST LOC
1E28 23                 INX     H
1E29 72                 MOV     M,D     ; D-REG TO NEXT LOC
1E2A CD321E             CALL    TEST
1E2D 23                 INX     H
1E2E D2231E             JNC     VEND    ; CONTINUE
1E31 C7                 RST     0
1E32 E5         TEST:   PUSH    H       ; SUBROUTINE FOR CHECK SIZE IN MEM
1E33 013317             LXI     B,5939D ; ABLE TO GEN CARRY AT LOC 17FF
1E36 09                 DAD     B       ; CARRY GENERATED ?
1E37 E1                 POP     H
1E38 C9                 RET
0000                    END
```

Ref.#BB18

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

BINARY TO BCD SUBROUTINE

Function

Converts unsigned binary number in D, E to 5 BCD digits

Required Hardware

None

Required Software

None

Input Parameters

D,E contain binary value
H,L pointer to output buffer

Output Results

5 BCD digits in buffer
H,L contain address of MSD

Registers Modified: None	Programmer: Niels S. Gundestrup
RAM Required: None	Company: Geophysical Isotope Lab.
ROM Required: 52_{10} bytes	Address: Haraldsgade 6
Maximum Subroutine Nesting Level: N.A.	City: DK-2200 Cph N
Assembler/Compiler Used: MAC-80 Vers 3.0	State: DENMARK

```
                ;REF. NO. BB18
                ;PROGRAM TITLE BINARY TO BCD SUBROUTINE
                ;
                ;
                ;BINARY TO BCD SUBROUTINE

                ;INPUT: UNSIGNED BINARY NUMBER IN D,E
                ;       POINTER TO LOWEST BUFFER LOC IN HL

                ;OUTPUT: 5 BCD-DIGITS, ONE DIGIT PER MEMORY LOC.
                ;       HL POINT TO MSD IN LOWEST LOCATION.

0000 F5         BNBCD:  PUSH PSW        ;SAVE VARIABLES
0001 C5                 PUSH B
0002 D5                 PUSH D
0003 E5                 PUSH H
0004 EB                 XCHG            ;GET NUMBER IN HL, ADDR IN DE
  05 01F0D8             LXI B,-10000
0008 CD2400             CALL DECNO      ;GET MSD
000B 0118FC             LXI B,-1000
000E CD2400             CALL DECNO
0011 019CFF             LXI B,-100
0014 CD2400             CALL DECNO
0017 01F6FF             LXI B,-10
001A CD2400             CALL DECNO
001D 7D                 MOV A,L         ;GET LSD
001E 12                 STAX D          ;STORE IT
001F E1                 POP H
0020 D1                 POP D
0021 C1                 POP B
0022 F1                 POP PSW
0023 C9                 RET

0024 AF         DECNO:  XRA A           ;0 TO A. USE 30H IF ASCII
0025 D5                 PUSH D          ;SAVE ADDR
0026 5D                 MOV E,L         ;SAVE BINARY
0027 54                 MOV D,H
0028 3C                 INR A           ;INCREMENT DIGIT
0029 09                 DAD B           ;SUBTRACT
002A DA2600             JC DECNO+2      ;RESULT NEGATIVE?
002D 3D                 DCR A           ;YES, RESTORE DIGIT COUNT
002E 6B                 MOV L,E         ;BINARY NUMBER
002F 62                 MOV H,D
0030 D1                 POP D           ;AND ADDRESS
0031 12                 STAX D          ;STORE DIGIT
0032 13                 INX D           ;INCREMENT POINTER
```

```
0033 C9                 RET
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BA2

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Hex to Decimal Conversion

Function

Converts any hex number between 0 and FFFFH to the decimal equivalent

Required Hardware

TTY
H.S. Reader
Intellec 8

Required Software

Program Object Tape

Input Parameters

Load program with system monitor. Enter hex number on TTY followed by a CR and LF.

Output Results

The decimal equivalent is printed on the following line and operation is returned to the system monitor.

Registers Modified:	**Assembler/Compiler Used:** Intellec 8 Macro Assembler
RAM Required:	**Programmer:** Jon Zoller
ROM Required:	**Company:** Dorsett Electronics
Maximum Subroutine Nesting Level:	**Address:** Box 36 Tulsa, Ok 74101

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB19

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

BCD Input and Direct Conversion to Binary Routine

Function

Fast and efficient BCD to Binary conversion code. Presented in (pseudo) subroutine form for implementation in ROM to allow reading of BCD input value, conversion to binary representation and branching based on loading H & L Registers to PC.

Required Hardware

An Input Port for BCD parameter.

Required Software

None, except for user-written main-line assembly coding.

Input Parameters

BCD value to be read from user-designated Input Port, or BCD value can be brought into subroutine instead, via A-Register, for example.

Output Results

Binary representation is in A-Register upon completion of execution. Two other registers, C and E (or other user-designated registers or memory locations depending upon use of ROM or RAM) are used and original contents are altered.

Registers Modified: 3, including A-Register	**Assembler/Compiler Used:** Intellec 8 Macro Assembler
RAM Required: None	**Programmer:** M.H. Gansler
ROM Required: 18 Bytes	**Company:** IGM/NTI
Maximum Subroutine Nesting Level:	**Address:** P.O.Box 3950 Bellingham, WA 98225

```
                        ;REF. NO. BB19
                        ;PROGRAM TITLE BCD INPUT & DIRECT CONVERSION TO BINARY
                        ;
                        ;
                        ;
                        ;
                        ; BCD INPUT AND DIRECT CONVERSION TO BINARY
                        ;
                        ; (PSEUDO) SUBROUTINE  BCDBN    --  INPUT &
                        ;
                        ; CONVERT  BCD VALUE TO BINARY REPRESENTATION
                        ;
0000 DB04       BCDBN:  IN      4       ; READ BCD VALUE
0002 4F                 MOV     C,A     ; STORE IN C. REGISTER
0003 E60F               ANI     0FH     ; MASK FOR UNIT'S DIGIT
0005 5F                 MOV     E,A     ; STORE IN E. REGISTER
0006 79                 MOV     A,C     ; GET BCD VALUE BACK
0007 E6F0               ANI     0F0H    ; MASK FOR TEN'S DIGIT
0009 0F                 RRC             ;
000A 0F                 RRC             ; ROTATE TWICE RIGHT
000B 4F                 MOV     C,A     ; RETAIN IN C. REGISTER
000C 0F                 RRC             ;
000D 0F                 RRC             ; ROTATE TWICE RIGHT AGAIN
000E 81                 ADD     C       ; ADD C. REGISTER CONTENTS
000F 07                 RLC             ; ROTATE ONE LEFT
0010 83                 ADD     E       ; ADD UNIT'S DIGIT
0011 E9                 PCHL            ; AND RETURN WITH BINARY
                                        ; REPRESENTATION IN
                                        ; A. REGISTER

0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BA4

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

HEX TO/FROM BCD

Function

Converts hexadecimal numbers input on TTY to decimal numbers and vice versa. Decimal numbers must be ended with D, hexadecimals with H. Conversion begins with space. If first char input is CR, control is given back to monitor. Largest number handled is two bytes binary.

Required Hardware

Intellec 8/MOD 80, TTY

Required Software

Intellec 8/MOD 80 Monitor, vers. 2.Ø

Input Parameters

Number to be converted is input on TTY ended either with H or D. Conversion begins with space. If first char input is CR, execution is ended. If CR is recognized later, it is interpreted as error.

Output Results

Converted number is printed on the same line as the input number. It errors are detected a! is printed.

Registers Modified: ALL	**Assembler/Compiler Used:** 8080 Macro Assembler, Vers. 3.0
RAM Required: 14 (for stack)	**Programmer:** Markus Warsta
ROM Required: 415	**Company:** TELEVA
Maximum Subroutine Nesting Level: 3	**Address:** Takkatie 7a SF-00370 Helsinki 37 Finland

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. BC3

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

FIXED AND FLOATING POINT ARITHMETIC ROUTINES - MATH

Function

Includes routines for fixed and floating point arithmetic together with a demonstration program that performs algebraic evaluation (from left to right, with no operator precedence) and allows unlimited parentheses nesting. An expression within parentheses can be evaluated and displayed by "=", and is preserved as a subtotal, etc.

Required Hardware

On input, space, rubout and all control characters are echoed and ignored. Allowable printing characters are:

0 through 9 and -
* / + -
( ) =
P i.e. the value 3.1416
E i.e. the value 2.7183 or exponent entry (context sensitive)

Required Software

Output uses exponential notation as required and preserves 4 decimal digits. Input may use exponential notation.

Input Parameters

Numbers can be expressed in the approximate range ± 0.7 * 10 - 39 to ± 0.8 * 10 38, and zero.

All code is re-entrant and recursive techniques are used. All memory is dynamically allocated and deallocated.

An example of program operation is attached.

Test system is entered @ 2000H.

Output Results

Program offered on diskette only.

Registers Modified: See comments on routines	**Assembler/Compiler Used:** Intel V3.0 modified
RAM Required: Dynamic allocation, based on stack pointer. Most usage requires less than 100 bytes.	**Programmer:** Charles B. Falconer
ROM Required: 2000-26DE HEX assembled plus monitor console I/O	**Company:** Yale University
Maximum Subroutine Nesting Level: No intrinsic limit. See comments.	**Address:** Room 6016 CB, Yale-New Haven Hospital, 789 Howard Ave, New Haven 06504

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC4

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

FLOATING POINT ELEMENTARY FUNCTION PACKAGE

Function

CALCULATES FLOATING POINT
- SQUARE ROOT
- LOGARITHM
- EXPONENTIAL FUNCTION
- SINE
- COSINE
- ARC TANGENT
- HYPERBOLIC SINE
- HYPERBOLIC COSINE

Required Hardware

NONE

Required Software

FLOATING POINT PACKAGE, INCLUDING FIX AND FLT ROUTINES - BC1

Input Parameters

FURNISHED IN FLOATING POINT ACCUMULATOR -
SEE ATTACHED DOCUMENTATION

Output Results

FURNISHED IN FLOATING POINT ACCUMULATOR -
SEE ATTACHED DOCUMENTATION

Registers Modified: ALL, SEE ATTACHED DOCUMENTATION	**Assembler/Compiler Used:** 8080 MACRO ASSEMBLER VER 2.2
RAM Required: 24 BYTES IN FLOATING POINT SCRATCH BANK	**Programmer:** O.C. JUELICH
ROM Required: UP TO 865 BYTES FOR ENTIRE PACKAGE	**Company:** MISSILE SYSTEMS DIVISION ROCKWELL INTERNATIONAL CORP. 165-796BG
Maximum Subroutine Nesting Level: 6 (incl. misc. uses of stack) plus Floating Point Package	**Address:** 4300 E. FIFTH AVENUE COLUMBUS, OHIO 43216

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC5

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

FLOATING POINT PACKAGE WITH BCD CONVERSION ROUTINE FOR 8080

Function

Performs floating addition, subtraction, multiplication, division, fixing, floating, negation, and conversion from floating point to BCD with exponent. (Details at beginning of program).

Required Hardware

8080 Microcomputer system with 767 locations free in ROM and 21 locations free in RAM. BCD output of FBCD routine can be arranged and output to, eg., a printer, if desired.

Required Software

No other software required except the calling program.

Input Parameters

For FADD, FSUB, FDIV, and FMPY, the numbers to be operated upon are loaded into C-D-E and B-H-L. For FIXX, FNEG, and FBCD, the number to be operated upon is loaded into C-D-E. For FLOT, the number to be floated is loaded into D-E. (See details at the beginning of the program).

Output Results

Results from FIXX are returned to D-E, all others in C-D-E. (For details, see descriptions at the beginning of the program and at each routine.)

Registers Modified: All registers	**Assembler/Compiler Used:** MACRO (8080)
RAM Required: 21 Locations (9 Variables, 12 locs. for Stack)	**Programmer:** Dr. Keith J. Caserta
ROM Required: 767 Locations (525 without BCD conversion routine)	**Company:** The Procter and Gamble Company
Maximum Subroutine Nesting Level: 3 (4 counting the CALL to these routines)	**Address:** The Ivorydale Technical Center Cincinnati, Ohio 45217

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

LEAST SQUARES QUADRATIC FITTING ROUTINE FOR 8080

Function

Performs summations and matrix manipulation for fitting up to 256 floating point X-Y pairs to a function of the form:

$$aX^2 + bX + c = Y$$

Required Hardware

8080 microcomputer system with 595 locations free in ROM for the FIT routine and 785 locations in ROM for the floating point package. 2338 RAM locations are required for the FIT routine (for a 256 point fit) and 21 locations for the floating point package.

Required Software

Floating Point Package with BCD Conversion Routine for 8080 - BC5

Input Parameters

X values must be contained in a 768 location array, XX1. The following 768 locations are reserved for calculated X^2 values (XS1 array). Floated Y values must be contained in the 768 location YY1 array. The A register must contain the number of points (up to 256) to be fitted.

Output Results

Coefficients a, b, and c are contained in 1134 and the following 2 locations (a), 1124 and the following 2 locations (b), and 1114 and the following 2 locations (c) upon return - in floating point format.

Registers Modified: All registers	Programmer: Dr. Keith J. Caserta
RAM Required: 2359 locations (for a 256 point fit)	Company: The Procter and Gamble Company
ROM Required: 1380 locations (920 without BCD conversion routine)	Address: The Ivorydale Technical Center
Maximum Subroutine Nesting Level: 4 (5 counting call to FIT)	City: Cincinnati
Assembler/Compiler Used: MACRO (8080)	State: Ohio 45217

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC7

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

FLOATING POINT PROCEDURES

Function

DUMMY PL/M INTERFACE PROCEDURES

Required Hardware

NONE

Required Software

PL/M - FLOATING POINT INTERFACE (ASSEMBLER) - BC8

Input Parameters

REVISED 8/8/77

Output Results

NOTE: BC7 is ordered as one program with BC9.

Registers Modified: ALL	**Assembler/Compiler Used:** PL/VER. 3.3
RAM Required: 0	**Programmer:** STOLBERG-ROHR MS E15V22
ROM Required: 124 byte	**Company:** DANFOSS A/S
Maximum Subroutine Nesting Level: SE PROGRAM	**Address:** DK6430 NORDBORG DENMARK

```
PLM1 VERS 3.3
                                    Ref. No. BC7
00001  1    /* STOLBERG-ROHR, EL-LAB, E15-V22, 2876, ISA1560. */
00002  1    /*************************************************************/
00003  1    /*                                                           */
00004  1    /*   FLOATING POINT PROCEDURES .  A1560                      */
00005  1    /*                                                           */
00006  1    /*************************************************************/
00007  1
00008  1    /* INITIALIZE: MOVES A SECTION OF CODE FROM ROM MEMORY TO SCRATCHPAD
00009  1       RAM MEMORY IN PREPARATION FOR EXECUTION OF MULTIPLY AND DIVIDE
00010  1       SUBROUTINES. OVERFLOW FLAG SET TO ZERO. 0 NEST LEVEL          */
00011  1    FINIT: PROCEDURE;
00012  2     GO TO 100H; /* ADDRESS IN PLM-FL.PNT. INTERFACE PROGRAM *
00013  2    END FINIT;
00014  1
00015  1     /* F.P.ADD: ADDS THE SPECIFIED OPERAND TO THE F.P. ACC., RESULT
00016  1       IN TEMP= COPY OF F.P.ACC., RETURN VALUE = ADDR. OF TEMP.
00017  1       3 NEST LEVELS.                                              */
00018  1    FADD: PROCEDURE (OPERAND) ADDRESS;
00019  2          DECLARE OPERAND ADDRESS;
00020  2     GO TO 103H;
00021  2    END FADD;
00022  1
00023  1     /* F.P. SUBTRACT: 3 NEST LEVELS.                               */
00024  1    FSUB: PROCEDURE (OPERAND) ADDRESS;
00025  2          DECLARE OPERAND ADDRESS;
00026  2     GO TO 106H;
00027  2    END FSUB;
00028  1
00029  1     /* F.P. MULTIPLY. 3 NEST LEVELS.    */
```

```
00030  1   FMUL: PROCEDURE (OPERAND) ADDRESS;
00031  2         DECLARE OPERAND ADDRESS;
00032  2    GO TO 109H;
00033  2   END FMUL;
00034  1
00035  1    /* F.P. DIVIDE. 3 NEST LEVELS.    */
00036  1   FDIV: PROCEDURE (OPERAND) ADDRESS;
00037  2          DECLARE OPERAND ADDRESS;
00038  2    GO TO 10CH;
00039  2   END FDIV;
00040  1
00041  1    /* F.P. ABSOLUTE: SETS THE SIGN OF F.P.ACC. POSITIVE.
00042  1       1 NEST LEVEL.  */
00043  1   FABS: PROCEDURE ADDRESS;
00044  2    GO TO 10FH;
00045  2   END FABS;
00046  1
00047  1            /* F.P. ZERO: SETS THE F.P.ACC. = 0. 1 NEST LEVEL   */
00048  1   FZRO: PROCEDURE ADDRESS;
00049  2    GO TO 112H;
00050  2   END FZRO;
00051  1
00052  1    /* F.P. TEST: SETS THE CONTROL FLAGS. 1 NEST LEVEL.  */
00053  1   FTST: PROCEDURE ADDRESS;
00054  2    GO TO 115H;
00055  2   END FTST;
00056  1
00057  1    /* F.P. COMPLEMENT: CHANGE THE SIGN OF THE F.P.ACC.
00058  1       1 NEST LEVEL   */
00059  1   FCHS: PROCEDURE ADDRESS;
```

```
00060  2    GO TO 118H;
00061  2   END FCHS;
00062  1
00063  1            /* F.P. LOAD: LOADS SPEF. OPERAND INTO F.P.ACC.
00064  1      2 NEST LEVELS.  */
00065  1   FLOD: PROCEDURE (OPERAND) ADDRESS;
00066  2         DECLARE OPERAND ADDRESS;
00067  2    GO TO 118H;
00068  2   END FLOD;
00069  1
00070  1            /* F.P. INPUT: CONVERTS CHAR.STRING (MAX. 32 DIG ) IN MEM-
00071  1              ORY, TO F.P. FORMAT IN F.P.ACC.. 4 NEST LEVELS  */
00072  1   FINP: PROCEDURE (STRINGBUFFER) ADDRESS;
00073  2         DECLARE STRINGBUFFER ADDRESS;
00074  2    GO TO 11EH;
00075  2   END FINP;
00076  1
00077  1    /* F.P. OUTPUT: COVERTS VALUE IN ACC. TO 13 DEC DIG. IN
00078  1      OUTBUFFER. 4 NEST LEVELS.  */
00079  1           FOUT: PROCEDURE (OUTBUFFER) ADDRESS;
00080  2         DECLARE OUTBUFFER ADDRESS;
00081  2    GO TO 121H;
00082  2   END FOUT;
00083  1    /* F.P. FLOAT: CONVERTS BINARY VALUE IN 5 BYTE BINSTRING
00084  1      ( 32 BIT + 8 BIT SCALE FACTOR ) TO F.P. FORMAT IN ACC.
00085  1      3 NEST LEVELS.  */
00086  1   FFLT: PROCEDURE (BINSTRING) ADDRESS;
00087  2         DECLARE BINSTRING ADDRESS;
00088  2   GO TO 124H;
00089  2   END FFLT;
```

```
 0090  1
00091  1    /* F.P. FIX: CONVERTS VALUE IN ACC. TO FIX FORMAT
00092  1      ( SCALE FACTOR ), RESULT IN TEMP. 2 NEST LEVELS.  */
00093  1    FFIX: PROCEDURE (SCALE) ADDRESS;
00094  2          DECLARE SCALE BYTE;
00095  2     GO TO 127H;
00096  2    END FFIX;
00097  1
00098  1    /* OVERFLOW, RETURNS THE VALUE OF THE OVERFLOW FLAG,
00099  1      AND CLEAR THE FLAG.  */
00100  1    OVERFLOW: PROCEDURE BYTE;
00101  2     GO TO 12AH;
00102  2    END OVERFLOW;
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC8

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

PL/M - FLOATING POINT INTERFACE

Function

INTERFACES PL/M CONVENTIONS WITH FLOATING POINT ASSEMBLER FORMAT.

Required Hardware

NONE

Required Software

FLOATING POINT MATH PACKAGE -BC1
FLOATING POINT FORMAT CONVERSION PACKAGE -BC2

Input Parameters

AS PR PL/M DEFINITIONS

Output Results

START ADDRESS OF A COPY OF FL. PNT. ACCUMULATOR
TEMP (0)= SIGNIFICANS INDEX
TEMP (1)= FL. PNT. ACC. EXPONENT
TEMP (2)= " SIGN & 1' FRACTION
TEMP (3)= " 2' FRACTION
TEMP (4) " 3' "

Registers Modified: All	**Assembler/Compiler Used:** INTELLEC8/MOD8 VER 2.0
RAM Required: AS TO FL.PNT. + BYTE	**Programmer:** STOLBERG-ROHR MS E15V22
ROM Required: 292 BYTE	**Company:** DANFOSS A/S
Maximum Subroutine Nesting Level: SEE PROGRAM	**Address:** DK6430 NORDBORG DENMARK

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/40 ☒ 8008 ☐ 8080 ☐ 8048 ☐ 8085 ☐ 3000 ☐ Other _____ (use additional sheets if necessary)

Program Title

FLOATING POINT DECIMAL & HEX FORMAT CONVERSION

Function

THE PROGRAM CONVERTS A NUMBER OF MAX 27 CHARACTERS TO STANDARD 13 DIGIT DECIMAL FORMAT AND TO FLOATING POINT ACCUMULATOR FORM IN HEX FORMAT ON THE TELETYPE (SEE EXAMPLE)

Required Hardware

TTY
INTELLEC 8

Required Software

1) INTELLEC 8/MOD 8 VER. 2.1 MONITOR
2) PL/M - FLOATING POINT INTERFACE (Ref. No. BC8)
3) PL/M FLOATING POINT PROCEDURES (Ref. No. BC7)
4) FLOATING POINT MATH PACKAGE (Ref. No. BC1)
5) FLOATING POINT FORMAT CONVERSION PACKAGE (Ref. No. BC2)

Input Parameters

A NUMBER AS DEFINED IN 5) OF MAX 27 CHARACTERS, TERMINATED WITH A 'RUB OUT'

REVISED 8/8/77

Output Results

THE PROGRAM STARTS OUTPUTTING A HEAD LINE, AND INDICATE INPUT MODE WITH A '?', AFTER RECEIVING A 'RUB OUT' THE PROGRAM OUTPUTS THE FLOATING POINT ACCUMULATOR IN DECIMAL AND HEX FORMAT

NOTE: BC9 is ordered as one program with BC7.

Registers Modified: ALL	**Programmer:** STOLBERG-ROHR, MSE15V22
RAM Required: 80 BYTE	**Company:** DANFOSS A/S
ROM Required: 717 BYTE	**Address:** DK 6430 NORDBORG
Maximum Subroutine Nesting Level: 5 LEVELS	**City:**
Assembler/Compiler Used: PL/M VER. 3.3	**State:** DENMARK

```
00001  1
00002  1
00003  1
0  )4  1   /*REF NO. BC9 */
00005  1   /*PROGRAM TITLE FLOATING POINT DECIMAL   HEX FORMAT CONVERSION */
00006  1   /* STOLBERG-ROHR, EL-LAB, E15-V22, 2876, ISAI560. */
00007  1
00008  1   /*************************************************/
00009  1   /*                                               */
00010  1   /* FLOATING POINT PROCEDURES . A1560          */
00011  1   /*                                               */
00012  1   /*************************************************/
00013  1
00014  1   /* INITIALIZE: MOVES A SECTION OF CODE FROM ROM MEMORY TO SCRATCHPAD
00015  1     RAM MEMORY IN PREPARATION FOR EXECUTION OF MULTIPLY AND DIVIDE
00016  1     SUBROUTINES. OVERFLOW FLAG SET TO ZERO. 0 NEST LEVEL
00017  1
00018  1   FINIT: PROCEDURE;
00019  2    GO TO 100H; /* ADDRESS IN PLM-FL.PNT. INTERFACE PROGRAM */
00020  2   END FINIT;
00021  1
00022  1   /* F.P.ADD; ADDS THE SPECIFIED OPERAND TO THE F.P. ACC., RESULT
00023  1     IN TEMP= COPY OF F.P.ACC., RETURN VALUE = ADDR. OF TEMP.
00024  1     3 NEST LEVELS.
00025  1
00026  1   FADD: PROCEDURE (OPERAND) ADDRESS;
00027  2         DECLARE OPERAND ADDRESS;
00028  2   GO TO 103H;
00029  2   END FADD;
0  30  1
00031  1   /* F.P. SUBTRACT: 3 NEST LEVELS.
00032  1
00033  1   FSUB: PROCEDURE (OPERAND) ADDRESS;
00034  1         DECLARE OPERAND ADDRESS;
00035  1    GO TO 106H;
00036  1   END FSUB;
00037  1
00038  1   /* F.P. MULTIPLY. 3 NEST LEVELS.  */
00039  1
00040  1   FMUL: PROCEDURE (OPERAND) ADDRESS;
00041  2         DECLARE OPERAND ADDRESS;
00042  2     GO TO 109H;
00043  2   END FMUL;
00044  1
00045  1    /* F.P. DIVIDE. 3 NEST LEVELS.  */
00046  1
00047  1   FDIV: PROCEDURE (OPERAND) ADDRESS;
00048  2          DECLARE OPERAND ADDRESS;
00049  2     GO TO 10CH;
00050  2   END FDIV;
00051  1
00052  1    /* F.P. ABSOLUTE: SETS THE SIGN OF F. P.ACC. POSITIVE.
00053  1      1 NEST LEVEL.  */
00054  1
0  55  1   FABS: PROCEDURE ADDRESS;
0  56  2    GO TO 10FH;
00057  2   END FABS;
00058  1
00059  1            /* F.P. ZERO: SETS THE F.P.ACC. = 0. 1 NEST LEVEL  */
00060  1
```

```
00061  1    FZRO: PROCEDURE ADDRESS;
00062  2     GO TO 112H;
00063  2    END FZRO;
0  54  1
00065  1     /* F.P. TEST: SETS THE CONTROL FLAGS. 1 NEST LEVEL.  */
00066  1
00067  1    FTST: PROCEDURE ADDRESS;
00068  2     GO TO 115H;
00069  2    END FTST;
00070  1
00071  1     /* F.P. COMPLEMENT: CHANGE THE SIGN OF THE F.P.ACC.
00072  1       1 NEST LEVEL   */
00073  1
00074  1    FCHS: PROCEDURE ADDRESS;
00075  2     GO TO 118H;
00076  2    END FCHS;
00077  1
00078  1            /* F.P. LOAD: LOADS SPEF. OPERAND INTO F.P.ACC.
00079  1        2 NEST LEVELS.  */
00080  1
00081  1    FLOD: PROCEDURE (OPERAND) ADDRESS;
00082  2          DECLARE OPERAND ADDRESS;
00083  2     GO TO 118H;
00084  2    END FLOD;
00085  1
00086  1            /* F.P. INPUT: CONVERTS CHAR.STRING (MAX. 32 DIG ) IN MEM-
00087  1              ORY, TO F.P. FORMAT IN F. P.ACC.. 4 NNEST LEVELS  */
00088  1
00089  1    FINP: PROCEDURE (STRINGBUFFER) ADDRESS;
0  90  2          DECLARE STRINGBUFFER ADDRESS;
00091  2     GO TO 11EH;
00092  2    END FINP;
00093  1
00094  1     /* F.P. OUTPUT: CONVERTS VALUE IN ACC. TO 13 DEC DIG. IN
00095  1       OUTBUFFER. 4 NEST LEVELS.  */
00096  1
00097  1            FOUT: PROCEDURE (OUTBUFFER) ADDRESS;
00098  2          DECLARE OUTBUFFER ADDRESS;
00099  2     GO TO 121H;
00100  2    END FOUT;
00101  1
00102  1     /* F.P. FLOAT: CONVERTS BINARY VALUE IN 5 BYTE BINSTRING
00103  1      ( 32 BIT + 8 BIT SCALE FACTOR ) TO F.P. FORMAT IN ACC.
00104  1      3 NEST LEVELS.  */
00105  1
00106  1    FFLT: PROCEDURE (BINSTRING) ADDRESS;
00107  2          DECLARE BINSTRING ADDRESS;
00108  2     GO TO 124H;
00109  2    END FFLT;
00110  1
00111  1     /* F.P. FIX: CONVERTS VALUE IN ACC. TO FIX FORMAT
00112  1       ( SCALE FACTOR ), RESULT IN TEMP. 2 NEST LEVELS.  */
00113  1
00114  1    FFIX: PROCEDURE (SCALE) ADDRESS;
6  15  2          DECLARE SCALE BYTE;
00116  2     GO TO 127H;
00117  2    END FFIX;
00118  1
00119  1     /* OVERFLOW, RETURNS THE VALUE OF THE OVERFLOW FLAG,
00120  1       AND CLEAR THE FLAG.  */
```

```
00121  1
00122  1   OVERFLOW: PROCEDURE BYTE;
00123  2    GO TO 12AH;
00124  2   END OVERFLOW;
00125  1
00126  1
00127  1   TTYOUT: PROCEDURE (X); /* ROUTINES IN VER 2.1 MONITOR */
00128  2           DECLARE X BYTE;
00129  2           GO TO 3809H;
00130  2    END TTYOUT;
00131  1
00132  1
00133  1   TTYIN: PROCEDURE BYTE;
00134  2           GO TO 3F44H;
00135  2   END TTYIN;
00136  1
00137  1
00138  1   CRLF: PROCEDURE;
00139  2           GO TO 3CC7H;
00140  2   END CRLF;
00141  1
00142  1
00143  1   HEX: PROCEDURE (X);
00144  2   DECLARE (X,Y) BYTE;
00145  2   Y=X AND 0FH;
00146  2   X=SHR(X,4);
00147  2   IF X>9 THEN X=X+'A'-10; ELSE
00148  2   X=X+'0';
00149  2   CALL TTYOUT(X);
00150  2   IF Y>9 THEN Y=Y+'A'-10; ELSE
00151  2   Y=Y+'0';
00152  2   CALL TTYOUT(Y),
00153  2   RETURN;
00154  2   END HEX;
00155  1
00156  1   /*
00157  1   OUTPUT HEAD
00158  1   */
00159  1
00160  1   DECLARE HEAD DATA ('INPUT F.P. FORMAT   EXP. ACC1 ACC2 ACC3');
00161  1   DECLARE (X,J) BYTE;
00162  1   DECLARE INBUF(28) BYTE, OUTBUF(13) BYTE;
00163  1   DECLARE Y ADDRESS; DECLARE (ACC BASED Y) (5) BYTE;
00164  1   CALL CRLF; CALL FINIT;
00165  1   DO J=1 TO 5; CALL TTYOUT(' '); END;
00166  1   DO J=0 TO 4; CALL TTYOUT(HEAD(J)); END;
00167  1   DO J=1 TO 20;CALL TTYOUT(' '); END;
00168  1   DO J=5 TO LAST(HEAD); CALL TTYOUT(HEAD(J)); END;
00169  1   CALL CRLF;
00170  1
00171  1   /*
00172  1   MAIN PROGRAM
00173  1   */
00174  1
00175  1   DO WHILE 1;
00176  1   J,X=0;
00177  2   CALL TTYOUT (3FH); CALL TTYOUT(' ');
00178  2
00179  2   /*
00180  2   KEYPBOARD INPUT ROUTINE
```

```
00181  2    */
00182  2
00183  2    DO WHILE X<>4FH;
0  84  2      X,INBUF(J)=TTYIN-30H;
00185  3      IF J=26 THEN DO; X,INBUF(J)=4FH; END; ELSE
00186  3      J=J+1;
00187  3    END;
00188  2    X=J+1;
00189  2    Y=FINP(.INBUF);
00190  2    DO J=1 TO 4; INBUF(J)=ACC(J); END;
00191  2    Y=FOUT(.OUTBUF);
00192  2    DO J=X TO 27; CALL TTYOUT(' '); END;
00193  2    DO J=0 TO 12; CALL TTYOUT(OUTBUF(J)+'0'); END;
00194  2    CALL TTYOUT(' '); CALL TTYOUT('=');
00195  2    DO J=1 TO 4;
00196  2      CALL TTYOUT(' '); CALL TTYOUT(' ');
00197  3      CALL HEX(INBUF(J));
00198  3      CALL TTYOUT(' ');
00199  3    END;
00200  2      CALL CRLF;
00201  2    END;
00202  1     EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB20

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

BINARY MULTIPLICATION AND LEADING ZERO BLANKING - N-BYTE

Function

THE PROGRAM PERFORMS BINARY MULTIPLICATION ON TWO NUMBERS AND RETURNS A RESULT THAT MAY BE UP TO 255 BYTES IN LENGTH.

Required Hardware

TTY ON PORT 0 AND 1.

Required Software

QBM11E- BINARY MULTIPLICATION,
QLZ11E- REMOVE LEADING ZEROES SUBROUTINE,

Input Parameters

Output Results

THE SYSTEM MONITOR IS USED TO VERIFY THE RESULT BY CHECKING THE APPROPRIATE MEMORY LOCATIONS.

Registers Modified: ALL	**Assembler/Compiler Used:** INTELLEC 8/MOD 80 V3.0
RAM Required: ALL 3 PROGRAMS STACK= 16 BYTES, PROGRAM= 219 BYTES.	**Programmer:** CHARLES SOOLEY
ROM Required: NONE	**Company:** ENVIRONMENT CANADA
Maximum Subroutine Nesting Level: TWO	**Address:** 4905 DUFFERIN STREET, DOWNSVIEW ONTARIO, CANADA.

```
;REF. NO. B620
;PROGRAM TITLE N-BYTE BINARY MULTIPLICATION & LEADING ZERO BLANK
;*****************************************************
;                    QBM11S
;*****************************************************
;    BINARY MULTIPLICATION PROGRAM.
;    THE PROGRAM ACCEPTS TWO FIELDS OF ANY LENGTH
;    (SUBJECT TO THE CONDITIONS BELOW) AND RETURNS
;    THEIR PRODUCT.
;    NOTES:
;    1) STACK ORGANIZATION OF REQUIRED PARAMETERS:
;       LOCATION       EXPLANATION          LABEL
;         SP+6   ADDRESS OF RESULT          MULTR
;         SP+4   ADDRESS OF MULTIPLICAND    MULTC
;         SP+2   ADDRESS OF MULTIPLIER      MULTP
;          SP    QBM11 ENTRY POINT          BEGIN
;    2) THE FIRST WORD OF EACH AREA IS SET UP AS
;       FOLLOWS:
;        FIELD WIDTH, MSBYTE,.....,LSBYTE.
;    3) THE RESULT WILL CONTAIN NO LEADING ZEROES.
;    4) THE RESULT AREA MUST BE AT LEAST THE T
;       WIDTH OF THE MULTIPLIER AREA PLUS THE
;       MULTIPLICAND AREA.
;    5) THE MULTIPLIER WILL CONTAIN ZEROES AT
;       COMPLETION OF THE PROGRAM.
;    6) THE SUM OF THE MULTIPLIER AND MULTIPLICAND
;       AREAS MUST NOT EXCEED 255 (FF) BYTES.
;    7) ADDITIONAL SUBROUTINES USED:
;       QLZ11- LEADING ZERO BLANKING.
;*****************************************************
;           DATE: JULY 22,1975.
;           AUTHOR: CHARLES SOOLEY.
;           DEPARTMENT: ATMOSPHERIC ENVIRONMENT
;                       SERVICE (AIDX),
;                       DOWNSVIEW CANADA.
;*****************************************************
;    SHIFTS RIGHT 1 BIT THE NUMBER OF BYTES
;    CONTAINED AT ADDRESS.
;
SHIFR   MACRO ADDRS
        LHLD ADDRS      ;NUMBER OF BYTES
        MOV A,M
        MOV C,A
ROTATE: INX H           ;LOCATION OF BYTE TO SHIFT
        MOV A,M         ;WORD TO SHIFT
        RAR             ;SHIFT RIGHT ONE BIT
        MOV M,A         ;RETURN WORD TO MEMORY
        DCR C           ;DECREASE WORD COUNT
        JNZ ROTATE      ;CONTINUE UNTIL ZERO
```

```
                        ENDM
0100                    ORG 0100H
0100            MULTP:  DS 2
0102            MULTC:  DS 2
0104            MULTP:  DS 2
0106 33         BEGIN:  INX SP
0107 33                 INX SP
0108 E1                 POP H           ;ADDRESS OF MULTP
0109 220001             SHLD 0100H
010C E1                 POP H           ;ADDRESS OF MULTC
010D 220201             SHLD 0102H
0110 E1                 POP H           ;ADDRESS OF MULTR
0111 220401             SHLD 0104H
0114 3B                 DCX SP          ;RETURN SP TO PROPER
0115 3B                 DCX SP          ;POSITION
0116 3B                 DCX SP
0117 3B                 DCX SP
0118 3B                 DCX SP
0119 3B                 DCX SP
011A 3B                 DCX SP
011B 3B                 DCX SP
  ‘00           QLZ11   EQU 0700H
‘.‘1C 2A0001    ZERO:   LHLD MULTP      ;NUMBER OF BYTES OF THE
011F 7E                 MOV A,M         ;MULTIPLIER IS ADDED
0120 2A0201             LHLD MULTC
0123 86                 ADD M           ;TO THE NUMBER OF BYTES
0124 4F                 MOV C,A
0125 2A0401             LHLD MULTR
0128 77                 MOV M,A         ;OF THE MULTIPLICAND
0129 AF                 XRA A           ;CLEAR ACCUMULATOR
012A 23         CLEAR:  INX H           ;THE TOTAL NUMBER OF BYTES
012B 77                 MOV M,A         ;THAT THE RESULT WILL
012C 0D                 DCR C           ;OCCUPY ARE ZEROED
012D C22A01             JNZ CLEAR       ;JUMP UNTIL COMPLETE
                ;
                ;    MAIN PROGRAM LOOP
                ;
0130 2A0001             LHLD MULTP      ;SET COUNTERS FOR LOOP
0133 46                 MOV B,M         ;NUMBER OF BYTES IN MULTP
0134 0E08               MVI C,8         ;8 BITS PER BYTE COUNTER
0136 C5         LOOP:   PUSH B
0137 F5                 PUSH PSW
0138 AF                 XRA A           ;RESETS CARRY
                +       SHIFR MULTP
0139 2A0001     +       LHLD 00100H     ;NUMBER OF BYTES
013C 7E         +       MOV A,M
‘13D 4F         +       MOV C,A
  ,3E 23        +ROTATE: INX H          ;LOCATION OF BYTE TO SHIFT
013F 7E         +       MOV A,M         ;WORD TO SHIFT
0140 1F         +       RAR             ;SHIFT RIGHT ONE BIT
```

```
0141 77       +          MOV M,A          ;RETURN WORD TO MEMORY
0142 0D       +          DCR C            ;DECREASE WORD COUNT
0143 C23E01   +          JNZ ROTATE       ;CONTINUE UNTIL ZERO

0146 D24C01              JNC PARTB         ;CARRY BIT IS ZERO ON JUMP
0149 CD6C01              CALL ADDRE        ;ADDING MULTC TO MULTR
             +PARTB:     SHIFR MULTR       ;SHIFT RESULT RIGHT
014C 2A0401   +          LHLD 00104H       ;NUMBER OF BYTES
014F 7E       +          MOV A,M
0150 4F       +          MOV C,A
0151 23      +ROTATE:    INX H            ;LOCATION OF BYTE TO SHIFT
0152 7E       +          MOV A,M          ;WORD TO SHIFT
0153 1F       +          RAR              ;SHIFT RIGHT ONE BIT
0154 77       +          MOV M,A          ;RETURN WORD TO MEMORY
0155 0D       +          DCR C            ;DECREASE WORD COUNT
0156 C25101   +          JNZ ROTATE       ;CONTINUE UNTIL ZERO

0159 F1                  POP PSW
015A C1                  POP B
015B 0D                  DCR C             ;DECREASE INTERNAL COUNTER
015C C23601              JNZ LOOP          ;CONTINUE FOR ONE BYTE
015F 05                  DCR B             ;LOOP UNTIL TOTAL
0160 0E08                MVI C,8
0162 C23601              JNZ LOOP          ;BYTES ARE DONE
0165 2A0401              LHLD MULTR
0168 CD0007              CALL QLZ11        ;PACK RESULT
016B C9                  RET
                         ;
                         ;    ADDS THE MULTC TO THE RESULT
                         ;
016C 2A0401   ADDRE:     LHLD MULTR
016F EB                  XCHG
0170 2A0201              LHLD MULTC        ;INCREASE HL AND DE
0173 46                  MOV B,M           ;COUNTERS TO THE ADDRESS
0174 48                  MOV C,B           ;OF THE LEAST SIGNIFICANT
                                           ;BYTE OF MULTC AND MULTR
0175 23       EMUC:      INX H
0176 13                  INX D
0177 05                  DCR B
0178 C27501              JNZ EMUC
017B AF                  XRA A             ;CLEAR CARRY BIT
017C 1A       ADDING:    LDAX D            ;LOAD LOWER BYTE
017D 8E                  ADC M             ;ADD MULTC
017E 12                  STAX D            ;RETURN SUM TO MULTR
017F 0D                  DCR C             ;DECREASE COUNTER
0180 C8                  RZ                ;RETURN ON ZERO
0181 1B                  DCX D             ;NEXT MEMORY LOCATION
  82 2B                  DCX H
0183 C37C01              JMP ADDING        ;CONTINUE ADDING.
0000                     END
```

```
                ;*****************************************************
                ;                    QLZ11S
                ;*****************************************************
                ;   LEADING ZERO BLANKING OF FIELD BEGINNING
                ;   AT ADDRESS CONTAINED IN REGISTERS HL.
                ;   NOTES:
                ;   1) ADDRESS    CONTENTS
                ;        HL       FIELD WIDTH (W)
                ;       HL+1      DATA BYTE 1
                ;         :            :
                ;         :            :
                ;        HL+W      DATA BYTE W.
                ;   2) THE FIELD WIDTH IS MODIFIED TO REFLECT
                ;      ANY LEADING ZEROES THAT ARE REMOVED.
                ;   3) CONTENTS OF ALL REGISTERS ARE MODIFIED.
0700                    ORG 0700H
                QLZ11:
0700 0600               MVI B,0         ;CLEAR REGISTER B
0702 E5                 PUSH H
0703 4E                 MOV C,M         ;NUMBER OF BYTES IN RESULT
0704 23         CHECK:  INX H
0705 7E                 MOV A,M
0706 D600               SUI 0           ;CHECK TO SEE IF LEADING
0708 C21207             JNZ PACK        ;ZERO IS FOUND
070B 04                 INR B           ;NUMBER OF LEADING ZEROES
070C 0D                 DCR C           ;NUMBER OF WORDS TO CHECK
070D C20407             JNZ CHECK
0710 E1                 POP H
0711 C9                 RET
0712 78         PACK:   MOV A,B
0713 D600               SUI 0           ;IF THERE IS NO LEADING
0715 C21A07             JNZ ALLZE       ;ZEROES RETURN
0718 E1                 POP H
0719 C9                 RET
071A D1         ALLZE:  POP D
071B 1A                 LDAX D          ;NUMBER OF WORDS TO
071C 90                 SUB B           ;TRANSFER
071D C8                 RZ              ;RETURN IF ALL OF RESULT
                                        ;IS ZERO
071E 12                 STAX D          ;PACK NUMBER OF WORDS
071F 47                 MOV B,A         ;NUMBER OF WORDS TO MOVE
0720 13         MOVE:   INX D           ;POINTER TO FIRST WORD
0721 7E                 MOV A,M         ;PACK WORDS
0722 12                 STAX D          ;START AT BEGINNING
0723 23                 INX H           ;OF RESULT AREA
0724 05                 DCR B           ;CONTINUE UNTIL ALL
0725 C22207             JNZ MOVE        ;WORDS ARE MOVED
0728 C9                 RET             ;PACK FINISHED.
0000                    END
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

DECIMAL MULTIPLICATION SUBROUTINE - DMULT

Function

To multiply M Decimal Digits by N Decimal Digits and store the product - 7 digits x 3 digits as written, but easy to expand if required
(See additional sheets for explanation in detail.)

Required Hardware

As required by supporting software

Required Software

User's program to load multiplicand and multiplier into specified memory locations, and to extract product from specified memory locations

Input Parameters

Multiplicand to be stored	1FØØ(LSD) to 1F06(MSD).
Multiplier to be stored	1FØA(LSD) to 1FØC(MSD).
Product delivered to	1F2Ø(LSD) to 1F29(MSD).

Total storage used by routine, including input and output, is from 1FØØ to 1F2C inclusive.

Output Results

Input and output data are in BCD Format, occupying the four lowest bits in each store location.

Registers Modified: A, B, C, D, E, H, L	Programmer: Peter Hand
RAM Required: (about) 44 bytes	Company: De La Rue Instruments Ltd.
ROM Required: (about) 320 bytes	Address: Norway Rd.
Maximum Subroutine Nesting Level: 3 including DMULT	City: Hilsea, Portsmouth
Assembler/Compiler Used: INTELLEC 8 MACRO ASSEMBLER	State: England

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BA6

☐ 4004 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

BCD Multiplication

Function

Multiplies an up to six digit bcd number by a 4 digit bcd number providing a ten digit bcd result. All numbers are unsigned.

Required Hardware

One ROM with partial product look-up table. (see attached)

Required Software

None

Input Parameters

RAM Contents	Symbol	#bytes	Data format
Multiplicand	HOLD	6	LS-Digit...MS-Digit One digit per byte Right justified
Multiplier	KONST	2	MS-Digit...LS-Digit Two digits per byte

Register contents - don't care

Output Results

Product is stored in RRAM at symbol location SUM, with two digits per byte, LS-byte first thru MS-byte.

Registers Modified: A,b,c,d,e,h,l	**Maximum Subroutine Nesting Level:** Three
RAM Required: 26 bytes	**Assembler/Compiler Used:** 8080 Macro Assembler
ROM Required: Table look up- 100 bytes	**Programmer:** Daniel S. Coolidge
Program- 225 bytes	**Company:** Barnes Engineerin Co. 80 Commerce Rd - Stamford, Conn.

ADDENDUM TO MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref.- Required Hardware for program.

The program makes use of a table look-up to determine partial products. It does this by composing the bcd numbers as the first and second half of the least significant byte of a PROM address. In a system which utilizes 256 X 8 PROMS, the most significant byte of a memory address calls up a particular PROM. This is the value of the symbol MLTBL. Hence the PROM is programmed by placing the product of the internal addresses read in BCD in memory. For example: assume PROM with most significant byte address 0AH, and with a call to read from memory the contents of 0A48H. PROM address 48H would contain the product of 4 and 8 expressed as 32H. This method wastes some memory, but considerably speeds up the process of BCD multiplication.

```
                ;REF. NO. BA6
                ;PROGRAM TITLE BCD MULTIPLICATION
                ;
                ;
                ;
                ;
                ; THESE VALUE ASSIGNED ONLY FOR PURPOSE OF
                ; ASSEMBLY - THEY MAY BE RE-DEFINED AT WILL
                ;
                ;
0050            MLTBL   EQU     50H
0FA0            KONST   EQU     4000D
0FA4            HOLD    EQU     4004D
0FAA            TEMP    EQU     4010D
0FB0            SUM     EQU     4016D
                OPEN:
0000 97                 SUB     A       ;CLEAR A AND CARRY
0001 32B00F             STA     SUM
0004 32B10F             STA     SUM+1
0007 32B20F             STA     SUM+2
000A 32B30F             STA     SUM+3
000D 32B40F             STA     SUM+4
0010 32B50F             STA     SUM+5
0013 32B60F             STA     SUM+6
0016 32B70F             STA     SUM+7
0019 32B80F             STA     SUM+8
001C 32B90F             STA     SUM+9
001F 3AA00F             LDA     KONST   ;GET LOW ORDER DIGITS OF MULTPLR
0022 E60F               ANI     0FH     ;MASK OUT HIGH ORDER DIGIT
0024 07                 RLC
0025 07                 RLC
0026 07                 RLC
0027 07                 RLC
0028 67                 MOV     H,A     ;STORE LEFT JUSTIFIED DIGIT IN H
0029 E60F               ANI     0FH     ;COMPUTE PARTIAL PRODUCT
002B 01AA0F             LXI     B,TEMP  ;PREPARE FOR SUBROUTINE
002E 21B00F             LXI     H,SUM
0031 1E00               MVI     E,0
0033 CDC200             CALL    CONST   ;CREATE 'SUM'
0036 3AA00F             LDA     KONST   ;GET NEXT DIGIT OF MULTIPLICAND
0039 E6F0               ANI     0F0H    ;MASK OUT LOW ORDER DIGIT
003B 67                 MOV     H,A
003C CDA700             CALL    PART    ;CREATE NEXT PARTIAL PRODUCT
003F 01AA0F             LXI     B,TEMP
0042 21B10F             LXI     H,SUM+1
0045 1E00               MVI     E,0
0047 CDC200             CALL    CONST
004A 3AA10F             LDA     KONST+1
004D E60F               ANI     0FH
```

```
004F 07              RLC
0050 07              RLC
0051 07              RLC
0052 07              RLC
0053 67              MOV     H,A
0054 CDA700          CALL    PART
0057 01AA0F          LXI     B,TEMP
005A 21B20F          LXI     H,SUM+2
005D 1E00            MVI     E,0
005F CDC200          CALL    CONST
0062 3AA10F          LDA     KONST+1
0065 67              MOV     H,A
0066 CDA700          CALL    PART
0069 01AA0F          LXI     B,TEMP
006C 21B30F          LXI     H,SUM+3
006F 1E00            MVI     E,0
0071 CDC200          CALL    CONST
0074 01B00F          LXI     B,SUM
0077 97      COMP:   SUB     A       ;CLEAR CARRY AND ACCUMULATOR
0078 5F              MOV     E,A     ;CLEAR E REG
0079 2608            MVI     H,8D    ;INITIALIZE LOOP COUNTER
007B 0A              LDAX    B
007C 57              MOV     D,A
007D E60F            ANI     0FH     ;PREPARE 1ST DIGIT OF PRODUCT
007F 02              STAX    B       ;REPLACE SUM DIGIT W/PRODUCT DIGIT
0080 7A      LOOP    MOV     A,D     ;GET HIGH ORDER DIGIT
0081 0F              RRC             ;RIGHT JUSTIFY
0082 0F              RRC
0083 0F              RRC
0084 0F              RRC
0085 E60F            ANI     0FH     ;MASK DIGIT
0087 57              MOV     D,A     ;STORE IN D REG
0088 97              SUB     A       ;CLEAR CARRY AND ACCULMULATOR
0089 03              INX     B
008A 0A              LDAX    B       ;GET NEXT DIGIT OF SUM
008B 83              ADD     E       ;ADD ANY CARRIES
008C 27              DAA
008D 1E00            MVI     E,0     ;CLEAR "CARRY REGISTER"
008F D29400          JNC     NEXT    ;IF CARRY RESULTS, PLACE IN E REG
0092 1E10            MVI     E,10H
0094 82      NEXT:   ADD     D       ;ADD SPIILLOVER DIGIT
0095 27              DAA
0096 D29B00          JNC     MORE
0099 1E10            MVI     E,10H   ;IF CARRY RESULT, STORE IN E REG
009B 57      MORE:   MOV     D,A     ;STORE RESULT IN D
009C E60F            ANI     0FH     ;MASK LOW ORDER DIGIT
009E 02              STAX    B       ;STORE, REPLACING 'SUM' DIGIT
009F 7C              MOV     A,H     ;GET LOOP COUNTER
00A0 D601            SUI     1D      ;DECREMENT LOOP COUNTER
00A2 67              MOV     H,A     ;STORE LOOP COUNTER
```

```
00A3 C28000            JNZ     LOOP    ;IF LOOP COUNTER <>0 REPEAT LOOP
00A6 C9                RET
00A7 11A40F   PART:    LXI     D,HOLD  ;MULTIPLICAND ADDRESS IN D
00AA 01AA0F            LXI     B,TEMP  ;PARTIAL PRODUCT ADDRESS IN B
00AD E5                PUSH    H
00AE 2E05              MVI     L,5
00B0 CDE800   LUP1:    CALL    MUL
00B3 02                STAX    B
00B4 03                INX     B
00B5 13                INX     D
00B6 2D                DCR     L
00B7 97                SUB     A
00B8 BD                CMP     L
00B9 C2B000            JNZ     LUP1
00BC CDE800            CALL    MUL
00BF E1                POP     H
00C0 02                STAX    B
00C1 C9                RET
00C2 97       CONST:   SUB     A
00C3 1E00              MVI     E,0
00C5 1606              MVI     D,6D
00C7 0A       LUP2:    LDAX    B
00C8 83                ADD     E
00C9 27                DAA
00CA 1E00              MVI     E,0
00CC D2D100            JNC     ALPHA
00CF 1E10              MVI     E,10H
00D1 86       ALPHA:   ADD     M
00D2 27                DAA
00D3 77                MOV     M,A
00D4 D2D900            JNC     NONE
00D7 1E10              MVI     E,10H
00D9 03       NONE:    INX     B
00DA 23                INX     H
00DB 15                DCR     D
00DC 97                SUB     A
00DD BA                CMP     D
00DE C2C700            JNZ     LUP2
00E1 D2E700            JNC     EXIT
00E4 7B                MOV     A,E
00E5 86                ADD     M
00E6 77                MOV     M,A
00E7 C9       EXIT:    RET
00E8 1A       MUL:     LDAX    D
00E9 B4                ORA     H
00EA 6F                MOV     L,A
00EB 2650              MVI     H,MLTBL
00ED 7E                MOV     A,M
00EE C9                RET
0000                   END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB21

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

BINARY MUL/DIV MULTI-PRECISION PACK FOR 8080

Function

Signed fixed-point binary fraction multiply and divide. Double-precision inputs, double-precision output for divide and 4-byte output for multiply.
Time for multiply 1.2845 - 1.5085 milliseconds.
Time for divide 1.200 - 1.230 milliseconds.

Required Hardware

Possibility to load the program into PROM or RAM or to simulate it. In the given example a RAM with addresses from 1024 to 2047 is used.

Required Software

A test program may be used.

Input Parameters

Multiply: multiplicand in BC, multiplier in DE, multiplicand ≠ -1, stack pointer must be initiated.
Divide: dividend in HL, divisor in DE,/dividend/ less then /divisor/, stack pointer must be initiated.

Output Results

Product in HLDE. Quotient in BC. Remainder in HL.

Registers Modified: All register and flags	**Assembler/Compiler Used:** 1
RAM Required: 12 bytes (stack +input + result)	**Programmer:** The INTELLEC 81 assembler
ROM Required: 120 bytes	**Company:** Lennart Wilholmsson SIFU Elteknik Sweden
Maximum Subroutine Nesting Level:	**Address:** Box 4012 S-102 61 Stockholm, SWEDEN

```
                ;REF. NO. BB21
                ;PROGRAM TITLE MUL/DIV MULTI-PRECISION PACK FOR 8080
                ;
                ;
                ;
                ;
                ;MUL/DIV MULTI-PRECISION PACK FOR 8080
0500                    ORG     1280
0500 214007 AMUL:       LXI     H,1856  ;LOOP COUNTER ADDRESS IN HL
0503 E5                 PUSH    H       ;LOOP COUNTER ADDRESS TO MEMORY STACK
0504 3610               MVI     M,16    ;LOOP COUNTER IN MEMORY IS SET TO 16
0506 210000             LXI     H,0     ;CLEAR RESULT IN HL
0509 7B                 MOV     A,E     ;LSW OF MULTIPLIER TO A
050A A7                 ANA     A       ;CLEAR CARRY
050B D21705 ALOOP:      JNC     ANEXT
050E E601               ANI     1       ;MASK LSB OF A
0510 C22205             JNZ     ACOUNT  ;DO NOTHING IF MULTIPLIER DIGIT IS 1 AND
                                        ;NEXT HIGHER ORDER DIGIT IS 1
0513 09                 DAD     B       ;ADD MULTIPLICAND TO PARITAL PRODUCT IF
                                        ;MULTIPLIER DIGIT IS 1 AND NEXT HIGHER O
                                        ;DIGIT IS 0
0514 C32205             JMP     ACOUNT
0517 E601    ANEXT:     ANI     1       ;MASK LSB OF A
0519 CA2205             JZ      ACOUNT  ;DO NOTHING IF MULTIPLIER DIGIT IS 0 AND
                                        ;NEXT HIGHER ORDER DIGIT IS 0
051C 7D                 MOV     A,L     ;SUBTRACT MULTIPLICAND FROM PARTIAL PROD
051D 91                 SUB     C       ;IF MULTIPLIER DIGIT IS 0 AND NEXT HIGHE
051E 6F                 MOV     L,A     ;DIGIT IS 1
051F 7C                 MOV     A,H
0520 98                 SBB     B
0521 67                 MOV     H,A
0522 E3      ACOUNT:    XTHL            ;CHANGE CONTENTS OF HL WITH ADDRESS TO L
                                        ;COUNTER IN STACK
0523 35                 DCR     M       ;DECREMEET LOOP COUNTER IN MEMORY
0524 E3                 XTHL
0525 CA3A05             JZ      AEND    ;JMP OUT OF LOOP IF LOOP COUNTER IS 0
0528 7C                 MOV     A,H     ;ARITHMETIC SHIFT RIGHT OF PARTIAL PRODU
0529 FE80               CPI     128     ;AND MULTIPLIER IN HLDE
052B 3F                 CMC
052C 1F                 RAR
052D 67                 MOV     H,A
052E 7D                 MOV     A,L
052F 1F                 RAR
0530 6F                 MOV     L,A
0531 7A                 MOV     A,D
0532 1F                 RAR
0533 57                 MOV     D,A
0534 7B                 MOV     A,E
0535 1F                 RAR
```

```
0536 5F                 MOV     E,A
0537 C30B05             JMP     ALOOP
053A 3EFE       AEND:   MVI     A,254
053C A3                 ANA     E       ;CLEAR LSB OF PRODUCT
053D 5F                 MOV     E,A
053E 33                 INX     SP      ;MAKE STACK POINTER POINT TO RETURN ADDR
053F 33                 INX     SP
0540 C9                 RET
0541 014007     ADIV:   LXI     B,1856  ;LOOP COUNTER ADDRESS IN BC
0544 C5                 PUSH    B       ;LOOP COUNTER ADDRESS TO MEMORY STACK
0545 E3                 XTHL            ;CHANGE CONTENTS OF HL WITH ADDRESS TO
                                        ;LOOP COUNTER IN STACK
0546 360F               MVI     M,15    ;LOOP COUNTER IN MEMORY IS SET TO 15
0548 E3                 XTHL
0549 7C         AAGAIN: MOV     A,H     ;MSW OF PARTIAL DIVIDEND ( REMAINDER ) T
054A AA                 XRA     D
054B 37                 STC
054C F25005             JP      AALT1   ;IF SIGNS OF PARTIAL DIVIDEND AND DIVISO
                                        ;ARE THE SAME, WRITE A 1 INTO THE
                                        ;APPROPRIATE QUOTIENT POSITION
054F 3F                 CMC             ;IF SIGNS ARE DIFFERENT, WRITE A 0
0550 79         AALT1:  MOV     A,C     ;INTO THE QUOTIENT
0551 17                 RAL
0552 4F                 MOV     C,A
0553 78                 MOV     A,B
0554 17                 RAL
0555 47                 MOV     B,A
0556 29                 DAD     H
0557 29                 DAD     H       ;ARITHMETIC SHIFT LEFT OF HL
0558 F25F05             JP      AALT2
055B 19                 DAD     D       ;IF SIGNS ARE DIFFERENT, ADD DIVISOR TO
                                        ;REMAINDER
055C C36505             JMP     ACONT
055F 7D         AALT2:  MOV     A,L     ;IF SIGNS ARE THE SAME, SUBTRACT DIVISOR
0560 93                 SUB     E       ;FROM REMAINDER
0561 6F                 MOV     L,A
0562 7C                 MOV     A,H
0563 9A                 SBB     D
0564 67                 MOV     H,A
0565 E3         ACONT:  XTHL            ;CHANGE CONTENTS OF HL WITH ADDRESS TO
                                        ;LOOP COUNTER IN STACK
0566 35                 DCR     M       ;DECREMENT LOOP COUNTER IN MEMORY
0567 E3                 XTHL
0568 C24905             JNZ     AAGAIN  ;CONTINUE IN LOOP IF LOOP COUNTER NOT 0
056B 37                 STC             ;ARITHMETIC SHIFT LEFT OF BC GIVES
056C 79                 MOV     A,C     ;THE PSEUDO-QUOTIENT
056D 17                 RAL             ;ADD CORRECTION 1000000000000001 TO
056E 4F                 MOV     C,A     ;PSEUDO QUOTIENT
056F 78                 MOV     A,B
0570 17                 RAL
```

```
0571 47             MOV     B,A
0572 3E80           MVI     A,128
0574 80             ADD     B
0575 47             MOV     B,A
0576 33             INX     SP      ;MAKE STACK POINTER POINT TO RETURN ADDR
0577 33             INX     SP
0578 C9             RET             ;RETURN TO MAIN PROGRAM
0000                END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB22

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

DOUBLE PRECISION MULTIPLY - DMPY

Function

TO MULTIPLY TWO 16-BIT NUMBERS, RETURNING THE MOST-SIGNIFICANT 16 BITS (IN ADDRESS FORM) THRU THE APPROPRIATE REGISTERS TO THE CALLING PROGRAM.
THE INTRINSIC PL/M MULTIPLY CAPABILITY IS EMPLOYED FOR THE BYTE-BY-BYTE MULTIPLICATIONS.

Required Hardware

NONE

Required Software

COMPILE WITH PL/M

Input Parameters

CALLING SEQUENCE IS AS FOLLOWS:

```
ADDRESS VARIABLE=DMPY(A,HI,LO);
```

WHERE
'ADDRESS VARIABLE' IS A VARIABLE WHICH HAS BEEN DECLARED AS AN ADDRESS
A IS ARGUMENT 1, A 16 BIT NUMBER DECLARED 'ADDRESS'
HI IS THE HIGH-ORDER BYTE OF ARGUMENT 2
LO IS THE LOW-ORDER BYTE OF ARGUMENT 2

Output Results

ARGUMENT 2 IS PASSED BYTE-BY-BYTE TO ALLOW THE ARGUMENT TO BE EITHER A CONSTANT SETUP BY A 'DATA' STATEMENT (WHICH IS ALWAYS TREATED AS A BYTE STRING), OR AN ADDRESS VARIABLE.

THE RESULTS, IN THE FORM OF THE HIGH 16 BITS OF THE PRODUCT, ARE RETURNED IN ADDRESS FORM TO THE CALLING PROGRAM.

Registers Modified:	Assembler/Compiler Used:
ALL	PL/M
RAM Required:	**Programmer:**
1Ø BYTES	WAYNE A. MILLER
ROM Required:	**Company:**
193 BYTES	CONTEC CONTROLS
Maximum Subroutine Nesting Level:	**Address:**
2 LEVELS	1485 DAVIS RD, ELGIN IL 6Ø12Ø

```
00001  1
00002  1   /*REF. NO. BB22 */
00003  1   /*PROGRAM TITLE DMPY DOUBLE PRECISION MULTIPLY */
00004  1
00005  1   DMPY: PROCEDURE(A,HI,LO) ADDRESS; /*DOUBLE PREC. MULTIPLY*/
00006  2     DECLARE (A,RES,P) ADDRESS; /*RTN 2 HI-ORD BYTES OF PRODUCT*/
00007  2     DECLARE (HI,LO)BYTE;
00008  2      P=LOW(A)*LO ; /* GET LOWEST 2 BYTES OF PRODUCT*/
00009  2      P=HIGH(P) + HIGH(A)*LO + LOW(A)*HI ; /*=MID 2 BYTES OF PRODUCT */
00010  2      IF CARRY THEN RES=100H; ELSE RES=0;
00011  2      IF LOW(P)>7FH THEN RES=RES+1; /*ROUND*/
00012  2      RETURN HIGH(A)*HI +RES+HIGH(P);
00013  2       END DMPY;
00014  1
00015  1   /* TEST PROGRAM STARTS HERE * * */
00016  1     DECLARE (A,B,X,Y,Z,R,S,T,U) ADDRESS, DB DATA (12136);
00017  1     A=49152; /*=.75 AS LEFT JUSTIFIED FRACTION*/
00018  1     Z=DMPY(A,DB,DB(1)); /*Z=9102,I HOPE*/
00019  1   /* TRY WORST CASES-WHERE 3 OR 4 ARG BYTES HAVE MSB SET, */
00020  1   /* WHICH PRODUCES CARRY IN STATEMENT 6 */
00021  1     A=61183; /* =EEFF HEX */
00022  1     B=65518; /* =FFEE HEX */
00023  1     X=DMPY(A,HIGH(B),LOW(B));/*X SHOULD=(61183*65518)/2**16 */
00024  1     Y=DMPY(B,HIGH(A),LOW(A));/*  =61166 */
00025  1
00026  1     A=65518;
00027  1     R=DMPY(A,HIGH(B),LOW(B));/*R=65518**2/2**16 =65500*/
00028  1
00029  1     A=60928; /*=EE00 HEX */
00030  1     S=DMPY(A,HIGH(B),LOW(B));/*S=(60928*65518)/2**16 =60911*/
00031  1
00032  1     B=61183; /*EEFF HEX */
00033  1     T=DMPY(B,HIGH(A),LOW(A));/*T=(61183*60928)/2**16 =56881*/
00034  1
00035  1     A=33333;
00036  1     U=DMPY(A,HIGH(B),LOW(B));/*U=31118.97=31119 ROUNED*/
00037  1   EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB23

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

16 BIT SQUARE ROOT ROUTINE

Function

Returns 16 bit square root (8 bit whole number joined with 8 bit fraction) of a 16 bit argument. The result conforms to standard signed number convention; therefore, its highest order bit will always be zero. The argument must have zeros in its two highest order bits, for its square root to lie in the valid range of the signed result.

An optional shortened version of the subroutine returns the result in two's complement form.

Required Hardware

none

Required Software

none

Input Parameters

16 bit argument in HL

Output Results

16 bit result in HL

(Option 2: 16 bit two's complemented result in BC)

Registers Modified:	Assembler/Compiler Used:
A B C D E H L	8080 Macro, version 2.3
RAM Required:	**Programmer:**
2 words (one level of stack)	R. E. DuPuy
ROM Required:	**Company:**
55 words	Tektronix
Maximum Subroutine Nesting Level:	**Address:**
0	Beaverton, Or. 97005 (MS 58-736)

```
                ;REF. NO.  BB23
                ;PROGRAM TITLE 16 BIT SQUARE ROOT ROUTINE
                ;
                ;
                ;
                ;    8080 SQUARE ROOT ROUTINE
                ;
                ;    16 BIT ARGUMENT, 16 BIT RESULT
                ;
                ;    ENTER WITH ARGUMENT IN HL
                ;
                ;    RESULT RETURNED IN HL, OR (OPTION 2, SEE BELOW) IN
                ;    TWO'S COMPLEMENT FORM IN BC
                ;
                ;
0000 110000     SQRT:   LXI     D,0000H ;  INITIALIZE EXTENDED ARGUMENT,
0003 01FFFF             LXI     B,0FFFFH;  COMPLEMENTED RESULT, AND
0006 3E10               MVI     A,10H   ;  LOOP COUNT.
0008 F5                 PUSH    PSW     ;  SAVE LOOP COUNT.

  09 EB         LOOP:   XCHG
000A 7C                 MOV     A,H
000B 65                 MOV     H,L
000C 6A                 MOV     L,D
000D 29                 DAD     H       ;  DOUBLE LEFT SHIFT OF
000E 17                 RAL             ;  REGISTER STRING DEHL
000F 29                 DAD     H       ;  (CONTAINING CURRENT REMAINDER).
0010 17                 RAL
0011 EB                 XCHG
0012 29                 DAD     H
0013 29                 DAD     H
0014 5A                 MOV     E,D
0015 57                 MOV     D,A

0016 F1                 POP     PSW     ;  RETRIEVE LOOP COUNT.
0017 E5                 PUSH    H       ;  SAVE LOW ORDER 16.
0018 60                 MOV     H,B     ;  MOVE PARTIAL RESULT
0019 69                 MOV     L,C     ;  TO HL.
001A 29                 DAD     H       ;  SHIFT LEFT, SHIFTING IN A ONE.
001B 2C                 INR     L
001C 44                 MOV     B,H     ;  SAVE SHIFTED PARTIAL RESULT.
001D 4D                 MOV     C,L
001E 29                 DAD     H       ;  SHIFT COPY OF PARTIAL RESULT AGAIN.
001F 2C                 INR     L
0020 19                 DAD     D       ;  SUBTRACT SHIFTED PARTIAL RESULT
                                        ;  FROM HIGH 16 OF CURRENT REMAINDER
                                        ;  (BY ADDING COMPLEMENT).
0021 D22600             JNC     NOGO    ;  TEST SUBTRACTION RESULT.
0024 0D                 DCR     C       ;  TACK A ZERO ONTO COMPLEMENTED
```

```
                        ;                                   ; PARTIAL RESULT.
0025 EB                 XCHG                                ; REPLACE HIGH ORDER 16 OF CURRENT
                        ;                                   ; REMAINDER WITH SUBTRACTION RESULT.
0026 3D         NOGO:   DCR     A                           ; DECREMENT AND TEST LOOP COUNT.
0027 CA2F00             JZ      DONE
002A E1                 POP     H
002B F5                 PUSH    PSW                         ; SAVE LOOP COUNT.
002C C30900             JMP     LOOP
002F E1         DONE:   POP     H                           ; REMOVE UNUSED DATA FROM STACK.

0030 78                 MOV     A,B
0031 2F                 CMA
0032 67                 MOV     H,A                         ; COMPLEMENT RESULT AND STORE IN HL.
0033 79                 MOV     A,C
0034 2F                 CMA                                 ; **OPTION 2:
0035 6F                 MOV     L,A                         ;   TO RETURN RESULT IN TWO'S
0036 C9                 RET                                 ;   COMPLEMENT FROM IN BC,
                        ;                                   ;   SUBSTITUTE THE FOLLOWING:
                        ;                                   ; DONE: POP     H
                        ;                                   ;       INX     B
                        ;                                   ;       RET
   00                   END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC10

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Floating Point Square Root

Function

Section 5: Math and Numerical Manipulation Programs

Operations performed are:
1) Test for negative argument (overflow set and return)
2) Computation of the square root for positive arguments.

Required Hardware

None

Required Software

Floating Point Math Package - BC1

Input Parameters

The floating point accumulator contains the operand x in floating point format. (See floating point math package documentation for further details)

Output Results

The floating point accumulator contains the value of the square root of x in floating point format.

Registers Modified: All registers affected	**Assembler/Compiler Used:** Intellec 8 Macro Assembler
RAM Required: 13 bytes	**Programmer:** Bob Eichenlaub
ROM Required: 108 bytes	**Company:** John Fluke Mfg. Co., Inc.
Maximum Subroutine Nesting Level: 4	**Address:** P.O. Box 43210 Mountlake Terrace, Wa 98043

FLOATING POINT SQUARE ROOT

A FLOATING POINT NUMBER CAN ALWAYS BE WRITTEN IN THE FORM

```
X =   M * 2**2K       IF THE EXPONENT IS EVEN OR
      M * 2 *2**2K    IF THE EXPONENT IS ODD
```

WHERE 0.5 < ABS(M) < 1.0 AND K = 0,+1,-1,+2,-2, ...

THUS THE SQUARE ROOT OF X IS GIVEN BY

```
(1)  SQRT(X) = SQRT(M) * 2**K                FOR EVEN EXPONENT OR

             = SQRT(M) * SQRT(2) * 2**K      FOR ODD EXPONENT
```

TO CALCULATE THE SQUARE ROOT, HERON'S ITERATIVE PROCESS IS USED. THIS ENTAILS MAKING AN INITIAL APPROXIMATION, DENOTED YO, TO SQRT(M) AND COMPUTING THE ITERATIVE VALUES:

```
Y(I+1) = (Y(I) + M/Y(I))/2  ,  I=0,1,2, ...
```

UNTIL Y(N) IS SUCH THAT THE DISCREPANCY IS INVISIBLE TO THE MACHINE I.E.

```
ABS(Y(N) - SQRT(M)) < EPSILON
```

FOR THE FLOATING POINT PACKAGE, EPSILON = 2**-24 OR ABOUT 6.2 * 10**-7 , SINCE THE RESOLUTION OF THE FLOATING POINT MANTISSA IS 24 BITS OF SIGNIFICANCE (I.E. 7.2 DIGITS OF SIGNIFICANCE).

HERON'S PROCESS IS DERIVED AS A SPECIAL CASE OF NEWTON'S METHOD AS APPLIED TO THE EQUATION

```
Y = SQRT(X)
```

THE INITIAL APPROXIMATION YO IS COMPUTED AS

```
YO = 0.41730759 + 0.59016206 * M ON THE INTERVAL [ 0.5,1.0 ] .
```

TWO ITERATIONS ARE THEN REQUIRED TO OBTAIN SQRT(M). THEN SQRT(X) CAN BE COMPUTED FROM EQUATION (1).

THE ROUTINE TAKES APPROXIMATELY 15 MILLISECONDS TO COMPUTE A TYPICAL SQUARE ROOT (I.E. TWO DIVIDES, ONE OR TWO MULTIPLIES, THREE ADDS, AND OTHER ASSORTED OPERATIONS) ON AN INTELLEC 8 / MOD 80 SYSTEM.

```
                        ;REF. NO. BC10
                        ;PROGRAM TITLE FLOATING POINT SQUARE ROOT
                        ;       ////////////////////////////////////////
                        ;
                        ;                   SQUARE ROOT SUBROUTINE
                        ;
                        ;                         8/30/75
                        ;
                        ;                      BOB EICHENLAUB
                        ;
                        ;       ////////////////////////////////////////
                        ;
                        ;       EQUATE TABLES FOR THE FLOATING POINT PACKAGE
                        ;         (SEE INTEL USER LIBRARY DOCUMENTATION)
                        ;
                        ;
0B00                    FR      EQU     0B00H           ;FLOATING POINT RAM PAGE
0300                    FS      EQU     0300H           ;FLOATING POINT CODE PAGE
                        ;
                        ;
0B2E                    OVER    EQU     FR+02EH
 B30                    ACCE    EQU     FR+030H
                        ;
038C                    FMUL    EQU     FS+08CH
0359                    FTEST   EQU     FS+059H
03D7                    FADD    EQU     FS+0D7H
03D4                    FSUB    EQU     FS+0D4H
034D                    FCHS    EQU     FS+04DH
036E                    FLOD    EQU     FS+06EH
03B4                    FDIV    EQU     FS+0B4H
033E                    FSTR    EQU     FS+03EH
032F                    FINIT   EQU     FS+02FH
03CA                    FOVER   EQU     FS+0CAH
                        ;
                        ;
                        ;       MAINLINE TEST LOOP
                        ;
                        ;
1400                            ORG     1400H
                        ;
                        ;
                        MAINLINE:
1400 31002F                     LXI     SP,2F00H        ;SET STACK POINTER
1403 CD2F03                     CALL    FINIT           ;FLOATING POINT INITIALIZATION
1406 210920                     LXI     H,ARG
1409 CD6E03                     CALL    FLOD            ;LOAD ARGUMENT FOR SQRT
'40C CD1614                     CALL    FSQRT
 40F 210021                     LXI     H,RESULT        ;STORE SQRT IN RESULT
1412 CD3E03                     CALL    FSTR
1415 C7                         RST     0
```

```
                        ;
                        ;
                        ;       SQUARE ROOT SUBROUTINE
                        ;
                        ;       ON ENTRY:       F.P. ACCUMULATOR HAS SQRT ARGUMENT
                        ;
                        ;       ON EXIT:        F.P. ACCUMULATOR HAS THE SQUARE ROOT
                        ;
                        ;       ALL REGISTERS ARE AFFECTED
                        ;
                        ;
                        FSQRT:
1416 CD5903             CALL    FTEST           ;TEST SIGN OF ARGUMENT
1419 C8                 RZ                      ;RETURN IF ARG. = 0
141A FACA03             JM      FOVER           ;RETURN ERROR IF ARG. < 0
141D 320020             STA     TEXP            ;STORE EXPONENT
1420 3E80               MVI     A,80H
1422 32300B             STA     ACCE            ;STORE IN EXPONENT
1425 210120             LXI     H,TMANTISSA
1428 CD3E03             CALL    FSTR            ;STORE IN TMANTISSA
                        ;
                        ;       INITIAL APPROXIMATION:
                        ;         TMP1 := 0.41730759 + 0.59016206*M
                        ;
142B 217714             LXI     H,IAMULT
142E CD8C03             CALL    FMUL
1431 217B14             LXI     H,IAADD
1434 CDD703             CALL    FADD
1437 210520             LXI     H,TMP1
143A CD3E03             CALL    FSTR            ;STORE IN TMP1
                        ;
                        ;       NEWTON'S METHOD OF ITERATION TO
                        ;         THE APPROXIMATE VALUE
                        ;         OF THE SQRT OF M
                        ;
143D CD5F14             CALL    FSQ1            ;FIRST ITERATION
1440 210520             LXI     H,TMP1          ;STORE RESULT
1443 CD3E03             CALL    FSTR            ;  IN TMP1
1446 CD5F14             CALL    FSQ1            ;SECOND ITERATION
                        ;
                        ;
                        ;       RESTORE RANGE TO OBTAIN THE FINAL RESULT
                        ;
                        ;
1449 3A0020             LDA     TEXP
144C D680               SUI     80H             ;UNBIAS THE EXPONENT
144E 1F                 RAR                     ;DIVIDE EXPONENT BY 2 (SQRT FNC)
144F F5                 PUSH    PSW             ;SAVE CARRY BIT (ODD OR EVEN)
1450 C680               ADI     80H             ;RESTORE BIAS TO EXPONENT
1452 32300B             STA     ACCE
```

```
1455 F1                 POP     PSW             ;RESTORE CARRY
1456 D25903             JNC     FTEST           ;JUMP IF EXPONENT WAS EVEN
1459 217F14             LXI     H,SQRT2
145C C38C03             JMP     FMUL            ;*SQRT(2) IF EXPONENT WAS ODD
                ;
                ;
                ;       SUBROUTINE FSQ1
                ;
                ;         THIS ROUTINE PERFORMS ONE NEWTON ITERATION
                ;         TO THE SQUARE ROOT FUNCTION
                ;
                FSQ1:
145F 210120             LXI     H,TMANTISSA     ;LOAD THE MANTISSA
1462 CD6E03             CALL    FLOD            ;  INTO F.P.  ACC.
1465 210520             LXI     H,TMP1          ;FORM M/TMP1
1468 CDB403             CALL    FDIV
146B 210520             LXI     H,TMP1
146E CDD703             CALL    FADD            ;FORM TMP1 + M/TMP1
1471 D601               SUI     1               ;DIVIDE BY 2 AND
1473 32300B             STA     ACCE            ;  STORE IN EXPONENT
 \76 C9                 RET                     ;RETURN TO MAIN SQRT ROUTINE
                ;
                ;       END OF SQUARE ROOT ALGORITHM
                ;
                ;
                ;       CONSTANTS NEEDED BY FSQRT
                ;
1477 801714EB   IAMULT:         DB      080H,017H,014H,0EBH     ;0.59016206
147B 7F55A956   IAADD:          DB      07FH,055H,0A9H,056H     ;0.41730759
147F 813504F3   SQRT2:          DB      081H,035H,004H,0F3H     ;1.41421356
                ;
                ;
                ;       SCRATCH STORAGE
                ;
2000                    ORG     2000H
2000            TEXP:           DS      1
                Y:
2001            TMANTISSA:      DS      4
2005            TMP1:           DS      4
2009            ARG:            DS      4
                ;
                ;
2100                    ORG     2100H
2100            RESULT:         DS      4
                ;
                ;
                ;       END OF SQUARE ROOT FUNCTION PACKAGE
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB24

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

SQRTF

Function

GENERATES 8 BIT ROOT OF 16 BIT NUMBER

Required Hardware

Required Software

Input Parameters

16 BIT NUMBER, MS BYTE IN B REGISTER, LS BYTE IN C REGISTER PRIOR TO CALLING SQRTF SUBROUTINE.

Output Results

8 BIT ROOT WRITTEN INTO A MEMORY LOCATION

Registers Modified: A,B,C,D,E,H,L	**Assembler/Compiler Used:** INTELLEC 8/MOD80 MACRO
RAM Required: 1 BYTE	**Programmer:** DENNIS GREEN
ROM Required: 66D BYTES	**Company:** RAYTHEON
Maximum Subroutine Nesting Level: NOT APPLICABLE	**Address:** MS: S3-52 HARTWELL ROAD, BEDFORD 01730

```
                        ;REF. NO. BB24
                        ;PROGRAM TITLE SQRTF
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
0340                    ORG     0340H
                                        ;B IS MSBYTE.
                                        ;C IS LSBYTE.
0421            ROOT    EQU     0421H
0340 AF         SQRTF:  XRA     A       ;CLR A & CARRY
0341 67                 MOV     H,A
0342 57                 MOV     D,A
0343 322104             STA     ROOT
0346 6F                 MOV     L,A     ;SET
0347 2C                 INR     L       ;L
0348 2C                 INR     L       ;TO 2.
0349 C612               ADI     12H     ;SET COUNTER.
 ^4B 5F                 MOV     E,A
 ^4C 79         SQRT1:  MOV     A,C
034D 1D                 DCR     E       ;SETS FLAGS 'CEPT CARRY.
034E CA8203             JZ      ENDRT   ;EXIT LOOP AT 9TH PASS.
0351 17                 RAL             ;GET MSB OF LSBYTE IN
                                        ;CARRY.
0352 4F                 MOV     C,A     ;SAVE REMAINIG LSBYTE.
0353 78                 MOV     A,B     ;GET MSBYTE.
0354 17                 RAL             ;GET MSB IN CARRY
                                        ;& LSBYTE'S MSB TOA.
0355 47                 MOV     B,A     ;SAVE SHIFTED MSBYTE.
0356 7C                 MOV     A,H     ;
0357 17                 RAL             ;MSB OF B TO CARRY &
0358 67                 MOV     H,A     ;TO H, RIGHT JUSTIFIED.
0359 2D                 DCR     L       ;
035A C24C03             JNZ     SQRT1   ;REPEAT TIL 2 BITS
                                        ;ARE IN H.
035D 2C                 INR     L       ;RESET L
035E 2C                 INR     L       ;TO 2.
035F 14                 INR     D       ;GET AN INR IN
                                        ;BEFORE TEST.
0360 DA6703             JC      MINH    ;
0363 92                 SUB     D       ;IF PLUS REMAINDER.
0364 C36A03             JMP     SQRT2   ;
0367 14         MINH:   INR     D       ;
^368 14                 INR     D       ;
 69 82                  ADD     D       ;IF PLUS REMAINDER.
036A 67         SQRT2:  MOV     H,A     ;
036B 3A2104             LDA     ROOT    ;FLAGS STILL SET
```

```
                                                ;BY SUB OR ADD.
036E FA7703             JM      MINH1           ;
0371 B7                 ORA     A               ;CLR CARRY.
0372 17                 RAL                     ;
0373 3C                 INR     A               ;ROOT CHARACTER
                                                ;IS 1.
0374 C37903             JMP     SQRT3           ;
0377 B7         MINH1:  ORA     A               ;CLR CARRY
0378 17                 RAL                     ;ROOT CHARACTER
                                                ;IS 0.
0379 322104     SQRT3:  STA     ROOT            ;
037C 17                 RAL                     ;MAKE ROOM IN D
037D 17                 RAL                     ;FOR THE NEXT 2 BIT
037E 57                 MOV     D,A             ;BYTE FROM ALGORITHM.
037F C34C03             JMP     SQRT1           ;
0382 C9         ENDRT:  RET
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC11

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080 (use additional sheets if necessary)

Program Title

FLOATING POINT SQRT SUBROUTINE

Function

This Subroutine takes the square root of a number in floating point notation.

Required Hardware

Required Software

Intel's floating point package, BC1, assembled at 800 HEX with the scratchpad at 1000 HEX

Input Parameters

Location "VARI" must be the input argument in floating point notation.

Output Results

Location "XP" becomes the square root of whatever was in "VARI".

Registers Modified:	**Assembler/Compiler Used:** 8008 Assembler 2.0
RAM Required:	**Programmer:** Jack G. Ganssle
ROM Required:	**Company:** Neotec Corporation
Maximum Subroutine Nesting Level:	**Address:** 2431 Linden Lane Silver Spring, MD. 20910

98-034B

```
                        ;REF. NO. BC11
                        ;PROGRAM TITLE SUBROUTINE SQRT
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;       GQA 41A
                        ;
                        ;       SUBROUTINE SQRT
                        ; DESCRIPTION:
                        ;       THIS SUBROUTINE TAKES THE SQUARE ROOT
                        ;      OF A NUMBER BY USING THE FOLLOWING RECURSION
                        ;      FORMULA:
                        ;
                        ;      X(2)=.5(X(1)+N/X(1))
                        ;
                        ;      X(3)=.5(X(2)+N/X(2))
                        ;
                        ;      ECT. X(I) APPROACHES THE SQUARE ROOT
                        ;      AS I BECOMES LARGER. N IS THE INPUT NUMBER/
                        ;
                        ; INPUT PARAMETERS:
                        ;       THIS ROUTINE MUST BE ENTERED WITH THE NUMBER
                        ;      N IN LOCATION VARI.
                        ;
                        ; OUTPUT PARAMETERS:
                        ;       UPON EXIT, XP= THE SQUARE ROOT OF
                        ;      VARI.
                        ;
                        ;
                        ;       CODED JUNE 13,1975 (FSIDAY THE 13TH) YFNED
                        ;
2620                    BEGIN   EQU     2620H
1479                    VARI    EQU     1479H   ;INPUT AND .OUTPUT BUFFER
14A8                    CTR     EQU     14A8H   ;COUNTER BUF
14A9                    XP      EQU     CTR+1   ;GUESS BUFFER
086E                    LOD     EQU     86EH
08B4                    DIV     EQU     8B4H
08D7                    AD      EQU     8D7H
0B4B                    STR     EQU     0B4BH
00A9                    XPL     EQU     XP AND 0FFH
00A8                    CTRL    EQU     CTR AND 0FFH
0079                    VARIL   EQU     VARI AND 0FFH
2620                            ORG     BEGIN
2620 21A814                     LXI     H,CTR
2623 360B                       MVI     M,11    ;# REPEATS=11
2625 2C                         INR     L
2626 3681                       MVI     M,81H   ;INITIAL QUESS=ABOUT 1
```

```
2628 2C                 INR     L
2629 3600               MVI     M,0     ;THATS A +1, BOYS
262B 2E79       ITER:   MVI     L,VARIL ;FIGGER (N/X+X).5
262D CD6E08             CALL    LOD     ;ACC=N
2630 21A914             LXI     H,XP
2633 CDB408             CALL    DIV     ;ACC=N/X
2636 21A914             LXI     H,XP
2639 CDD708             CALL    AD      ;ACC=N/X+X
263C 21A914             LXI     H,XP
263F CD4B0B             CALL    STR     ;SAVE NEW GUESS
2642 2EA9               MVI     L,XPL
2644 46                 MOV     B,M
2645 05                 DCR     B       ;DIVY BY 2
2646 70                 MOV     M,B     ;NEW GUESS=.5(N/X+X)
2647 2D                 DCR     L
2648 46                 MOV     B,M
2649 05                 DCR     B       ;DEC (11-#ITERATIONS)
264A C8                 RZ
264B 70                 MOV     M,B
264C C32B26             JMP     ITER
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC12

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

FLOATING POINT SQUARE ROOT ROUTINE - FAST

Function

Calculate square root of a floating point number by Heron's method.
Execution time ≤ 50 ms for any number.

Required Hardware

None.

Required Software

Intel Floating Point Math Package - BC1
Floating Point Format Conversion - BC2

Input Parameters

Assumes number in floating point accumulator.

Output Results

Returns square root in floating point accumulator.

Registers Modified: All	Assembler/Compiler Used: Intellec 8/MOD 80 Ver 3.0
RAM Required: 16 Bytes	Programmer: W. Moritz/L. Mace
ROM Required: 161 Bytes	Company: Center for Bioengineering
Maximum Subroutine Nesting Level:	Address: U. of Wash., Seattle, WA. 98195

```
                ;REF. NO. BC12
                ;PROGRAM TITLE FAST FLOATING POINT SQUARE ROOT ROUTINE
                ;
                ;
                ;
                ;
                ;*****************************************************************
                ;
                ;
                ;
                ;       SQUARE ROOT SUBROUTINE ---SQRT
                ;       PERFORMS SQUARE ROOT OF A FLOATING POINT
                ;       NUMBER ASSUMES NUMBER IS IN FLOATING
                ;       POINT ACCUMULATOR.  THE SQUARE ROOT
                ;       IS RETURNED IN THE FLOATING POINT
                ;       ACCUMULATOR.  THIS ROUTINE REQUIRES
                ;       THE AVAILABILITY OF INTEL'S
                ;       FLOATING POINT PACKAGE AND
                ;       CALLS SUBROUTINES FROM THAT PACKAGE.  THE
                ;       FLOATING POINT ACCUMULATOR IS LOCATED AT
                ;       LOCATIONS 0B30H TO 0B34H.
                ;
                ;                 UNIVERSITY OF WASHINGTON
                ;               CENTER FOR BIOENGINERRING
                ;              W.  MORITZ AND L.  MACE   MAY 15.  1975
                ;
                ;*****************************************************************

                ;
                ;SUBROUTINES USED
                ;
03D7            AD      EQU     03D7H   ;FL.  PT.  ADD SUBROUTINE
                                        ;   ENTRY POINT
03D4            SB      EQU     03D4H   ;FL.  PT.  SUBTRACT SUBROUTINE
                                        ;   ENTRY POINT
038C            MUL     EQU     038CH   ;FL.  PT.  MULTIPLY SUBROUTINE
                                        ;   ENTRY POINT
03B4            DIV     EQU     03B4H   ;FL.  PT.  DIVIDE SUBROUTINE
                                        ;   ENTRY POINT
036E            LOD     EQU     036EH   ;FL.  PT.  LOAD ENTRY POINT
0359            TST     EQU     0359H   ;FL.  PT.  TEST ENTRY POINT
033E            STR     EQU     033EH   ;FL.  PT.  STORE ENTRY POINT
0346            ZRO     EQU     0346H   ;FL.  PT.  ZERO ENTRY POINT

                ;
                ;SCRATCHPAD ALLOCATIONS
                ;
0B40            SCR1    EQU     0B40H
0B44            SCR2    EQU     0B44H
```

```
0B48            SCR3    EQU     0B48H
0B4C            SCR4    EQU     0B4CH
                ;
                ;SQRT ROUTINE   HERON'S METHOD
                ;
0900                    ORG     0800H
                ;
                ;LOAD ACC AND SCRATCH WITH NUMBER
                ;
0800 CD5903     SQRT:   CALL    TST     ;LOAD FLT PT ACC
0803 21400B             LXI     H,SCR1
0806 CD3E03             CALL    STR     ;STORE ORIGINAL NUMBER
                ;
                ;TEST FOR ILLEGAL ENTRY
                ;
0809 21400B             LXI     H,SCR1
080C CDD703             CALL    AD      ;SET CONTROL BITS
080F DA9608             JC      OFOUT   ;OUT IF OVERFLOW
0812 CA9608             JZ      OFOUT   ;OUT IF ZERO
0815 FA9608             JM      OFOUT   ;OUT IF NEGATIVE
                ;
                ;STORE ORIGINAL EXPONENTT       ;REPLACE WITH ZERO
0818 21400B             LXI     H,SCR1
081B CD6E03             CALL    LOD     ;LOAD EXP INTO REG A
081E 214C0B             LXI     H,SCR4
0821 77                 MOV     M,A     ;STORE EXP IN SCR4
0822 3E80               MVI     A,80H   ;ZERO OUT EXPONENT
0824 21400B             LXI     H,SCR1
0827 CD3E03             CALL    STR     ;STORE NEW NUMBER
082A 21440B             LXI     H,SCR2
082D CD3E03             CALL    STR     ;STORAGE FOR LAST GUESS
0830 21480B             LXI     H,SCR3
0833 CD3E03             CALL    STR     ;STORAGE FOR NEW GUESS
                ;
                ;FIND SQRT OF MANTISSA
                ;BEGIN ITERATION
                ;
0836 21480B     LOOP:   LXI     H,SCR3
0839 CD6E03             CALL    LOD     ;PUT NEW GUESS IN---
083C 21440B             LXI     H,SCR2
083F CD3E03             CALL    STR     ;---OLD GUESS SLOT
0842 21400B             LXI     H,SCR1
0845 CD6E03             CALL    LOD     ;START INTERATION LOAD NUMBER
0848 21440B             LXI     H,SCR2
084B CDB403             CALL    DIV     ;DIVIDE BY LAST GUESS
084E 21440B             LXI     H,SCR2
0851 CDD703             CALL    AD      ;ADD LAST GUESS
0854 21440B             LXI     H,SCR2
0857 CDB403             CALL    DIV     ;DIVIDE BY 2
085A 21480B             LXI     H,SCR3
```

```
085D CD3E03            CALL    STR     ;STORE NEW GUESS
0860 21440B            LXI     H,SCR2
0863 CDD403            CALL    SB      ;SUBTRACT LAST FROM NEW GUESS
0866 CA6C08            JZ      SROUT   ;FINISHED IF FINAL ITERATION
0869 C33608            JMP     LOOP    ;JUMP TO NEXT ITERATION
                       ;
                       ;FIND SQRT OF EXPONENT
                       ;PREPARE FLT PT ACC FOR RETURN
                       ;
086C AF        SROUT:  XRA     A       ;RESET CARRY
086D 214C0B            LXI     H,SCR4
0870 7E                MOV     A,M     ;LOAD ORIGINAL EXP INTO A
0871 E601              ANI     01H     ;CHECK EVEN/ODD EXP
0873 CA8A08            JZ      EVEN
0876 7E        ODD:    MOV     A,M
0877 DE01              SBI     01H     ;MAKE EXP EVEN
0879 1F                RAR             ;DIVIDE BY TWO
087A C640              ADI     40H     ;ADJUST BIAS
087C 21480B            LXI     H,SCR3
087F 77                MOV     M,A     ;STORE NEW EXPONENT
0880 CD6E03            CALL    LOD
0883 219A08            LXI     H,SQRT2 ;ADDR, OF SQRT2 CONST.
0886 CD8C03            CALL    MUL     ;SQRT2*SQRTX
0889 C9                RET             ;RETURN TO CALLER
088A 7E        EVEN:   MOV     A,M
088B 1F                RAR             ;DIVIDE BY TWO
088C C640              ADI     40H     ;ADJUST BIAS
088E 21480B            LXI     H,SCR3
0891 77                MOV     M,A     ;STORE NEW EXPONENT
0892 CD6E03            CALL    LOD
0895 C9                RET             ;RETURN TO CALLER
0896 CD4603    OFOUT:  CALL    ZRO     ;ZERO OUT ACC FOR RETURN
0899 C9                RET
089A 813504F3  SQRT2:  DB      81H,35H,04H,0F3H
089E 82000000  TW0:    DB      82H,00H,00H,00H
0000                   END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. BB25

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

NATURAL LOGARITHM

Function

Computes the natural logarithm of a number between 1 and 65535

Required Hardware

Basic 8080 System

Required Software

70 Instructions
(+12 instructions for MULTIplication Subroutine)

Input Parameters

Number to be converted in H and L

Output Results

Mantissa of LN Z in A

Characteristic of LNZ in H and L, expressed in 1/65536

(Fixed point with format (A). (H) (L)

Registers Modified: PSW, H, L	**Assembler/Compiler Used:** 1
RAM Required: 4 Bytes	**Programmer:** MAC 80
ROM Required: 128 Bytes for LN 20 Bytes for MUL	**Company:** B. Hauert Battelle Institute
Maximum Subroutine Nesting Level:	**Address:** 7 Route de Drize 1277 Carouge Geneva, SWITZERLAND

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC13

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080 (use additional sheets if necessary)

Program Title

FLOATING POINT LOG SUBROUTINE - COMMON LOGARITHMS

Function

This subroutine takes the log to any integer base of any positive floating point number.

Required Hardware

Required Software

Intel's floating point package - BC1
(Note: In the supplied listing, the package was assembled at 800 HEX. The RAM scratchpad starts at 1000 HEX.)

Input Parameters

Location "LOG" is the input argument in floating point notation.

Location "BASE" is the natural log of the base in floating point notation.

Output Results

Location "ANS" becomes the result in floating point notation.

Registers Modified: All	**Assembler/Compiler Used:** 8008 Assembler 2.0
RAM Required:	**Programmer:** Jack G. Ganssle
ROM Required:	**Company:** Neotec Corporation
Maximum Subroutine Nesting Level:	**Address:** 2431 Linden Lane Silver Spring, MD. 20910

```
                ;REF, NO. BC13
                ;PROGRAM TITLE SUBROUTINE LOG
                ;
                ;
                ;
                ;
                ;
                ;       SUBROUTINE LOG
                ;
                ;
                ; DESCRIPTION:
                ;        THIS SUBROUTINE CALCULATES THE
                ;       LOG TO ANY BASE OF A FLOATING
                ;       POINT NUMBER IN MEMORY USING
                ;       THE FOLLOWING FORMULAE:
                ;
                ; LOG(BASE B)(X)=LN(X)/LN(B)
                ;
                ; LN(X)=(LN(M)++N*(.6931472)]
                ;
                ;  WHERE X=M*2**N
                ;   AND 1<M<2
                ;    S0 2**N<X<2**(N+1)
                ;
                ;   AND
                ; LN(M)=2*Z(1+(Z**2/3)*(1+15*Z**2/23)
                ;
                ;   WHERE Z=(M-1)/(M+1)
                ;
                ;        UPON ENTRY TO THIS ROUTINE,
                ;       THE LOCATION SPECIFIED BY 'LOG'
                ;       MUST CONTAIN X, THE NUMBER TO
                ;       BE LOGIFIED. NO(BASE B) (X)
                ;       WILL APPEAR IN LOCATION 'ANS'.
                ;
                ;        THE BASE IS DETERMINED BY SETTING ADDRESS
                ;       'BASE' EQUAL TO THE FLOATING
                ;       POINT VALUE OF LN(BASE). THUS, FOR
                ;       COMMON LOGS BASE WOULD EQUAL THE
                ;       FLOATIOG POINT REPRESENTATION
                ;       OF 2.302585093.
                ;
                ;       CODE MAY 5, 1975   YFNED
                ;
0100            BEGIN   EQU     100H
08D7            AD      EQU     8D7H    ; FP ADD
088C            MUL     EQU     88CH    ; FP MULTIPLY
083E            STR     EQU     83EH    ; STORE FP ACCUM
086E            LOD     EQU     86EH    ; FP LOAD
```

```
08B4            DIV     EQU     8B4H    ; FP DIVIDE
0B00            FLT     EQU     0B00H   ; FP FLOAT
08D4            SB      EQU     8D4H    ; FP SUBTRACT
1100            BUF     EQU     1100H   ; BUFFER AREA
1150            LOG     EQU     1150H   ; LOCATION OF ARGUMENT
1154            ANS     EQU     1154H   ; LOCATION OF ANSWER
0100                    ORG     BEGIN
                ;
                ;  CALCULATE N SUCH THAT 2**N<=X<=2**(N+1)
                ; THEN FIGGER M, M=X/2**N.
                ;
0100 215011             LXI     H,LOG
0103 7E                 MOV     A,M     ; A= X EXPONENT
0104 D671               SUI     71H     ; MAKE 2**N BINARY POINT
0106 5F                 MOV     E,A     ; E=BINARY POINT
0107 AF                 XRA     A
0108 4F                 MOV     C,A
0109 57                 MOV     D,A
010A 0601               MVI     B,1     ; ABCD=UNJUSTIFIED 1
010C CD000B             CALL    FLT     ; ACCUM=2**N
010F 210011             LXI     H,BUF
0112 CD3E08             CALL    STR     ; SAVE 2**N
0115 215011             LXI     H,LOG
0118 CD6E08             CALL    LOD     ; ACCUM=X
011B 210011             LXI     H,BUF
011E CDB408             CALL    DIV     ; ACCUM=X/2**N=M
0121 21A801             LXI     H,FP1
0124 CDD708             CALL    AD      ; ACCUM=M+1
0127 210011             LXI     H,BUF
012A CD3E08             CALL    STR     ; SAVE M+1
012D 21AC01             LXI     H,FP2
0130 CDD408             CALL    SB      ; ACCUM=M-1
0133 210011             LXI     H,BUF
0136 CDB408             CALL    DIV     ; ACCUM=Z=(M-1)/M+1)
0139 210011             LXI     H,BUF
013C CD3E08             CALL    STR     ; SAVE Z
013F CD8C08             CALL    MUL     ; ACCUM=Z**2
0142 2E00               MVI     L,BUF AND 0FFH
0144 210411             LXI     H,BUF+4
0147 CD3E08             CALL    STR     ; SAVE Z**2
014A 21B401             LXI     H,FP15
014D CD8C08             CALL    MUL     ; (Z**2)/15
0150 21B801             LXI     H,FP23
0153 CDB408             CALL    DIV     ; (15*Z**2)/23
0156 21A801             LXI     H,FP1
0159 CDD708             CALL    AD      ; 1+15*Z**2/23
015C 210411             LXI     H,BUF+4
015F CD8C08             CALL    MUL     ; (1+15*Z**1/3)*Z**2
0162 21B001             LXI     H,FP3
0165 CDB408             CALL    DIV     ; T=(1+15*Z**2/3)*Z**2/3
```

```
0168 21A801           LXI     H,FP1
016B CDD708           CALL    AD        ; 1+T
016E 210011           LXI     H,BUF
0171 CD8C08           CALL    MUL       ; (1+T)*Z
0174 21AC01           LXI     H,FP2
0177 CD8C08           CALL    MUL       ; (1+T)*Z*2=LN(M)
017A 210011           LXI     H,BUF
017D CD3E08           CALL    STR
0180 215011           LXI     H,LOG
0183 7E               MOV     A,M       ; A=X EXPONENT
0184 D681             SUI     81H       ; A=N
0186 57               MOV     D,A
0187 AF               XRA     A
0188 47               MOV     B,A
0189 4F               MOV     C,A
018A 1E20             MVI     E,20H     ; BINARY POINT-32
018C CD000B           CALL    FLT       ; ACCUM=N
018F 21C001           LXI     H,LN2
0192 CD8C08           CALL    MUL       ; ACC=N*.6931472
0195 210011           LXI     H,BUF
0198 CDD708           CALL    AD        ; ACCUM=LN(X)
  9B 21BC01           LXI     H,BASE
↵↓9E CDB408           CALL    DIV       ; ACCUM=LOG(BASE B)(X)
01A1 215411           LXI     H,ANS
01A4 CD3E08           CALL    STR       ; SAVE ANSWER
01A7 C9               RET               ; THAT'S ALL, FOLKS!
01A8 81000000 FP1:    DB      81H,0,0,0         ;1.00
01AC 82000000 FP2:    DB      82H,0,0,0         ;2.00
01B0 82400000 FP3:    DB      82H,40H,0,0       ;3.00
01B4 84700000 FP15:   DB      84H,70H,0,0,      ;15.00
01B8 85380000 FP23:   DB      85H,38H,0,0,      ;23.00
01BC 82135D8E BASE:   DB      82H,13H,5DH,8EH ; LN(10)
01C0 80317218 LN2:    DB      80H,31H,72H,18H ; LN(2)
0000                  END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB26

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080 (use additional sheets if necessary)

Program Title

Approximating Routine

Function

To solve functions such as the log, the antilog, the sine, and the tangent function. The program given is set up to solve the antilog (base 2) function.

Required Hardware

None

Required Software

Various uses may require other routines. For example, to use the routine for the antilog routine, a double register left shift routine is required.

Input Parameters

For the antilog routine, the input is the mantissa, as an integral multiple of 1/256.

For the Sine routine, the input is the angle, as integral multiple of 90°/256.

Output Results

The antilog routine returns the fractional part of $2^{(X/256)}$ where X is the input.

The Sine routine returns the integral value of 256 * Sine (X) where X is the input.

Registers Modified: A, C, D, H and L	**Assembler/Compiler Used:** MAC 8 VER 2.3
RAM Required: NONE	**Programmer:** W.E. BAKER
ROM Required: 59 bytes	**Company:** BELL TELEPHONE LABORATORIES 1G-625
Maximum Subroutine Nesting Level: ONE	**Address:** HOLMDEL, NEW JERSEY 07733

The approximation routine solves equations of the form Y = F(X) where X and Y are 8 bit integers. The routine contains a table of values of F(16·N) for N = 0, 1, 2, ---, 16. For any given input, X, the routine calculates an N and a D such that X = 16·N+D. It then approximates the value of F(X) as

$$F(X) = F(16\cdot N) + \frac{D}{16}\,[\,F(16(N+1)) - F(16\cdot N)\,].$$

ANTILOG FUNCTION

The routine can be used to solve the equation $y = 2^x$ where

$$8 > x \geq 0$$

by using the substitution

$$x = N + \frac{X}{256}\ .$$

Working only with the term $\frac{X}{256}$ use the routine to solve the equation

$$\frac{Y}{256} = 2^{\frac{X}{256}} - 1.$$

Assume a double register left shift routine which operates on the register pair B, C. Load 1 into B, Y into C, and shift left N times. The solution y is given by

$$y = B + \frac{C}{256}\ .$$

Sine Function

The routine can be used for solving the equation $y = \sin(x)$ by defining the input argument, X, as $X = 256\cdot\frac{x}{90^\circ}$ The values of the table are the integer values of $256\cdot\sin(16\cdot N\cdot\frac{90^\circ}{256})$ for N = 0, 1, 2, ---, 16. The result y is given with 8 bit accuracy by y = Y/256.

Restrictions

To use the approximating routine, the function must meet the following criteria:

1. It must be monotonic increasing
2. Input and output must be expressible as integers between 0 and 255.
3. $F(16N + 16) - F(16N) < 32$ for $0 \leq N \leq 15$.

Test Program

The program TEST will generate the 256 values of ALOG for inputs from 0 to 255 and will store them on page 11. These results can then be compared to the contents of Table 1, which were calculated by a BASIC program. The maximum error in any location should be ±1.

```
                        ;REF. NO. BB26
                        ;PROGRAM TITLE APPROXIMATING ROUTINE
                        ;
                        ;
                        ;
                        ;
0800                            ORG     4000Q
0800 2E2B               ALOG:   MVI     L,053Q  ;POINT TO
0802 2608               AP1:    MVI     H,010Q  ;F(1)
0804 07                 AP2:    RLC
0805 07                         RLC
0806 07                         RLC
0807 07                         RLC
0808 4F                         MOV     C,A     ;C=16*DEL+N
0809 E6F0                       ANI     360Q
080B 57                         MOV     D,A     ;D=16*DEL
080C 79                         MOV     A,C
080D E60F                       ANI     017Q    ;A=N
080F 85                         ADD     L
0810 6F                         MOV     L,A     ;L POINTS TO F(N+1)
0811 7E                         MOV     A,M     ;A=F(N+1)
0812 2D                         DCR     L
0813 96                         SUB     M       ;A=F(N+1)-F(N)
0814 6E                         MOV     L,M     ;L=F(N)
0815 2605                       MVI     H,5
0817 A7                         ANA     A
0818 1F                 AP3:    RAR
0819 D21E08                     JNC     AP4
081C A7                         ANA     A       ;CLEAR CARRY
081D 82                         ADD     D
081E 25                 AP4:    DCR     H
081F C21808                     JNZ     AP3
0822 1F                         RAR
0823 1F                         RAR
0824 1F                         RAR
0825 1F                         RAR
0826 E61F                       ANI     037Q
0828 85                         ADD     L
0829 C9                         RET
082A 000B1723                   DB      0,11,23,35
082E 303E4C5B                   DB      48,62,76,91
0832 6A7A8B9C                   DB      106,122,139,156
0836 AFC2D6EA                   DB      175,194,214,234
083A 00                         DB      0
0000                            END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB27

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

SIN X, COS X SUBROUTINE

Function

Generates Sine or Cosine accurate to 8 bits of an input angle that is accurate to 8 bits. Uses a Chebyshev Economization of Taylor Series for Cosine X. Sine X is generated by complementing the angle X with respect to 90° (x' = 90° -X) and then taking Cosine X.

Required Hardware

Machine line and configuration for cross products.

Required Software

Binary Multiplication Loop given in Intel Specification Bulletin on 8080 CPU (MCS-064-474/25K), page 17. Routine must be modified to act as a subroutine and is shown in listing.

Input Parameters

Subroutine is entered with call COS X or call SINX with X in register A.
For $0° \leq \text{Angle} < 90°$
Then $0H \leq X \leq FFH$

Output Results

Subroutine is exited with the Cos of X (Sine of X) in register A.
For $0 \leq \text{Cos} \leq 1.00$

$0H \leq \text{Cos } X \leq FFH$

Registers Modified:	Assembler/Compiler Used:
All register modified	8080 Macro Assembler Version 2.0
RAM Required:	**Programmer:**
Minimum (256 Bytes)	Billy R. Slater
ROM Required:	**Company:**
25H	Forney Engineering
Maximum Subroutine Nesting Level:	**Address:** P.O. Box 189 Addison, Texas 75001
Two	

```
                ;REF. NO. BB27
                ;PROGRAM TITLE SIN X, COS X SUBROUTINES
                ;
                ;
                ;
                ;
                ;COS X  ( CHEBYSHEV ECONOMIZATION OF
                ;        TAYLOR EXPANSION )
                ;
                ;CALLING SEQUENCE        CALL    COSX
                ;                        A =     INPUT X
                ;                        OUTPUT - A, COS X
                ;
                ; 0<= X <=255.  FOR 0'<= ANGLE <= 90'
                ;
                ;COS X =255/255-2*159/255*X^2+60*X^4
                ;
                ;
                ;
1000                    ORG     1000H
  )00 2F        SINX:   CMA                     ;FORM SINX=COS(90'-X)
1001 110000     COSX:   LXI     D,0             ;CLEAR D&E
1004 5F                 MOV     E,A
1005 CD2510             CALL    MULT            ;SQUARE X
1008 5C                 MOV     E,H
1009 1600               MVI     D,0             ;CLEAR D
100B D5                 PUSH    D               ;SAVE SQUARE X
100C 7B                 MOV     A,E
100D CD2510             CALL    MULT            ;RAISE X TO
                                                ;FOURTH POWER
1010 5C                 MOV     E,H
1011 1600               MVI     D,0
1013 3E3C               MVI     A,60
1015 CD2510             CALL    MULT            ;FORM 60*X^4
1018 4C                 MOV     C,H             ;AND SAVE
1019 D1                 POP     D               ;GET X SQUARED
101A 3E9F               MVI     A,159
101C CD2510             CALL    MULT            ;FORM 159*X^2
101F 3EFF               MVI     A,255
1021 94                 SUB     H               ;FORM 255-159*X^2
1022 81                 ADD     C               ;ADD +60*X^4
1023 94                 SUB     H               ;SUBT 159*X^2
1024 C9                 RET
                ;
                ;
                ;MULTIPLY SUBROUTINE
                ;
                ;CALLING SEQUENCE        CALL    MULT
                ;                        A =MULTIPLIER
```

```
                        ;                               D&E = MULTIPLICAN
                        ;
                        ;                               OUTPUT = H&L (PRODUCT)
                        ;
                        ;
                        ;
1025 210000     MULT:   LXI     H,0     ;INIT PARTIAL PROD TO 0
1028 0608               MVI     B,8     ;8>B TO CONTROL LOOP
102A 29         LOOP:   DAD     H       ;SHIFT PARTIAL TO LEFT
                                        ;INTO CARRY
102B 17                 RAL             ;ROTATE MULTIPLIER BIT
                                        ;TO CARRY
102C D23210             JNC     DEC     ;TEST MULTIPLIER
                                        ;AT CARRY
102F 19                 DAD     D       ;ADDMULTIPLICAND TO
                                        ;PARTIAL PRODUCT
                                        ;IF CARRY = 1
1030 CE00               ACI     0
1032 05         DEC:    DCR     B       ;DECREMENT B
                                        ;LOOP COUNTER
1033 C22A10             JNZ     LOOP    ;TEST TO SEE IF
                                        ;B = 0 TO ITERATE
                                        ;8 TIMES
1036 C9                 RET
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB28

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

INTEGRATION ROUTINE - RMSTF

Function

To calculate the integration "T" of any continuous function "F(x)" between two limits "a" and "b".

$$T = \frac{1}{b-a} \int_{x=a}^{x=b} F(x)\,dx$$

Required Hardware

None

Required Software

PL/M Compiler

Input Parameters

The integrated function is given by (N+1) equally spaced points stored sequentially in contiguous memory locations starting with "F(a)" and ending with "F(b)".

The input arguments are a pointer to the starting address, and the number "N" which represents the number of the interval sub-divisions.

Output Results

"N" should be representable in the form $N = 2^m$; where "m" is an integer number less or equal to 7. Each of the function ordinates can be any integer from 0 to 32787.

The procedure returns the calculated value of the integration with an error proportional to the "2(m + 1)"th power of the smallest interval subdivision "H_m".

All other errors corresponding to the smaller powers of "H_m" are completely eliminated.

Registers Modified:	Assembler/Compiler Used:
ALL	8080 PLM1, PLM2 VERS 1.3
RAM Required:	**Programmer:**
37 BYTES	M.A. Madkour
ROM Required:	**Company:**
947 BYTES	ATOMIC ENERGY ESTAB., (EGYPT)
Maximum Subroutine Nesting Level:	**Address:**
2	ANSHASS, EGYPT

Explanation of RMSTF

The integration is performed following the Romberg-Stiefel algorithm[1]. Based on the simple trapizoidal rule, and using any number of equal subdivisions "NL", the integration "T" can be approximated by:

$$T(L) = (1/2\ (F_a + F_b) + \sum_i F_i)/NL \qquad .....(1)$$

where F_i stands for all intermediate ordinates of $F(x)$.

The procedure starts by calculating a set of "m + 1" approximations T(0) to T(m) starting by considering a single subdivision whose width is the entire integration interval (b - a), and then using the suitable intermediate points such that the number of subdivisions is doubled every next calculation until ending with the given "N" smallest ones having width of of $H_m = (b - a)/N$ each. The error in the above approximations is proportional to the square of the relevant subdivision width in every case.

However, to get that error of the last case down to the stated accuracy, the above mentioned "m + 1" set of approximations is used to obtain a second set of "m" elements representing new approximate values of "T" and having their errors proportional to the 4th power of the relevant subdivision widths. Such correction steps are carried out "m" times ending with a set of a single element representing the best value of the integration and having an error proportional to the "2 (m + 1)"th power of the smallest subdivision "H_m".

The combination of these "m + 1" sets represents a left hand side triangular matrix in which the elements of each column "K" are computed from those of the previous one using the following formula:

$$T(L, K) = T(L, K - 1) + (T(L, K - 1) - T(L - 1, K - 1))/(2^{2K} - 1) \quad ..(2)$$

During execution, the new elements T(L, K) overwrite the old ones T(L, K - 1), and therefore it is necessary to store both the elements T(L) and their differences D(L) in every correction step "K". Moreover, the values of D(L) are rectified to ensure proper operation of the division, and their signs are stored in parallel.

Included is a test driver to demonstrate the calling sequence of the procedure.

Reference:

1. Introduction to Numerical Analysis, 2nd Ed., C.E. Fröberg, Addison-Wesley Publication Company, 1969, p.p. 203.

```
00001  1
00002  1    /*REF. NO. BB28 */
00003  1    /*PROGRAM TITLE RMSTF*/
00004  1
00005  1    20: DECLARE AREA ADDRESS;
0  06  1
00007  1    RMSTF:
00008  1      PROCEDURE (PTR,N) ADDRESS;
00009  2        DECLARE PTR ADDRESS, (F BASED PTR) (129) ADDRESS;
00010  2        DECLARE (TEMP,MAX,C)  ADDRESS;
00011  2        DECLARE POS LITERALLY '0', NEG LITERALLY '0FFH',
00012  2            CNSTNT LITERALLY 'MAX';
00013  2        DECLARE (T,D) (8) ADDRESS;
00014  2        DECLARE S (8) BYTE;
00015  2        DECLARE (N,NL,M,I,L,K,P,H,HIGH) BYTE;
00016  2        DIFFERENCE:
00017  2          PROCEDURE;
00018  3            S(L) = POS;
00019  3            D(L) = T(L) - T(L-1);
00020  3            IF CARRY THEN
00021  3            DO;
00022  3              D(L) = -D(L);
00023  4              S(L) = NEG;
00024  4            END;
00025  3            D(L) = D(L) / (  CNSTNT - 1  );
00026  3           RETURN;
00027  3         END DIFFERENCE;
00028  2
00029  2         H,NL = N;
00030  2         M = 0;
00031  2         DO WHILE NL>1;
0  32  2           M = M + 1;
00033  3           NL = SHR(NL,1);
00034  3         END;
00035  2         MAX = 0;
00036  2         DO I = 0 TO N;
00037  2           IF F(I)>MAX THEN  MAX = F (I);    /* 0<=F(I)<=32787 */
00038  3         END;
00039  2         IF MAX=0 THEN   RETURN MAX;
00040  2         P = P + 1;              /* RESET CARRY */
00041  2         DO WHILE CARRY=0;
00042  2           P = P + 1;
00043  3           MAX = SHL(MAX,1);
00044  3           END;
00045  2                                 /* ALL VALUES OF F(I) WILL BE SHIFTED
00046  2                                    LEFT P TIMES TO INCREASE THE ACCURACY */
00047  2
00048  2           TEMP = F(0) +F(N);
00049  2           IF P>1 THEN  TEMP = SHL(TEMP,P=1);
00050  2           T(0) = TEMP;          /* FIRST APPROX. USING SINGLE TRAPIZOIDE */
00051  2           HIGH = 0;
00052  2           CNSTNT = 4;
00053  2           /* NOW NL = 2 */
00054  2
00055  2           DO L = 1 TO M;        /* FIND BETTER APPROXIMATIONS T(L) */
00056  2
0  7  2              /* SUM UP ALL NEW INTERMEDIATE ORDINATES */
0058  2
00059  2              C = 0;
00060  3              I = SHR(H,1);
```

```
00061  3              DO WHILE   I<N;
00062  3                C = C + SHL(F(I),P);
00063  4                IF CARRY THEN    HIGH = HIGH + 1;
00064  4                I = I + H;
00065  4              END;
00066  3              H = SHR(H,1);
00067  3
00068  3              /* 3-BYTE SUM OF TERMS IN EQUATION (1) */
00069  3              TEMP = TEMP + C;
00070  3              IF CARRY THEN  HIGH = HIGH + 1;
00071  3              /* DIVIDE BY NL USING RIGHT SHIFTS L TIMES */
00072  3              C = TEMP;
00073  3              K = HIGH;
00074  3              DO I = 1 TO L;
00075  3                K = SHR(K,1);
00076  4                C = SCR(C,1);
00077  4              END;
00078  3              T(L) = C;
00079  3
00080  3              CALL DIFFERENCE;
00081  3              NL = NL + NL;     /* USE TWICE AS MANY TRAPIZOIDES FOR
00082  3                                   NEXT CALCULATION */
00083  3              END;
00084  2
00085  2              /* START CORRECTION FOR THE EFFECT OF ALL ERRORS INTRODUCED
00086  2              BY UP TO THE (2M)TH POWER OF THE SMALLEST SUBDIVISION */
00087  2
00088  2              DO K = 1 TO M;
00089  2                CNSTNT = SHL(CNSTNT,2);
00090  3              DO L = K TO M;
00091  3                IF S(L) THEN   T(L) = T(L) - D(L);
00092  4                ELSE   T(L) = T(L) + D(L);
00093  4              IF L>K THEN   CALL DIFFERENCE;
00094  4              END;
00095  3            END;
00096  2         RETURN SHR(T(M),P);
00097  2       END RMSTF;
00098  1                              /* RMSTF INVOCATION  */
00099  1
00100  1       DECLARE Y(17) ADDRESS INITIAL
00101  1               (10000,9412,8889,8421,8000,7619,7273,6956,6667,
00102  1               6400,6154,5926,5714,5517,5333,5161,5000);
00103  1       DECLARE (X1,X2) ADDRESS INITIAL (1,2);  /* X2 > X1 */
00104  1
00105  1       AREA = (X2-X1) * RMSTF(.Y,LAST(Y));
00106  1       EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB29

☐ 4004 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

BINARY TO BCD CONVERSION, LEADING ZERO BLANKING - 32-BIT

Function

THE PROGRAM FIRST LOADS THE BINARY WORD FROM EXTERNAL MEMORY INTO REGISTERS D,E,H,L ACCORDING TO THE ADDRESS IN REG HL. REG A IS LOADED WITH THE MAXIMUM NUMBER OF BCD DIGITS EXPECTED. THIS NUMBER IS USED FOR LEADING ZERO SUPPRESSION - IT INITIALIZES THE RESULT TO ALL BLANKS. THE BINARY WORD IS DIVIDED BY 10 AND THE REMAINDER +30H (ASCII "0") IS TAKEN AS THE NEXT BCD DIGIT IN THE RESULT - LSB FIRST. THE CALCULATION IS TERMINATED WHEN THE BINARY WORD AFTER DIVISION EQUALS ZERO.

Required Hardware

NO HARDWARE REQUIRED

Required Software

16 BIT X 16 BIT DIVIDE ROUTINE - INCLUDED

Input Parameters

HL - ADDRESS OF BINARY WORD
[MSB,...,LSB] , [HL,...,HL-3]
BC - ADDRESS OF BCD (ASCII) RESULT
[MSB,...,LSB] , [BC-9,...,BC]

Output Results

ASCII RESULT IN [BC-9,...,BC] - SEE ABOVE

*

Registers Modified: ALL REGISTERS MODIFIED	**Maximum Subroutine Nesting Level:** 4
RAM Required: UP TO 10 LOCATIONS FOR RESULT 4*N LOCATIONS FOR SOURCE	**Assembler/Compiler Used:** 8080 MACRO ASSEMBLER, VER 2.0
ROM Required: 6DH -109 WORDS	**Programmer:** Dr. Ilan Tal Elscint Ltd.
	Company: Box 5258 Haifa 31-051, ISRAEL

*

```
                        ;REF. NO. BB29
                        ;PROGRAM TITLE 32 BIT BINARY TO BCD ASCII FORMAT
                        ;
                        ;
                        ;
                        ;
                        ;INTERRUPTABLE 32 BIT (OR LESS) BINARY TO BCD ROUTINE
                        ;THESE ARE DESIGNED FOR USE WITH INTERRUPTING
                        ;PERIFERALS WHICH OUTPUT ASCII
                        ;
                        ;
                        ;BCD CONVERSION OF 32 BIT WORD
                        ;REG (HL) CONTAINS ADDRESS OF MSB OF BINARY WORD
                        ;REG (BC) CONTAINS ADDRESS OF RESULT
                        ;REG A SET TO NUMBER BCD DIGITS(10)
                        ;RESULT WILL BE IN [BC-9,BC-8,..BC],MSB FIRST
0010                            ORG     10H
0010 56                 BCD32:  MOV     D,M     ;LOAD WORD
0011 2B                         DCX     H
0012 5E                         MOV     E,M
  13 2B                         DCX     H
0014 7E                         MOV     A,M
0015 2B                         DCX     H
0016 6E                         MOV     L,M
0017 67                         MOV     H,A
0018 3E0A                       MVI     A,10    ;NUM DIGITS
001A C31D00                     JMP     BCD     ;BCD CONVERSION
                        ;BINARY TO BCD (ACSII CODE) CONVERSION ROUTINE
                        ;MAXIMUM 32 BITS BINARY NUMBER
                        ;THE RESULT CAN BE WRITTEN ANYWHERE IN MEMORY
                        ;(BC) CONTAINS ADDRESS OF MEMORY WHERE
                        ;NUMBER OF BCD DIGITS IN RESULT IS WRITTEN
                        ;RESULT WILL BE FOUND IN MEMORY AT ADDRESS
                        ;[(BC+NUM BCD DIGITS-1),..,BC-1,BC], IN FORMAT
                        ;[MSB'S,...,LSB'S]
                        ;REG A CONTAINS NUMBER OF BCD DIGITS IN RESULT
001D C5                 BCD:    PUSH    B       ;STORE RESULT ADDR
001E E5                         PUSH    H       ;STORE LSB
001F 60                         MOV     H,B     ;RESULT ADDR TO HL
0020 69                         MOV     L,C
0021 3620               BCD1:   MVI     M,20H   ;ASCII BLANK
0023 2B                         DCX     H
0024 3D                         DCR     A
0025 C22100                     JNZ     BCD1
0028 210000             BCD2:   LXI     H,0     ;NO REMAINDER
  2B 01F6FF                     LXI     B,NOT 9 ;2'S COMPLEMENT 10
  2E CD4D00                     CALL    DIVIDE  ;MSB/10
0031 E3                         XTHL
0032 EB                         XCHG
```

```
0033 CD4D00            CALL    DIVIDE  ;LSB/10
0036 3E30              MVI     A,30H   ;ASCII "0"
0038 83                ADD     E       ;ADD REMAINDER
0039 47                MOV     B,A     ;TEMP STORE IN B
003A D1                POP     D       ;MSB
003B 7D                MOV     A,L
003C B4                ORA     H
003D B2                ORA     D
003E B3                ORA     E       ;WORD=0?
003F CA4A00            JZ      BCD3
0042 E3                XTHL            ;HL=ADDR OF RESULT
0043 70                MOV     M,B     ;WRITE RESULT
0044 2B                DCX     H       ;DCR ADDR
0045 E3                XTHL
0046 E5                PUSH    H       ;STORE IN STACK ADDR,LSB
0047 C32800            JMP     BCD2
004A E1        BCD3:   POP     H       ;HL=ADDR OF RESULT
004B 70                MOV     M,B     ;WRITE RESULT
004C C9                RET
004D 3E3C      DIVIDE: MVI     A,3CH   ;COUNTER
004F E5                PUSH    H
0050 33        DIV1:   INX     SP      ;ADJUST STACK POINTER FOR
0051 33                INX     SP      ;NO UNDERFLOW
0052 EB        DIV2:   XCHG
0053 1F                RAR             ;SAVE CARRY BIT
0054 29                DAD     H       ;SHIFT DE
0055 17                RAL             ;RESTORE CARRY,SAVE DE OVERFLOW
0056 D25A00            JNC     $+4
0059 23                INX     H       ;WRITE CARRY IN DE
005A C604              ADI     4       ;INCREMENT COUNTER
005C F8                RM
005D EB                XCHG
005E 29                DAD     H
005F 1F                RAR             ;RESTORE DE OVERFLOW
0060 D26400            JNC     $+4
0063 23                INX     H       ;WRITE DE OVERFLOW IN HL
0064 87                ADD     A       ;CORRECT A SHIFT
0065 E5                PUSH    H       ;SAVE HL FOR UNDERFLOW
0066 09                DAD     B       ;2'S COMPLEMENT ADD
0067 DA5000            JC      DIV1    ;NO UNDERFLOW
006A E1                POP     H       ;UNDERFLOW, RESTORE HL
006B C35200            JMP     DIV2
0000                   END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB31

☐ 4004 ☐ 4040 ☐ 8008 XX 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

Fixed point Chebyshev sine and cosine for PL/M users

Function

This package, which enables the PL/M user to compute sines and cosines without the need for floating point arithmetic, consists of four procedures.

MULT performs unsigned multiplication of ADDRESS variables using the shift and add technique. The 16 high order bits of the 32 bit product are returned.

SIN and COS evaluate the sine and cosine, respectively, of angles in the range 0 to 2π. The argument of these routines is a variable of type ADDRESS which contains the angle in radians. To minimize truncation errors, this value is scaled by 2^{13} allowing 3 bits for the representation of the integer part and 13 for the fraction. The value to be returned is obtained by calling the procedure COSINE using the following rules:

Required Hardware

$\cos(x) = \text{COSINE}(x)$	$\sin(x) = \text{COSINE}(\frac{\pi}{2} - x)$	$0 \leq x \leq \frac{\pi}{2}$
$\cos(x) = -\text{COSINE}(\pi - x)$	$\sin(x) = \text{COSINE}(x - \frac{\pi}{2})$	$\frac{\pi}{2} < x \leq \pi$
$\cos(x) = -\text{COSINE}(x - \pi)$	$\sin(x) = -\text{COSINE}(\frac{3\pi}{2} - x)$	$\pi < x \leq \frac{3\pi}{2}$
$\cos(x) = \text{COSINE}(2\pi - x)$	$\sin(x) = -\text{COSINE}(x - \frac{3\pi}{2})$	$\frac{3\pi}{2} < x \leq 2\pi$

Required Software

Input Parameters

COSINE, the heart of the package, finds the cosine of an angle between 0 and $\frac{\pi}{2}$ radians by evaluating the Chebyshev polynomial

$$\cos = 1 + x^2 (a_2 + x^2 (a_4 + x^2 (a_6 + x^2 (a_8 + a_{10} x^2))))$$

$a_2 = -.49999\ 99963$ $\quad a_8 = .00002\ 47609$

$a_4 = .04166\ 66418$ $\quad a_{10} = -.00000\ 02605$

$a_6 = -.00138\ 88397$

Output Results

The ADDRESS input parameter in this case is scaled by 2^{15} with the output a signed ADDRESS value scaled by 2^{14}. Users not requiring this much accuracy may use the standard economization techniques to reduce the degree of the polynomial.

The main program should cause the following values to be computed:

TSTCOS (0) = 2D 43	TSTSIN (0) = 2D 42
TSTCOS (1) = D2 BE	TSTSIN (1) = 2D 42
TSTCOS (2) = D2 BE	TSTSIN (2) = D2 BE
TSTCOS (3) = 20 02	TSTSIN (3) = C8 93

Although not exhaustively tested, it appears the average error in COSINE is less than .0001 when the truncation of the input angle is taken into account.

Registers Modified: ALL	**Assembler/Compiler Used:** PL/M
RAM Required: 42 words	**Programmer:** Bradley G. Stewart
ROM Required: 711 words	**Company:** Aeronutronic Ford Corp.
Maximum Subroutine Nesting Level: 3	**Address:** 3939 Fabian Way Palo Alto, CA 94303

```
00001  1
00002  1  /*REF. NO. BB31 */
00003  1  /*PROGRAM TITLE FIXED POINT CHEBYSHEV SINE AND COSINE FOR PL/M USERS */
00004  1
00005  1
00006  1  DECLARE (TSTCOS,TSTSIN) (4) ADDRESS;
00007  1  DECLARE VALUES(4) ADDRESS INITIAL (6433,19301,32169,42893);
00008  1  DECLARE I BYTE;
00009  1
00010  1    /*    THE MULT ROUTINE MULTIPLIES TWO BIT NUMBERS
00011  1          AND RETURNS THE HIGH ORDER 16 BITS OF THE RESULT */
00012  1
00013  1  MULT:     PROCEDURE (MULX,MULY) ADDRESS;
00014  2            DECLARE (MULX,MULY,MULHO,MULLO) ADDRESS;
00015  2            DECLARE MULI BYTE;
00016  2            MULHO = MULX;   MULLO = 0;
00017  2            DO MULI = 1 TO 16;
00018  2               MULLO = SHL(MULLO,1);
00019  3               MULHO = SCL(MULHO,1);
00020  3               IF CARRY THEN DO;
00021  3                  MULLO = MULLO + MULY;
00022  4                  MULHO = MULHO PLUS 0;
00023  4                END;
00024  3             END;
00025  2             RETURN MULHO;
00026  2             END MULT;
00027  1
00028  1    /*       THIS ROUTINE FINDS THE COSINE OF AN ANGLE BETWEEN
00029  1             0 AND PI/2  THE ARGUMENT IS SCALED BY 2**15    */
00030  1
00031  1  COSINE:   PROCEDURE (RADS) ADDRESS;
00032  2            DECLARE (RADS,RADT) ADDRESS;
00033  2            RADS = MULT (RADS,RADS);
00034  2            RADT = 10923 - MULT(RADS,1456 - MULT(RADS,
00035  2            104 - MULT(5,RADS)));
00036  2            RETURN  16384 - MULT(RADS,32768 - MULT(RADS,RADT));
00037  2            END COSINE;
00038  1
00039  1    /*       THE NEXT TWO FUNCTIONS FIND THE SINE AND COSINE OF
00040  1             ANGLES BETWEEN 0 AND 2*PI  THE ARGUMENTS ARE IN
00041  1             RADIANS AND ARE SCALED BY 2**13  */
00042  1
00043  1  SIN:        PROCEDURE (RADS) ADDRESS;
00044  2             DECLARE RADS ADDRESS;
00045  2             IF RADS>51471 THEN RADS = RADS - 51471;
00046  2             IF RADS>38603 THEN
00047  2                RETURN -COSINE(SHL(RADS - 38603,2));
00048  2             IF RADS>25735 THEN
00049  2                RETURN - COSINE(SHL(38603 - RADS,2));
00050  2             IF RADS>12867 THEN
00051  2                RETURN COSINE(SHL(RADS - 12867,2));
00052  2             RETURN COSINE(SHL(12867 - RADS,2));
00053  2             END SIN;
00054  1
00055  1  COS:        PROCEDURE (RADS) ADDRESS;
00056  2               DECLARE RADS ADDRESS;
00057  2               IF RADS>51471 THEN RADS = RADS - 51471;
00058  2               IF RADS>38603 THEN
00059  2                  RETURN COSINE(SHL(51471 - RADS,2));
00060  2               IF RADS>25735 THEN
```

```
00061  2                    RETURN -COSINE(SHL(RADS - 25735,2));
00062  2                 IF RADS>12867 THEN
00063  2                    RETURN -COSINE(SHL(25735 - RADS,2));
00064  2                 RETURN COSINE(SHL(RADS,2));
00065  2                    END COS;
00066  1
00067  1     /*            THE MAIN PROGRAM CALCULATES THE SINE AND COSINE OF
00068  1                   45, 135, 225, AND 300 DEGREES  */
00069  1
00070  1     DO I = 0 TO 3;
00071  1        TSTSIN(I) = SIN(VALUES(I));
00072  2        TSTCOS(I) = COS(VALUES(I));
00073  2     END;
00074  1   EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB32

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

32 BIT DIVIDE SUBROUTINE

Function

This is a non reentrant divide routine. The dividend is 32 bits; the divisor and quotient are 16 bits in length. All are signed numbers. Carry is set if overflow occurred, clear otherwise.

Required Hardware

Minimum: SDK-80

Required Software

None

Input Parameters

Routine is entered with return address at top of stack, HL pointing to the following parameter block:

```
((H) (L)) + 0 MS BYTE DIVIDEND      ((H) (L)) + 4 MS BYTE DIVISOR
            1          :                        5 LS BYTE DIVISOR
            2          :
            3 LS BYTE DIVIDEND
```

Output Results

```
((H)(L)) -6 MS BYTE DIVIDEND        ((H)(L)) - 2 MS BYTE DIVISOR
         -5                                  - 1 LS BYTE DIVISOR
         -4                                  + 0 MS BYTE QUOTIENT
         -3 LS BYTE DIVIDEND                 + 1 LS BYTE QUOTIENT
```

Carry clear for no overflow, carry set for overflow. If overflow had occurred, quotient value is undefined.

Registers Modified: A,B,C,D,E,H,L, F/F's	**Assembler/Compiler Used:** MAC80 on TYMSHARE
RAM Required: 202 $(CA)_{16}$+8+6 Routine & Temp Storage & Stack	**Programmer:** Albert J. Gibbons
ROM Required: None	**Company:** Westinghouse Industry Systems Div.
Maximum Subroutine Nesting Level: Non reentrant	**Address:** 200 Beta Drive Pittsburgh, Pa. 15238

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB33

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

8 BIT MULT AND DIV.

Function

MULT : TO MULT TWO 8 BIT OPERANDS TO FORM A 16 BIT RESULT

DIV : TO DIV 15 BITS BY 8

Required Hardware

8080 PROCESSOR SYSTEM

Required Software

NONE

Input Parameters

SEE NOTES ON ATTACHED DOCUMENTATION

Output Results

Registers Modified: A,D,H, AND L	**Assembler/Compiler Used:** 8080 ASSEMBLER MODIFIED
RAM Required: 30 BYTES	**Programmer:** RONALD D. WILFONG
ROM Required: 0	**Company:** DIABLO SYSTEMS
Maximum Subroutine Nesting Level: 0	**Address:** 1270 E. ARQUES, SUNNYVALE, CALIF.

```
                ;REF. NO. BB33
                ;PROGRAM TITLE 8 BIT MULT & DIVIDE
                ;
                ;
                ;
                ;    MULTIPLY TWO 8 BIT OPERANDS TO FROM A 16 BIT RESULT
                ;    OPERANDS AND RESULT ARE UNSIGNED NUMBER.
                ;    SAVES B, C, AND E REGS
                ;    CALLING SEQ:    MVI    H, OPERAND1
                ;                    MVI    E, OPERAND2
                ;                    CALL   MULT
                ;                    RESULT IS IN HL REG

0000 3E08       MULT:   MVI     A, 8        ;LOOP COUNTER
0002 2E00               MVI     L, 0
0004 55                 MOV     D, L
0005 29         MULT1:  DAD     H
0006 D20A00             JNC     MULT2
0009 19                 DAD     D
000A 3D         MULT2:  DCR     A
000B C20500             JNZ     MULT1
000E C9                 RET
                ;    DIVIDE SUB. 15 BITS/8 = 8
                ;    L REG = HL/E REG
                ;    REMAINDER IN H REG
                ;    HL IS UNSIGNED 15 BITIDEND
                ;    E IS UNSIGNED 8 BIT DIVISOR
                ;    QUOTIENT MUST BE AN UNSIGNED 8 BIT QUANTITY
                ;    H MUST BE LESS THAN E REG

000F 1608       DIV:    MVI     D, 8        ;LOOP COUNTER
0011 29         D16L:   DAD     H
0012 7C                 MOV     A, H
0013 93                 SUB     E
0014 DA1900             JC      D16N
0017 67                 MOV     H, A
0018 2C                 INR     L
0019 15         D16N:   DCR     D
001A C21100             JNZ     D16L
001D C9                 RET
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB34

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Double-Precision Integer Arithmetic Package

Function

- Compute sine and cosine of angle defined by an orthagonal 16-bit coordinate pair.
- Normalize 16-bit integer.
- Multiply two 16-bit numbers for a 32-bit result.
- Divide a 32-bit number by a 16-bit divisor to yield a 16-bit quotient.
- Take the square root of a 30-bit number.

Required Hardware

Intel 8080 Processor

Required Software

None

Input Parameters

See next page

Output Results

See next page

Registers Modified: All	**Assembler/Compiler Used:** ISIS 8080 Macro Assembler V1.0
RAM Required: approximately 30 bytes	**Programmer:** N. Anderson/G. Woodley
ROM Required: a total of 581 bytes	**Company:** Woodley Associates
Maximum Subroutine Nesting Level:	**Address:** 604 Indian Home Road Danville, CA 94526

ADDENDUM TO:

Microcomputer User's Library

Submittal Form

Double-Precision Arithmetic Package

Definition of Input Parameters and Output Results

Sine and Cosine

The COSIN entry is used if the coordinate pair is pointed to by HL.

The COSIN1 entry is used if the X coordinate is in BC, and the Y coordinate is in DE.

In either case, the coordinate pairs are stored in four-byte groups, from lowest addressed byte to highest addressed byte in the following order: X-coordinate, Low-order byte, High order byte, Y-coordinate, Low-order byte, High-order byte.

Exit with cosine and sine of the angle defined by X and Y coordinates on the stack, in that order.

Normalize

This subroutine is used primarily internally by the cosine subroutine. Its function is to increase the significance of the computed values.

Enter NORM with a comparison value in HL and item to be normalized in BCDE. The comparison value places an upper limit on the normalization and is usually equal to $7FFF_{16}$.

Exit with the scale factor in A, BCDE normalized, and HL undisturbed.

Multiply

Enter MUL with BC and DE containing 16-bit 2's complement multiplicand and multiplier. On exit, a 32-bit 2's complement result will be in BCDE.

Divide

Enter DIV with BCDE containing a 32-bit 2's complement dividend, and HL containing a 16-bit 2's complement divisor. Exit with carry equal to Ø and 16-bit 2's complement result in HL if division is successful. If the division is not successful (overflow or division by Ø) exit with carry equals 1 and result equals Ø.

Square Root

Enter SQRT with a 30-bit positive operand in BCDE. On exit, the 15-bit result is in HL. Maximum result is $7FFE_{16}$.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BC14

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

FLOATING POINT EXTENDED MATH PACKAGE FOR 8080 - XMATH

Function

FLOATING POINT ROUTINES
INTEGER/FRACTIONAL PART
SQUARE ROOT
LOG BASE E
EXPONENTIAL, E↑X
LOG BASE 10
10↑X
REAL BASE TO REAL EXPONENT A↑X
TRIG SIN, COS, AND TAN
ARCSIN, ARCCOS, AND ARCTAN
POLYNOMIAL EXPANDER
DEGREES <- -> RADIANS CONVERSIONS

Required Hardware

8080

Required Software

8080 LIBRARY BASIC FLOATING POINT MATH PACKAGE BC1
OR EXACT FUNCTIONAL EQUIVALENT

Input Parameters

SEE ATTABHED DOCUMENTATION

Output Results

SEE ATTACHED DOCUMENTATION

Program offered on diskette only.

Registers Modified: SEE DOCUMENTATION	**Assembler/Compiler Used:** RESIDENT MACRO-ASSEMBLER
RAM Required: 256 BYTES	**Programmer:** RICHARD C. ALLEN
ROM Required: 1700 BYTES	**Company:** TEXAS MICROSYSTEMS, INC.
Maximum Subroutine Nesting Level:	**Address:** 6610 HARWIN, SUITE 125 HOUSTON, TEXAS 77036

SOFTWARE USER GUIDE AND DESCRIPTION, SYSTEM - XMATH

NAME: XMATH

FUNCTION: 8080 EXTENDED FUNCTION MATH PACKAGE

AUTHOR: RICHARD C. ALLEN, TEXAS MICROSYSTEMS, INC.

DATE: 20SEP75

MACHINE: 8080

LANGUAGE: ASSEMBLY

EXTERNALS: LIBRARY MATH PACKAGE, INTEL BC1

ROUTINES:
FPOLY - POLYNOMIAL EXPANDER
FINT - FRACTIONAL/INTEGER PART
FSQR - SQUARE ROOT
FLN - LOG BASE E
FEXP - EXPONENTIAL, E↑X
FLOG - LOG BASE 10
FALOG - 10↑X
FAX - REAL BASE TO REAL EXPONENT, A↑X
FSIN - SINE
FCOS - COSINE
FTAN - TANGENT
FASIN - INVERSE SINE (ARCSIN)
FACOS - INVERSE COSINE (ARCCOS)
FATAN - INVERSE TANGENT (ARCTAN)
FCDR - DEGREES TO RADIANS CONVERSION
FCRD - RADIANS TO DEGREES CONVERSION

DESCRIPTION: SEE ATTACHED DESCRIPTION FOR GENERAL NOTES

XMATH - 8080 FLOATING POINT EXTENDED FUNCTION MATH PACKAGE

XMATH, IN CONJUNCTION WITH THE INTEL LIBRARY FLOATING POINT MATH PACKAGE BC1 OR EQUIVALENT, PROVIDES THE 8080 USER WITH ALL OF THE COMMONLY USED EXTENDED MATH FUNCTIONS.

IN GENERAL, THE ROUTINES RETURN STATUS AND REGISTER INFORMATION IN THE SAME MANNER AS THE LIBRARY BASIC MATH PACK. INPUT ARGUMENTS ARE IN EITHER THE FLOATING POINT ACCUMULATOR, REFERRED TO AS FAC, AND/OR THE HL REGISTER.

UNLESS SPECIFIED BY THE CALLING SEQUENCE, THE GENERAL MACHINE REGISTERS ARE INSIGNIFICANT ON INPUT AND RESULTS ON EXIT FROM A ROUTINE WILL BE THE SAME AS FOR THE BASIC MATH PACK.

*** FLOATING POINT NUMBERS MAY NOT CROSS PAGE BOUNDRIES! ***

ERRORS SUCH AS ILLEGAL INPUT ARGUMENTS OR NUMERICAL OVERFLOW WILL CALL AN INTERNAL ERROR HANDLING ROUTINE 'ERROR' THIS ROUTINE WILL STORE THE ADDRESS OF THE 'CALL ERROR' IN A RAM LOCATION LABELED 'XMERR' (TWO BYTES) AND RETURN TO THE CALLING SUBROUTINE. ILLEGAL NEGATIVE INPUTS TO ROUTINES LIKE SQUARE ROOT, WILL CAUSE THE ERROR FLAG TO BE SET AND THE ABSOLUTE VALUE OF THE INPUT WILL BE USED. WHEN THE INPUT CAUSES OR WILL CAUSE NUMERICAL OVERFLOW OR UNDERFLOW THE FAC WILL BE SET TO +/- FULL SCALE (APPROXIMATELY +/- 1E38) OR ZERO.

SOME OF THE ROUTINES CALL OTHER ROUTINES IN THE XMATH SYSTEM, SO THE USER SHOULD BE SURE TO INCLUDE THOSE ROUTINES WHEN ATTEMPTING TO STRIP OUT AN INDIVIDUAL FUNCTION.

THE MEMORY ALLOCATION VARIABLES, ROM, ROMK, AND RAM, ARE USED IN THE FOLLOWING MANNER. SINCE THE BASIC MATH PACK WAS WRITTEN FOR THE 8008, ALL FLOATING POINT NUMBERS USED BY BOTH SYSTEMS MAY NOT CROSS PAGE BOUNDRIES. BECAUSE OF THIS LIMITATION, THE READ-ONLY SECTION OF XMATH IS DIVIDED INTO TWO SECTIONS, ONE FOR PROGRAM, AND ONE FOR CONSTANTS. THE MEMORY ALLOCATION VARIABLES ROM AND ROMK CONTROL THESE TWO AREAS WITH THE VARIABLE RAM CONTROLLING THE READ/WRITE AREA. TO IMPROVE READABILITY AND ALSO TO ALLOW EASY REMOVAL OF SINGLE ROUTINES, THE 'ROMK' AND 'RAM' AREAS FOR EACH ROUTINE FOLLOW THE ROUTINE IN THE LISTING EVEN THOUGH THEY ARE IN DIFFERENT MEMORY AREAS.

THE TOTAL PACKAGE REQUIRES SLIGHTLY LESS THAN 2K BYTES OF ROM AND SLIGHTLY LESS THAN 256 BYTES OF RAM. THE RAM AREA COULD BE REDUCED SLIGHTLY BE COMBINING TEMPORARY STORAGE LOCATIONS BETWEEN FUNCTIONS.

WITH THE EXCEPTION OF TRIGONOMETRIC FUNCTIONS INVOLVING ANGLES CLOSE TO ZERO OR EVEN MULTIPLES OF PI, THE ROUTINES ARE ACCURATE TO BETTER THAN SIX DECIMAL PLACES. THIS ACCURACY HAS NOT BEEN FULLY TESTED HOWEVER. THE ROUTINES HAVE BEEN TESTED WITH RANDOM INPUTS COMPARED TO RESULTS FROM A HEWLETT-PACKARD HP-55 AND SPECIFIC TESTING HAS BEEN PERFORMED AROUND THE ALGORITHM SWITCHING POINTS. NO WARRANTY IS EXPRESSED OR IMPLIED ON THE PART OF THE AUTHOR OR TEXAS MICROSYSTEMS FOR THE ACCURACY OF THE SYSTEM.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. BC15

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

Floating Point Package for Intel 8008 and 8080 Microprocessors

Function

Add, subtract, multiply, divide, square root

Required Hardware

8008/8080 with 1.5K of memory (RAM/ROM)

Required Software

Octal Debugging Routine,
Ref. No. F-8, Page 9-9

Listing and tape available from Intel Corporation. Complete report available for $4 from:
National Tech. Info. Service
US Dept. of Commerce
5285 Port Royal Road
Springfield, Va. 22151
Order No. UCRL-51940

Input Parameters

Address of both operands
Address of result

Output Results

Output to TTY routines included
Input from TTY routines included

Registers Modified: All	**Assembler/Compiler Used:** Intel assembler
RAM Required: 4 words per operator 4 words per result +17 word scratch	**Programmer:** M.D. Maples/H. Brand
ROM Required:	**Company:** Lawrence Livermore Lab
Maximum Subroutine Nesting Level:	**Address:** Box 808, M/S L403 Livermore, Ca. 94566

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB36

☐ 4004 ☐ 8008 ☒ 8080 ☐ 4040 (use additional sheets if necessary)

Program Title

RNGEN RANDOM NUMBER GENERATOR

Function

TO GENERATE UNIFORM RANDOM NUMBERS BETWEEN 0 AND AN INTEGER LIMIT SPECIFIED BY THE USER. A MULTIPLICATIVE CONGRUENTIAL METHOD IS USED. THIS METHOD IS BASED ON OVERFLOW.

Required Hardware

8080 MICROPROCESSOR

Required Software

PL/M COMPILER TO GENERATE CODING.
A COMPLETE EXAMPLE OF USE AND CODING GENERATED BY PL/M IS ATTACHED.

Input Parameters

ONE "INTERNAL" PARAMETER MUST BE PASSED TO THE PROCEDURE. THIS IS THE UPPER LIMIT OF THE DESIRED DISTRIBUTION.

Output Results

FOR EVERY CALL, THE PROCEDURE RNGEN WILL RETURN ONE RANDOM NUMBER BETWEEN 0 AND THE UPPER LIMIT.

Registers Modified:	**Maximum Subroutine Nesting Level:** 1
RAM Required: SEE ATTACHED SHEET	**Assembler/Compiler Used:** PL/M 80
ROM Required:	**Programmer:** K. CHRISTIAN KNUDSEN
	Company: DATA INDUSTRI A/S PILESTREDET 75 c; OSLO 3, NORWAY

NOTES REGARDING:

RNGEN

RNGEN is written in PLM80 for the Intel 8080 microprocessor. The program will calculate uniform random numbers in the range 0 - (MAX -1), i.e. integers from zero to one less than the specified MAX. MAX is assumed to be ADDRESS.

The user must declare a global variable named SEED and initialize this to any odd positive number. This initialization is only needed once. If the random numbers should be in the 0 - 1.0 range, a floating point conversion routine must be used.

The complete program, including the main program, uses 251 bytes.

```
00001  1   /*REF. NO. BB36 */
00002  1   /*PROGRAM TITLE RNGEN (RANDOM GENERATOR NUMBER */
00003  1         /*  RANDOM NUMBER GENERATOR  */
00004  1         /*  DEFINE GLOBAL VARIABLES  */
00005  1             DECLARE (SEED,MAX,NUM) ADDRESS;
00006  1         /*  DEFINE THE RNGEN FUNCTION */
00007  1         /*  AND LOCAL VARIABLES      */
00008  1   RNGEN:    PROCEDURE (ARG) ADDRESS;
00009  2             DECLARE (ARG,LOC1,LOC2) ADDRESS;
00010  2             LOC1=SEED*899;
00011  2         /*  SET THE NEW SEED  */
00012  2             SEED=LOC1;
00013  2             LOC2=LOC1/ARG;
00014  2             LOC2=LOC1-LOC2*ARG;
00015  2             RETURN LOC2)
00016  2             END RNGEN;
00017  1         /*        EXAMPLE OF THE USE OF RNGEN  */
00018  1         /*        SET SEED = ODD NUMBER .  */
00019  1         /*        THIS SHOULD BE DONE ONCE ONLY */
00020  1                   SEED=11365;
00021  1         /*        SET MAX. ALL RANDOM NUMBER WILL BE */
00022  1         /*        IN THE RANGE 0 - (MAX-1)     */
00023  1                   MAX=100;
00024  1         LOOP:     NUM=RNGEN(MAX);
00025  1                   OUTPUT(0)=NUM;
00026  1                   GO TO LOOP;  /* ENDLESS LOOP */
00027  1       EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB37

☐ 4004 ☐ 4040 ☒ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

RANDOM NUMBER GENERATOR - 8-BIT

Function

The program reads data from page Ø, address FF and generates a random number. The new number is writen back in the same location. All numbers except zero are generated Zero is a disallowed state and is corrected in the program.

Required Hardware

TTY

Required Software

TTY Routine

Input Parameters

The program may be called with any number in memory page Ø, address FF

Output Results

Program exits with the pseudo random number in register A and memory, page Ø, address FF

Registers Modified: A, B, C, H, L	**Assembler/Compiler Used:** MAC80, VER 2.0
RAM Required: 2 Bytes	**Programmer:** Joe J. Gentle
ROM Required: 26 Bytes	**Company:** Collins Radio Company
Maximum Subroutine Nesting Level: 0	**Address:** Mail Station 106-171 Cedar Rapids, Iowa 52402

```
                ;REF. NO. BB37
                ;PROGRAM TITLE 8 BIT PSEUDO RANDOM NUMBER GENERATOR
                ;
                ;
                ;
                ;
                                                ;8 BIT PSEUDO RANDOM NUMBER GENE
                ;
                                        ;THE PROGRAM READS DATA FROM ADDRESS FF
                                ;GENERATES A RANDOM NUMBER WHICH WHICH IS WRITTE
                                ;IN THE SAME LOCATION. ALL NUMBERS EXCEPT ZERO
                                ;ARE GENENERATED. ZERO IS A DISALLOWED STATE AND
                                ;IS CORRECTED IN THE PROGRAM
                ;
0000 2600       RANDOM: MVI   H,00H             ;SET MEMORY POINTERS
0002 2EFF               MVI   L,0FFH            ;
0004 4C                 MOV   C,H               ;C=0
0005 7E                 MOV   A,M               ;GET NUMBER
0006 BC                 CMP   H                 ;IS IT ZERO
0007 C20B00             JNZ   SKIP              ;SKIP IF NOT
000A 7D                 MOV   A,L               ;SET TO ALL ONES IF SO
000B 47         SKIP:   MOV   B,A               ;PIT IT IN B
000C E61D               ANI   1DH               ;SAVE BITS 0,2,3,AND 4
000E EA1300             JPE   PAR               ;JUMP IF PARITY EVEN
0011 0E80               MVI   C,80H             ;C=80H IF PARITY ODD
0013 78         PAR:    MOV   A,B               ;GET NUMBER AGAIN
0014 0F                 RRC                     ;ROTATE IT RIGHT
0015 E67F               ANI   7FH               ;SET MSB TO ZERO
0017 81                 ADD   C                 ;ADD C TO IT
0018 77                 MOV   M,A               ;STORE NEW NUMBER
0019 C9                 RET                     ;RETURN
                ;
                ;
                                                ;JOE J GENTLE
                                                ;COLLINS RADIO CO
                                                ;JUNE 17, 1975
                ;
                ;
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. Nc BB38

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

16-Bit Random Number Generator

Function

*The subroutine implements a linear congruential sequence which generates 16-bit random numbers. The random numbers produced range from 0000 to FFFF with a period less than or equivalent to 2 ** 16. An 8-bit random number is available as the upper byte of the 16-bit random number.*

*X(N+1) = (2053*X(N) +13849)mod 2**16*

Required Hardware

An INTELLEC 8/MOD 80 with an ASR 33 TTY connected on PORT 0 and 1.

Required Software

The INTELLEC 8/MOD 80 System Monitor can be used for testing the subroutine.

Input Parameters

The Stack Pointer must be initialized to some area in RAM memory. The H and L registers must be preset to a location containing the latest random number, X(N).

Output Results

The new random number, X(N+1), is returned to the location specified by the H and L registers.

Registers Modified: *None*	**Assembler/Compiler Used:** *INTELLEC 8 MACRO ASSEMBLER*
RAM Required: *42H for the subroutine; 0AH for the stack; 2 for random no.*	**Programmer:** *Vito A. Trujillo*
ROM Required: *None*	**Company:** *Zot Manufacturing Co.*
Maximum Subroutine Nesting Level: *Stack size greater than 0AH*	**Address:** *1619 Reed St. Lakewood, Colorado 80215*

```
                        ;REF. NO. BB38
                        ;PROGRAM TITLE 16 BIT RANDOM NUMBER GENERATOR
                        ;
                        ;
                        ;
                        ; 16 BIT RANDOM NUMBER GENERATOR

                        ;UTILIZES A LINEAR CONGRUENTIAL SEQUENCE
                        ;OF THE FORM X(N+1) = (A*X(N) +C) MOD M, N>-0
                        ;WITH A=2053 (10) AND C=13849 (10)
                        ;PERIOD <= 2**16

                        ;THE STACK POINTER MUST BE INITIALIZED
                        ;H AND L MUST POINT TO X(N)

                        ;THE NEW VALUE X(N+1) IS RETURNED TO LOCATION
                        ;SPECIFIED BY H AND L

                        ;NONE OF THE REGISTER CONTENTS ARE DESTROYED


1000                    ORG     1000H

1000 F5         RAND:   PUSH    PSW             ;SAVE PSW
1001 D5                 PUSH    D               ;SAVE D AND E
1002 C5                 PUSH    B               ;SAVE B AND C
1003 4E                 MOV     C,M             ;LOAD B AND C
1004 23                 INX     H               ;WITH X(N)
1005 46                 MOV     B,M
1006 2B                 DCX     H
1007 CD1C10             CALL    AROUT           ;CALCULATE X(N)*2053D
100A E5                 PUSH    H               ;ADD X(N) TO THE RESULT
100B 211A10             LXI     H,CNST
100E CD3910             CALL    SBR2
1011 E1                 POP     H
1012 71                 MOV     M,C             ;SAVE X(N+1) AT PWTR
1013 23                 INX     H
1014 70                 MOV     M,B
1015 2B                 DCX     H
1016 C1                 POP     B               ;RESTORE B AND C
1017 D1                 POP     D               ;RESTORE D AND E
1018 F1                 POP     PSW             ;RESTORE PSW
1019 C9                 RET                     ;RETURN TO MAIN PROGRAM
101A 1936       CNST:   DW      13849
                        ;SUBROUTINE TO CALCULATE X(N)*2053D
                        ;ASSUMES X(N) IS CONTAINED IN B AND C
                        ;RETURNS X(N)*2053D TO B AND C

101C 1609       AROUT:  MVI     D,9             ;X(N)*2^9
```

```
101E CD2D10             CALL    SBR1
1021 CD3910             CALL    SBR2            ;X(N)+X(N)*2^9
1024 1602               MVI     D,2             ;2^2*(X(N)+X(N)*2^9
1026 CD2D10             CALL    SBR1
1029 CD3910             CALL    SBR2            ;ADD TO X(N)
102C C9                 RET                     ;RETURN


                        ;FORMS  (B AND C)*2^D

102D 97         SBR1:   SUB     A               ;CLEAR A AND CARY
102E 79                 MOV     A,C             ;SHIFT C LEFT
102F 17                 RAL
1030 4F                 MOV     C,A
1031 78                 MOV     A,B             ;SHIFT B LEFT
1032 17                 RAL
1033 47                 MOV     B,A
1034 15                 DCR     D               ;TEST D=0
1035 C8                 RZ                      ;YES, RETURN
 036 C32D10             JMP     SBR1            ;NO, SHIFT AGAIN

                        ;ADDS A 16-BIT NUMBER POINTED TO BY H AND L
                        ;TO THE CONTENT OF B AND C, AND PLACES THE
                        ;  SUM IN B AND C

1039 97         SBR2:   SUB     A               ;CLEARS A AND CARRY

103A 7E                 MOV     A,M             ;ADD LOW BYTE
103B 81                 ADD     C
103C 4F                 MOV     C,A
103D 23                 INX     H               ;ADD HIGH BYTE
103E 7E                 MOV     A,M
103F 88                 ADC     B
1040 47                 MOV     B,A
1041 2B                 DCX     H
1042 C9                 RET                     ;RETURN

0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB39

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

PL/M HISTOGRAM PROCEDURE AND RANDOM NUMBER GENERATOR

Function

main program generates an 8 bit shift register sequence
by xoring the first and last bits and shifting the result
into the next random numbers bottom bit.
1000 numbers are generated and then histogrammed.
histogram procedure sets up an output histogram array
and then prints the histogram on the tty when commanded
after printing , the array is not zeroed so that intermediate
results may be displayed without effect on the final histogram.

Required Hardware

TTY on ports 0 and 1

Required Software

INTELLEC MONITOR VER 3.0 (tty output routine only)
PL/M compiler on a host machine

Input Parameters

main program needs a starting value for the random number
(set to 1 in listing XX=1;)

histogram procedure needs
1) min , max value to be histogrammed
2) number of histogram bins
3) data to be histogrammed
4) control variable (see comments in listing)

Output Results

main program prints random numbers as they are generated
(this can be defeated by pulling the two call print
cards in the main program)
histogram procedure generates a histogram scaled to fit on
the TTY

Registers Modified: PL/M program	**Assembler/Compiler Used:** PL/M VER 2.0
RAM Required: 800H locations	**Programmer:** Rex Tracy
ROM Required: 0	**Company:** Colorado State University
Maximum Subroutine Nesting Level: stack size = 8 bytes	**Address:** Electrical Engineering Dept. Ft. Collins , Colo. 80523

```
00001  1
00002  1   /*REF. NO. 8839 */
00003  1   /*PROGRAM TITLE PL/M HISTOGRAM PROCEDURE */
00004  1
00005  1
00006  1
00007  1   DECLARE CR LITERALLY '0DH', LF LITERALLY '0AH', TRUE LITERALLY '1',
00008  1           FALSE LITERALLY '0';
00009  1
00010  1   /* PRINT VIA MONITOR */
00011  1   PRINTSCHAR: PROCEDURE (CHAR);
00012  2       DECLARE CHAR BYTE;
00013  2            DECLARE IOCO LITERALLY '3809H';
00014  2                 GOTO IOCO;
00015  2       END PRINTSCHAR;
00016  1
00017  1   PRINTSSTRING: PROCEDURE(NAME,LENGTH);
00018  2       DECLARE NAME ADDRESS,
00019  2           (LENGTH,I,CHAR BASED NAME) BYTE;
00020  2           DO I = 0 TO LENGTH-1;
00021  2           CALL PRINTSCHAR(CHAR(I));
00022  3           END;
00023  2       END PRINTSSTRING;
00024  1
00025  1   PRINTSNUMBER: PROCEDURE(NUMBER,BASE,CHARS,ZEROSSUPPRESS);
00026  2       DECLARE NUMBER ADDRESS, (BASE,CHARS,ZEROSSUPPRESS,I,J) BYTE;
00027  2       DECLARE TEMP (16) BYTE;
00028  2       IF CHARS > LAST(TEMP) THEN CHARS = LAST(TEMP);
00029  2           DO I = 1 TO CHARS;
00030  2           J=NUMBER MOD BASE + '0';
00031  3           IF J > '9' THEN J = J + 7;
00032  3           IF ZEROSSUPPRESS AND I <> 1 AND NUMBER = 0 THEN
00033  3               J = ' ';
00034  3           TEMP(LENGTH(TEMP)-I) = J;
00035  3           NUMBER = NUMBER / BASE;
00036  3           END;
00037  2       CALL PRINTSSTRING(.TEMP + LENGTH(TEMP) - CHARS,CHARS);
00038  2       END PRINTSNUMBER;
00039  1   /*
00040  1   /* PL/M PROCEDURE TO PRODUCE A HISTOGRAM */
00041  1   /*
00042  1                       REX TRACY - 07/29/75
00043  1                       DEPARTMENT OF ELECTRICAL ENGINEERING
00044  1                       COLORADO STATE UNIVERSITY
00045  1                       FT. COLLINS , COLORADO        80523
00046  1                       303-491-5691
00047  1   */
00048  1   /*
00049  1   */
00050  1   HISTOGRAM:    PROCEDURE (MIN,MAX,DTAA,NBINS,CONTRL) ;
00051  2   /*  MIN = MINIMUM VALUE TO BE HISTOGRAMMED
00052  2       MAX = MAXIMUM VALUE TO BE HISTOGRAMMED
00053  2       DTAA = DATA TO BE PLACED INTO HISTOGRAM
00054  2       NBINS = NUMBER OF BINS FOR HISTOGRAM
00055  2       CONTRL = 0 THEN ZERO HISTOGRAM OUTPUT ARRAY
00056  2                1 THEN PRINT CURRENT HISTOGRAM ARRAY
00057  2                2 THEN INSERT DTAA INTO HISTOGRAM ARRAY
00058  2   */
00059  2                DECLARE (MIN,MAX,DTAA,NBINS,CONTRL) BYTE ;
00060  2                DECLARE ARRAY (255) ADDRESS ;
```

```
00061  2                 DECLARE (I,J,K) BYTE ;
00062  2                 DECLARE (RANGE,R,PERBIN) ADDRESS;
00063  2                 DECLARE (V,V1) ADDRESS;
00064  2                 DECLARE MXN ADDRESS INITIAL (0);
0  65  2                 DECLARE HDR1 DATA ('MIN=');
00066  2                 DECLARE HDR2 DATA (', MAX=');
00067  2                 DECLARE HDR3 DATA (', NBINS=');
00068  2                 DECLARE HDR4 DATA (', EACH * =');
00069  2                 DECLARE HDR5 DATA (' UNITS');
00070  2                 DECLARE BDR1 DATA ('        .');
00071  2                 DECLARE BDR2 DATA (' .');
00072  2                 DECLARE STAR DATA ('*');
00073  2   /*
00074  2   */
00075  2       IF        CONTRL=0        THEN
00076  2                 /* ZERO OUTPUT ARRAY */
00077  2       DO;        DO I=0 TO LAST(ARRAY);
00078  3                 ARRAY(I)=0;
00079  4                 MXN=0;
00080  4                 END;
00081  3       END;
00082  2       ELSE DO;
00083  2                 IF                CONTRL=1          THEN
00084  3                 /* PRINT THE HISTOGRAM */
00085  3                 DO;
00086  3   /*  SCALE OUTPUT TO 72 COLUMNS              (72-8=64)
00087  3                 V1=SHR(MXN,6)+1;
00088  4                 CALL PRINT$STRING(.HDR1,LENGTH(HDR1));
00089  4                 CALL PRINT$NUMBER (MIN,10,6,1);
00090  4                 CALL PRINT$STRING(.HDR2,LENGTH(HDR2));
0  91  4                 CALL PRINT$NUMBER (MAX,10,6,1) ;
00092  4                 CALL PRINT$STRING (.HDR3,LENGTH(HDR3)) ;
00093  4                 CALL PRINT$NUMBER (NBINS,10,6,1);
00094  4                 CALL PRINT$STRING (.HDR4,LENGTH(HDR4)) ;
00095  4                 CALL PRINT$NUMBER (V1,10,6,1);
00096  4                 CALL PRINT$STRING (.HDR5,LENGTH(HDR5));
00097  4                 CALL PRINT$STRING (.(CR,LF,LF),3);
00098  4                 DO I=1 TO 36;
00099  4                      CALL PRINT$STRING(.BDR2,LENGTH(BDR2));
00100  5                 END;
00101  4                 CALL PRINT$STRING (.(CR,LF),2);
00102  4                 RANGE = MAX-MIN;
00103  4                 PERBIN=RANGE/NBINS;
00104  4                 DO J=0 TO NBINS;
00105  4                      IF I MOD 5 = 0 THEN
00106  5                      DO:
00107  5                           J=MIN+I*PERBIN;
00108  6                           CALL PRINT$NUMBER (J,10,6,1);
00109  6                           CALL PRINT$STRING (.BDR2,LENGTH(BDR2));
00110  6                      END;
00111  5                      ELSE CALL PRINT$STRING(.BDR1,LENGTH(BDR1));
00112  5                 V=ARRAY(I);
00113  5                 DO WHILE V>=V1 ;
00114  5                      CALL PRINT$STRING (.STAR,1);
00115  6                      V=V-V1;
0  16  6                      END;
00117  5                 IF V<>0 THEN CALL PRINT$NUMBER (V,16,1,1);
00118  5                 CALL PRINT$STRING (.(CR,LF),2);
00119  5                 END;
00120  4                 END;
```

```
00121  3                ELSE
00122  3                IF   CONTRL=2  THEN
00123  3      /*       INSERT DTAA INTO CURRENT ARRAY */
00124  3                DO;
0  25  3                IF DTAA<=MAX AND DTAA>=MIN THEN
00126  4                DO;
00127  4                RANGE=MAX-MIN;
00128  5                R=DTAA+MIN;
00129  5                PERBIN=RANGE/NBINS;
00130  5                J=R/PERBIN;
00131  5                ARRAY (J)=ARRAY(J)+1;
00132  5                IF ARRAY(J)>MXN THEN MXN=ARRAY(J);
00133  5                END;
00134  4                END;
00135  3            END;
00136  2         RETURN;
00137  2         END HISTOGRAM;
00138  1    /* MAIN PROGRAM -- PRODUCES RANDOM NUMBERS AND HISTOGRAMS THEM */
00139  1    /*    RANDOM NUMBERS PRODUCES BY AN 8 BIT SHIFT REGISTER
00140  1          SEQUENCE
00141  1          THIS SEQUENCE IS GENERATED BY XOR THE TOP AND BOTTOM BITS AND
00142  1          SHIFTING THE RESULT INTO THE BOTTOM BIT
00143  1    */
00144  1               DECLARE I ADDRESS;
00145  1               DECLARE (Z,Y,XX) BYTE;
00146  1               DECLARE W BYTE;
00147  1               DECLARE MONIT LITERALLY '3800H';
00148  1    STRT:      CALL HISTOGRAM (0,0,0,0,0);
00149  1               XX=1;
00150  1               DO I= 1 TO 1000;
0  51  1               Y=XX;
00152  2               Z=XX AND 1B;
00153  2               W=(Z+ROL(XX,1)) AND 1B;
00154  2               XX=SHL(XX,1) OR W;
00155  2               CALL PRINT$NUMBER(XX,10,6,1);
00156  2               CALL PRINT$STRING (.(CR,LF),2);
00157  2               CALL HISTOGRAM (0,0FFH,XX,50,2);
00158  2               END;
00159  1               CALL HISTOGRAM (0,0FFH,0,50,1);
00160  1    /*  RETURN TO THE MONITOR */
00161  1              GOTO MONIT;
00162  1    EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. BB40

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

RANDOM NUMBER GENERATOR - RANDOM$BITS

Function

A non-multiplicative pseudo-random number generator.

Required Hardware

No special hardware is required, merely an 8080 processor and memory.

Required Software

PL/M compiler

Input Parameters

The calling sequence is CALL RANDM$BITS(POINTER,COUNT);
POINTER is an address variable containing the address of the area in which the pseudo-random values are to be placed.
COUNT is a byte variable containing the number of 16bit pseudo-random values which are to be placed contiguously in the area indicated by POINTER.

Output Results

Registers Modified: none	**Assembler/Compiler Used:** PL/M VER 3.0
RAM Required: about 300 bytes, but may be decreased.	**Programmer:** KARL AUERBACH
ROM Required: about 300 bytes, for program storage	**Company:** SYSTEM DEVELOPMENT CORPORATION
Maximum Subroutine Nesting Level:	**Address:** 2500 Colorado Avenue Santa Monica, CA, 90406

```
00001  1   /*REF. NO. BB40 */
00002  1   /*PROGRAM TITLE RANDOMSBITS */
00003  1
00004  1
00005  1   DECLARE LXTAB LITERALLY '8';
00006  1   DECLARE XTAB(LXTAB) ADDRESS INITIAL(
00007  1   10480,57948,31935,20673,16110,13245,56625,22878);
00008  1   DECLARE XINDEX BYTE INITIAL(0);
00009  1   DECLARE LYTAB LITERALLY '8';
00010  1   DECLARE YTAB(LYTAB) BYTE
00011  1   INITIAL(156,165,133,071,236,189,069,059);
00012  1   DECLARE YINDEX BYTE INITIAL(0);
00013  1   DECLARE LVTAB LITERALLY '128';
00014  1   DECLARE VTAB(LVTAB) ADDRESS INITIAL(
00015  1   20969,39615,58629,16379,54613,42880,12952,32307,
00016  1   56941,64952,01547,33703,30613,29975,28551,40719,
00017  1   55157,64951,35749,58104,32812,44592,22851,18510,
00018  1   50720,13300,25280,64580,34963,07844,62028,36086,
00019  1   48501,03574,38917,09250,42971,20562,20486,18062,
00020  1   45709,32427,00102,06541,59583,41546,51900,45578,
00021  1   62730,32261,02338,04822,43040,12515,25499,44437,
00022  1   19746,59846,60332,08930,46920,51805,16296,16834,
00023  1   34191,06004,21597,05269,12682,50501,14951,34405,
00024  1   07896,34925,48280,59894,52924,49106,46942,54238,
00025  1   51275,28225,23541,19585,50136,56613,50585,55230,
00026  1   47908,60859,05250,57031,35503,40129,19233,64239,
00027  1   47625,42579,28672,56047,45960,24120,35848,11059,
00028  1   44488,62077,50813,52639,37005,29812,18327,59685,
00029  1   00265,44247,32286,57676,21059,58581,34994,34698,
00030  1   30671,57352,37554,53464,55580,07672,19308,53214);
00031  1   RANDMSBITS: PROCEDURE(POINTER,COUNT);
00032  2   /*
00033  2       PROCEDURE RANDMSBITS
00034  2            THIS PROCEDURE PROVIDES THE CALLER WITH SOME NUMBER OF
00035  2            16 BIT RANDOM NUMBERS.
00036  2
00037  2       CALLING SEQUENCE
00038  2            CALL RANDMSBITS(POINTER,COUNT);
00039  2
00040  2            POINTER IS THE ADDRESS OF THE AREA WHERE THE
00041  2            RANDOM NUMBERS ARE TO BE PLACED.
00042  2            COUNT IS A BYTE VALUE CONTAING THE NUMBER OF
00043  2            16 BIT RANDOM NUMBERS TO BE PLACED, ONE AFTER
00044  2            ANOTHER, IN THE AREA DESIGNATED BY POINTER.
00045  2
00046  2       VALUES RETURNED
00047  2            A RANDOM BIT STRING.
00048  2
00049  2       SUBPROCEDURES CALLED
00050  2            NONE
00051  2
00052  2       GLOBAL VARIABLES REFERENCED
00053  2            XTAB
00054  2            YTAB
00055  2            XINDEX
00056  2            YINDEX
00057  2            VTAB
00058  2
00059  2       REMARKS
00060  2            THE SEQUENCE OF RANDOM NUMBERS GENERATED BY THIS
```

```
00061  2            PROCEDURE IS REPEATIBLE. E.G. GIVEN THE SAME
00062  2            VALUES IN THE GLOBAL VARIABLES XTAB, YTAB, INDEX.
00063  2            YINDEX, AND VTAB, THE SAME RANDOM NUMBER WILL RESULT.
00064  2
   65  2            IT IS EXPECTED THAT THESE GLOBAL VARIABLES WILL BE
00066  2            INITIALIZED BY SOME OTHER PROCEDURE.  THE CONTENTS OF
00067  2            THE GLOBAL VARIABLES MAY BE ALTERED EXTERNALLY AT ANY TIME
00068  2            EXCEPT WHEN THIS PROCEDURE IS IN CONTROL.
00069  2
00070  2            THE GENERAL TECHNIQUE USED IS TAKEN FROM
00071  2            KNUTH, THE ART OF COMPUTER PROGRAMMING VOL. 2
00072  2            'SEMINUMERICAL ALGORITHMS'.
00073  2            ALGORITHM M PAGE 30.
00074  2            THE METHOD USES TWO SUB-RANDOM NUMBER GENERATORS, X AND Y.
00075  2            NEITHER X NOR Y NEED BE VERY GOOD.
00076  2            A TABLE V IS INDEXED BY Y AND THE VALUE THEREIN IS OUTPUT.
00077  2            X IS USED TO REPLACE THE USED CONTENTS OF V.
00078  2
00079  2            HERE WE USE AN ADDITIVE RANDOM GENERATOR FOR BOTH
00080  2            X AND Y. THE METHOD IS DESCRIBED IN 'A BETTER
00081  2            ADDITIVE CONGFUENTIAL RANDOM NUMBER GENERATOR'
00082  2            BY ROGER BURFORD AS PUBLISHED IN DECISION SCIENCE
00083  2            VOLUME 4 NUMBER 2 APRIL '73.
00084  2            THIS METHOD USES A FIFO LIST.
00085  2            A RANDOM NUMBER IS GENERATED BY SUMMING THE CONTENTS
00086  2            OF THE ELEMENTS OF A FIXED-LENGTH FIFO LIST.
00087  2            THE LIST IS THEN UPDATE BY REMOVING THE OLDEST
00088  2            ELEMENT FROM THE HEAD OF THE LIST AND ADDING THE
00089  2            NEWLY GENERATED RANDOM NUMBER TO THE TAIL OF THE
00090  2            LIST.
00091  2            THE FIFO LIST IS IMPLEMENTED USING A TABLE (XTAB
00092  2            AND YTAB) AND AN INDEX POINTER (XINDEX AND YINDEX).
00093  2            THE INDEX SHOWS THE CURRENT HEAD AND NEXT AVAILABLE
00094  2            TAIL. THE INDEX IS ADVANCED AS EACH RANDOM NUMBER
00095  2            IS GENERATED AND IS WRAPPED AROUND WHENEVER
00096  2            NECESSARY.
00097  2            IT IS IMPORTANT THAT THESE TABLES BE INITIALIZED SO THAT
00098  2            THERE IS AT LEAST ONE ODD VALUE IN THEM.
00099  2            OTHERWISE IT IS POSSIBLE FOR THE TABLES TO REACH A
00100  2            DEGENERATE STATE WHERE ZERO IS ALWAYS GENERATED.
00101  2
00102  2   */
00103  2   DECLARE POINTER ADDRESS;
00104  2   DECLARE RANDNO BASED POINTER (0) ADDRESS;
00105  2   DECLARE TADDR ADDRESS;
00106  2   DECLARE COUNT BYTE;
00107  2   DECLARE (I,J,TBYTE) BYTE;
00108  2   DO J = 0 TO COUNT-1;
00109  2       TADDR,TBYTE = 0;
00110  3       DO I = 0 TO LAST(YTAB); /*SUM THE Y FIFO LIST */
00111  3            TBYTE = TBYTE + YTAB(I);
00112  4            END;
00113  3       YTAB(YINDEX) = TBYTE; /*DELETE OLD HEAD AND ADD NEW TAIL*/
00114  3       YINDEX = (YINDEX + 1) AND 07H;      /*INCREMENT INDEX MODULO 8*/
00115  3       DO I = 0 TO LAST(XTAB);
   16  3            TADDR = TADDR + XTAB(I);
00117  4             END;
00118  3       XTAB(XINDEX) = TADDR;
00119  3       XINDEX = (XINDEX + 1) AND 07H;
00120  3       TBYTE = TBYTE AND 07FH;             /*TBYTE MODULO 128*/
```

```
00121  3        RANDNO(J) = VTAB(TBYTE);/*GET NEXT 16 BITS OF OUTPUT*/
00122  3        VTAB(TBYTE) = TADDR;/*REPLACE THE USED VALUE*/
00123  3        END;
00124  2    END RANDM$BITS;
   25  1    DECLARE T1 BYTE;
00126  1    DECLARE TVAL(256) ADDRESS;
00127  1    DO T1 = 0 TO 255;
00128  1         TVAL(T1) = 0;
00129  2         END;
00130  1    DO T1 = 0 TO 127;
00131  1         CALL RANDM$BITS(.TVAL(T1),1);
00132  2         END;
00133  1    DO T1 = 128 TO 200 BY 2;
00134  1         CALL RANDM$BITS(.TVAL(T1),2);
00135  2         END;
00136  1    CALL RANDM$BITS(.TVAL(202),54);
00137  1    EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Dec. 1976

Ref. BA11

☐ 4004 ☐ 4040 ☐ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

Factorial of a Decimal Number

Function

A decimal number in the range 1 to 99 is accepted via register D. Then, through the use of add and subtract subroutines (which adjust the numbers in decimal), the factorial of that number is calculated.

Required Hardware

Required Software

Input Parameters

A decimal number (1 - 99) in register D.
A value 0 will give 99.

Output Results

The value is expected from the address contained in register pair H. The result is also kept in "SUM" and "SUM + 1" where "SUM" is the least significant number (in decimal).

Registers Modified: A, C, D, E, H, L, and Carry	**Assembler/Compiler Used:** 8080 MDS Macro Assembler 1.0
RAM Required: 64 bytes	**Programmer:** J.L. Marcel Lalonde
ROM Required: Ø	**Company:** Agriculture Canada
Maximum Subroutine Nesting Level: 2	**Address:** Engineering Research Service, Ottawa, Ontario, Canada, K1A 0C6

98-034B

```
                ;REF. NO. BA11
                ;PROGRAM TITLE FACTORIAL OF A DECIMAL NUMBER
                ;
                ;
                ;
                ;
                ;TO CALCULATE FACTORIAL OF A DECIMAL NUMBER
                ;IN THE RANGE FROM 1 TO 99.

                FACTR:
0000 7A                 MOV     A,D         ;STORE DECIMAL #
0001 323D00             STA     NUMB
0004 97                 SUB     A
0005 114100             LXI     D,SUM       ;CLEAR THE SUM
0008 12                 STAX    D
0009 13                 INX     D
000A 12                 STAX    D
000B CD2900     AGAIN:  CALL    DADD        ;FIND ! OF #
000E CD1800             CALL    DSUB
0011 C20B00             JNZ     AGAIN       ;FINISHED WHEN A=00
0014 214100             LXI     H,SUM       ;REG PAIR H= ADDR OF RESULT
0017 C9                 RET

                DSUB:                       ;TO SUBTRACT 1 FROM A NUMBER
0018 113D00             LXI     D,NUMB      ;GET NUMBER
001B 213F00             LXI     H,SBTRA     ;AND 1
001E 37                 STC
001F 3E99               MVI     A,99H       ;SET 100'S COMPLEMENT
0021 CE00               ACI     0H
0023 96                 SUB     M           ;OF NUMBER
0024 EB                 XCHG                ;&SUBTRACT 1 FROM IT
0025 86                 ADD     M
0026 27                 DAA                 ;CONVERT TO DECIMAL
0027 77                 MOV     M,A         ;STORE RESULT
0028 C9                 RET

                DADD:                       ;TO ADD A NUMBER TO A SUM
0029 114100             LXI     D,SUM       ;GET SUM
002C 213D00             LXI     H,NUMB      ;& NUMBER
002F 0E02               MVI     C,2         ;MAKE IT DOUBLE PRECISION
0031 AF                 XRA     A           ;CLEAR CARRY
0032 1A         LOOP:   LDAX    D
0033 8E                 ADC     M
0034 27                 DAA
0035 12                 STAX    D           ;STORE SUM
0036 0D                 DCR     C           ;DONE WHEN C REG = 0
0037 C8                 RZ
0038 13                 INX     D           ;LOAD NEXT BYTE
0039 23                 INX     H
```

```
003A C33200             JMP     LOOP    ;ADD NEXT 2 BYTES

003D 00         NUMB:   DB      00
003E 00                 DB      00
003F 01         SBTRA:  DB      01
0040 00                 DB      00
0041 00         SUM:    DB      00
0042 00                 DB      00
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Dec. 1976
Ref. BA12

☐ 4004 ☐ 8008 ☒ 8080 ☐ 4040 (use additional sheets if necessary)

Program Title

BCD UP/DOWN COUNTER

Function

Implementation of a 2-digit BCD up/down counter.
See description on source tape.

Required Hardware

1 output port and 1 input pprt.

Required Software

None

Input Parameters

Bits 1 and 0 of input port for an up or down command, respectively.

Output Results

2-digit BCD number to output port and stored in COUNT.

Registers Modified: A,B,C,D	**Maximum Subroutine Nesting Level:** 1
RAM Required: 2 (for COUNT and SLOCT)	**Assembler/Compiler Used:** Assembler Ver. 4.1
ROM Required: 197	**Programmer:** Richard Young
	Company: Naval Air Rework Facility Code 323 Alameda, Calif 94501 (415) 869-4472

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Dec. 1976

Ref. BB42

☐ 4004 ☐ 4040 ☐ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

FRACTIONAL MULTIPLY - POSITIVE - 3-BYTE

Function

Multiplies two signed fractional; 24-bit numbers producing a 24-bit fractional answer.

Required Hardware

24-bit number = S $2^{-1}_2 2^{-2}_2 3^{-3}_2 4^{-4} \ldots 2^{-23}$; S = sign
i.e., +1/2 = 01000000 00000000 00000000
= 400000H

Required Software

Input Parameters

MARRY: DW Mult 1; multiplier
DW Mult 2; multipicand
DW Mult 3; results
(see listing for further description)

Output Results

Registers Modified: All	**Assembler/Compiler Used:** 370/Fortran IV
RAM Required: 10 bytes	**Programmer:** Wm. James - Dept. 75
ROM Required: 333 bytes	**Company:** Airesearch Mfg. Co.
Maximum Subroutine Nesting Level:	**Address:** Torrence, CA. 90509

98-0348

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Dec. 1976
Ref. BB43

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

ALGEBRAIC CØMPARE SUBRØUTINE

Function

ALGEBRAIC CØMPARE BETWEEN TWØ 16 BIT NUMBERS (A & B) WHICH ARE REPRESENTED IN SIGNED 2'S CØMPLEMENT NØTATIØN.

Required Hardware

NØNE

Required Software

NØNE

Input Parameters

A-VALUE IN D & E REGS WITH MSB IN D-REG.
B-VALUE IN H & L REGS WITH MSB IN H-REG.

Output Results

CØNDITIØN FLAGS AS FØLLØWS:

ZERØ SET, CARRY RESET, MINUS RESET, IF A = B.

ZERØ RESET, CARRY SET, MINUS RESET, IF A > B.

ZERØ RESET, CARRY RESET, MINUS SET, IF A < B.

Registers Modified: PSW (A-REG & FLAGS)	Assembler/Compiler Used: INTELLEC 8/MØD 80 MACRØ ASSMB. VERSIØN 1. 0
RAM Required: NØNE	Programmer: C. Messerle
ROM Required: 31 BYTES	Company: INTECO
Maximum Subroutine Nesting Level: NØNE WITHIN CPR	Address: HARRISØN, ØHIØ

98-034C

```
                        ;REF. NO. BB43
                        ;PROGRAM TITLE ALGEBRAIC COMPARE SUBROUTINE
                        ;
                        ;
                        ;
                        ;
                        ;
                        ; SUBROUTINE CPR --- 08/04/76
                        ;
                        ; ALGRBRAIC COMPARE BETWEEN TWO 16 BIT NUMBERS
                        ; ( A & B ) WHICH ARE REPRESENTED IN SIGNED
                        ; 2'S COMPLEMENT NOTATION.
                        ;
                        ; ENTER WITH A-VALUE IN D&E REGS - MSB IN D-REG.
                        ;   "     "   B-VALUE IN H&L REGS - MSB IN H-REG.
                        ;
                        ; RETURNS CONDITION FLAGS AS FOLLOWS:
                        ;
                        ; ZERO SET - CARRY RESET - MINUS RESET, IF A = B.
                        ; ZERO RESET - CARRY SET - MINUS RESET, IF A > B.
                        ; ZERO RESET - CARRY RESET - MINUS SET, IF A < B.
                        ;
                        ; AFFECTS PSW ( A-REG & FLAGS ).
                        ;
                        ;
0300                            ORG     0300H
                        ;
                        ;
0300 7A         CPR:    MOV     A,D     ;LIKE SIGNS?
0301 AC                 XRA     H
0302 F20B03             JP      LIKE
0305 7C                 MOV     A,H     ;NO.  AFFECT CARRY AND
0306 C680               ADI     80H     ;MINUS FLAGS WITH B-VALUE.
0308 C31A03             JMP     BELOW
030B 7B         LIKE:   MOV     A,E     ;PERFORM A-B.
030C 95                 SUB     L
030D C21603             JNZ     NZERO
0310 7A                 MOV     A,D
0311 9C                 SBB     H
0312 C8                 RZ              ;A-B = 0?
0313 17                 RAL             ;NO.  AFFECT CARRY WITH COMP
0314 3F                 CMC             ;OF SIGN OF DIFFERENCE.
0315 C9                 RET
0316 7A         NZERO:  MOV     A,D     ;FINISH A-B.
0317 9C                 SBB     H
0318 17                 RAL             ;AFFECT CARRY WITH COMP
0319 3F                 CMC             ;OF SIGN OF DIFFERENCE.
031A C0         BELOW:  RNZ
031B 1F                 RAR             ;RESET ZERO FLAG.
```

```
031C C681               ADI     81H
031E C9                 RET
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. # BB44

intel

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

ARC TANGENT (DOUBLE PRECISION) FOR 8080 - DPATN

Function

Performs double precision calculation of the arc tangent for an unsigned sixteen bit number less than unity. Also included with this package are routines for double precision addition, multiplication and division.

Required Hardware

Any 8080 based microcomputer system with 233 bytes free in PROM and 9 bytes free in RAM.

Required Software

No other software required except the calling program. Note that the calling program has access also to the double precision arithmetic routines DPADD, DPMPY, and DPDIV.

Input Parameters

For DPATN, the sixteen bit augment is assumed to be stored in register pair B-C with the MSB at bit 7 of register B and the LSB at bit position 0 of register C. The sixteen bit number is required to be a positive binary fraction with the binary point at the extreme left. For DPADD, DPMPY and DPDIV, see details at beginning of each one of these subroutines.

Output Results

Results from DPATN, DPADD, DPMPY and DPDIV are all returned in register pairs B-C.

Registers Modified: All Registers	Assembler/Compiler Used: VØRTEX MCS 8080 (Varian 73)
RAM Required: 11 Locations (9 variables, 2 for Stack)	Programmer: Dr. Barry B. Woo
ROM Required: 233 bytes	Company: Boeing Aerospace Company
Maximum Subroutine Nesting Level: One (1)	Address: P. O. Box 3999 Seattle, Wash. 98124

METHOD: The rational approximation[1] below is used:

$$\tan^{-1}x = \frac{a_0 + a_1x^2 + a_2x^4}{b_0 + b_1x^2 + b_2x^4}, \quad 0 < X < 1$$

Computation is shortened by one multiplication step if nesting is done as follows:

$$\tan^{-1}x = \frac{a_0 + X^2(a_1 + a_2x^2)}{b_0 + X^2(b_1 + b_2x^2)}$$

Here, six multiplications and one divide is required. Assuming a 2 MHz 8080 clock, the time for a multiply is 1.3 MS and a divide is 2.1 MS, thus giving a total of approximately 9.9 MS for the 16 bit arc tangent calculation.

In hexadecimal, the constants for the above approximation are given by

$a_0 = b_0 = b_1 =$ 75 A4H

$a_1 =$ 4EGEH

$a_2 =$ O4FEH

$b_2 =$ 14B8H

1. The author is indebted to Paul S. Smith of BAC for this rational function approximation which has a precision exceeding 2 ppm over the full range of X (0,1). The coefficients in decimal are $a_0 = b_0 = b_1 = 0.459534$, $a_1 = 0.306366$, $a_2 = 0.019501$, $b_2 = 0.080933$.

Test Program: The following test program was ran to validate the contributed program DPATN. Note that the argument X input is loaded directly in the BC register pair, while the output (THETA) is stored in memory locations 6113, 6114 as well as in register pair BC.

```
0000    31FF00    LXI SP, OFFH      ;Initialize Stack
0003    010080    LXI D, 8000H      ;Load X
0006    CD0041    Call DPATN        ;Call tan -1X
0009    C30900    S: JMP S          ;Stop
```

The various test cases are tabulated below:

		1	2	3	4	
Input	0004	00	8F	66	FF	X_L
	0005	80	02	E6	FF	X_h
Output	6113	BD	8E	99	OF	Θ_L
	6114	76	02	BB	C9	Θ_h

Ref. # BA13

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☒8008 ☐8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

ARC.TAN 2 Subroutine

Function

The function is to generate the angle θ = arctan y/x in any of the four quadrants.

Required Hardware

Intel 8080 and 400 bytes of memory chips

Required Software

none

Input Parameters

x, a fractional number, in DE
y, a fractional number, in HL

Output Results

θ, a fractional number scaled by 180° (π), will be in HL.

1. If x = -1. = 8000H, $\theta = \frac{180^\circ}{180^\circ}$.

2. If y = -1. = 8000H, $\theta = \frac{270^\circ}{180^\circ}$.

3. Ohterwise, $\theta = \frac{\tan \frac{-1y}{x}}{180^\circ}$

Registers Modified: All	Programmer: Linn Zien
RAM Required: 8 bytes	Company: Naval Weapons Center
ROM Required: 371 bytes	Address:
Maximum Subroutine Nesting Level: 4	City: China Lake
Assembler/Compiler Used: 16K ISIS 8080 Assembler, V 1.0	State: California 93555

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. # BC16

intel

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

Floating Point Interpreter (FPIP)

Function

Interprets floating point psuedo-op codes and arguments. This results in a significant reduction in memory space required for floating point code and also simplifies the coding of floating point operations. Additional useful floating point operations and conversions are provided including tangent, degree/radian conversions, log 10, exp 10, y to the x power, inverse subtraction, negate the accumulator, square, and simple I/O handlers for BCD I/O floating point routines.

Required Hardware

No special hardware is required.
Provisions are made for a TTY.

Required Software

Floating Point Package (Refs. BC1, BC2, BC4)
TTY routines if using TTY.

Input Parameters

No parameters are required in the registers. See description for discussion of floating point pseudo-operation codes.

Output Results

Upon exit from the interpreter, registers A-D contain the last floating point accumulator (FAC), and the condition bits reflect the status of the FAC and the overflow flag.

Registers Modified: All	**Assembler/Compiler Used:** 8080 Macro Assembler
RAM Required: 0 to 4 additional over floating point package	**Programmer:** D. W. Lovse
ROM Required: 77H minimum, 149H for complete package	**Company:** Univ. of Illinois
Maximum Subroutine Nesting Level: 2 levels plus floating point package	**Address:** 76 Roger Adams Lab Urbana, IL 61801

98-034C

Floating Point Interpreter.

This floating point interpreter was designed for the Intel User's Library Floating Point Package (Ref. BC1, BC2, BC4). Its purpose is to reduce the program space required for floating point operations and also simplify code writing itself. The amount of memory space saved is at least 1/2 to 2/3 compared to the direct single-call method.

The interpreter, when implemented, is an integral part of the floating point package. It is called as a subroutine. Code following the call is interpreted as floating point pseudo-instructions which are decoded by the interpreter which generates the necessary floating point package subroutine calls. The address of arguments requested by instructions such as floating add, subtract, divide, etc., is stored as an address constant word directly following the pseudo instruction. Thus these are effectively three-byte pseudo instructions.

Floating point operations such as CHS, ZRO, SIN, etc., operate directly on the floating accumulator (FAC) and thus need be only single-byte pseudo-instructions. An "exit" pseudo-instruction is provided to terminate interpreting. The interpreter returns program control to the location directly following the exit pseudo-instruction. After an exit, all condition bits reflect the status of the FAC and overflow flag, and the FAC is in registers A, B, C, D.

Codes can be easily added-to or deleted-from the interpreter. The main key to this is the floating point instruction entry point table (see listing). The floating point interpreter entry point table contains the entry of the floating point operations. Entries are indexed by this pseudo-op code value (0 to 7 FH). The length of the table is contained in the dataword "INSL". Pseudo-codes equal-to or greater-than INSL are treated as NOPs by the interpreter. Entries which require arguments or multiple subroutine calls are indicated by entry point names containing @. Entries without @ signs are invariably single-byte pseudo-instructions and a subroutine call is directly made to the corresponding floating point routine. For operations requiring arguments such as the floating add, the interpreter fetches the argument address and puts it in the H,L registers before calling the requested floating point routine. In all cases, the floating point routine returns to the interpreter, except for the exit routine.

A special description is required for the floating point package two-float/fix conversions, and BCD I/O operations. The address argument for FLOAT points to memory space where five bytes of data are stored. The first byte is the scale factor. The next four bytes contain the 32-bit fixed word. For the FIX operation the argument also points to a five-byte memory. The first byte is the scaling factor desired. The next four bytes are where the "fixed" 32-bit word is deposited. The INPUT and OUTPUT routines are single-byte pseudo-instructions with internal handlers written into this version for console terminal I/O. The buffer used is in the same

floating point scratch RAM used by the elementary functions. This INPUT also provides for a rubout code which restarts the input operation after echoing a backslash.

Extended functions beyond the basic elementary function floating point package are included in this interpreter. Others could easily be added to reduce code space requirements. Floating point subroutines can be written in the interpretive mode since the interpreter can be recursively called. Examples included are the TANGENT and SQUARE functions. Extended functions provided in this package include Y to the X power, where Y is the FAC and X is an argument. Also included are LOG base 10 and EXP base 10 which operate directly on the FAC. RAD and DEG are conversion routines to change the FAC from degrees to radians and back respectively. Another routine, ISUB, provides inverted subtraction where the FAC is subtracted from the argument. FNEG negates the sign of the FAC. SQU and TAN functions are performed respectively on the FAC, and are examples themselves of interpretive code. Macros are used to generate these pseudo-instructions, or, the pseudo-instructions could be made permanent op-codes in the user's assembler or cross assembler. A suggested naming scheme follows:

Floating Point Pseudo-Instructions

FEXIT	FINIT	FADD arg
FSUB arg	FGET arg	FMPY arg
FDIV arg	FCHS	FABS
FSTR arg	FZRO	FIN
FOUT	FFIX arg	FLOAT
FIDIV	FSQRT	FSIN
FCOS	FATAN	FSINH
FCOSH	FEXP	FLOG

Extended Set:

FYX arg	FEX10	FLG10
FDEG	FRAD	FSQU
FISUB arg	FNEG	FTAN
FNOP		

Ref. # BB47

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☒ 8048 ☐ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

8048 - DIV -- DIVISION ROUTINE

Function

R34 = R23/A = remainder

Required Hardware

8748 or 8035

Required Software

None

Input Parameters

R23 = 16 bit dividend
A = 8 bit divisor

Output Results

R34 = 16 bit result
R2 = 8 bit remainder

Registers Modified: A, R0, 2, 3, 4, 5	Programmer: H. Serindat
RAM Required: None	Company: Societe ECA
ROM Required: 47 Bytes	Address: Z.I. Toulon-EST
Maximum Subroutine Nesting Level: 1	City: 83087 Toulon-Cedex
Assembler/Compiler Used:	State: France

```
                ;REF.  NO.  BB45
                ;PROGRAM TITLE MLT12(12X12 MULTIPLY)
                ;
                ;
                ;
                ;MLT12
                ;12X12 MULTIPLY.
                ;MULTIPLICAND IN D,E.  MOST SIGNIFICANT 4 BITS
                ;OF MULTIPLIER IN A.  LEAST 8 BITS IN C.
                ;ANSWER IN A,H,L.

0000 87         MLT12:ADD A         ;GET HI 4 BITS OF MULTIPLIER READY.
0001 87               ADD A
0002 87               ADD A
0003 87               ADD A
0004 210000           LXI H,0       ;SET UP SUBROUTINE.
0007 0604             MVI B,4
0009 CD1300           CALL MULT
000C 79               MOV A,C
000D 0608             MVI B,8
 )0F CD1300           CALL MULT
0012 C9               RET

0013 29         MULT  :DAD H        ;SHIFT PARTIAL RESULT LEFT ONE PLACE.
0014 17               RAL           ;ROTATE MULTTIPLIER LEFT TO CARRY.
0015 D21B00           JNC DEC       ;TEST MULTIPLIER AT CARRY.
0018 19               DAD D         ;ADD MULTIPLICAND TO RESULT.
0019 CE00             ACI 0         ;ACCUMULATE HI ORDER RESULT.
001B 05         DEC   :DCR B        ;DECREMENT BIT COUNTER.
001C C21300           JNZ MULT
001F C9               RET
0000                  END MLT12
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

DECIMAL MULTIPLY SUBROUTINE (N1 x N2 BYTES) - MBCD

Function

2N1 digits by 2N2 digits decimal multiply for unsigned integers giving a 2(N1 + N2) digits results.

Required Hardware

None

Required Software

None

Input Parameters

- Memory MTR (N1 bytes) contains multiplier (least significant byte in lower adress)
- Memory MTD (N2 + 1 bytes) contains multiplicand (least significant byte in lower adress, higher adress = 0)
- Memory area: see fig. 1

Output Results

- Memory RES (N1 + N1 bytes) contains product

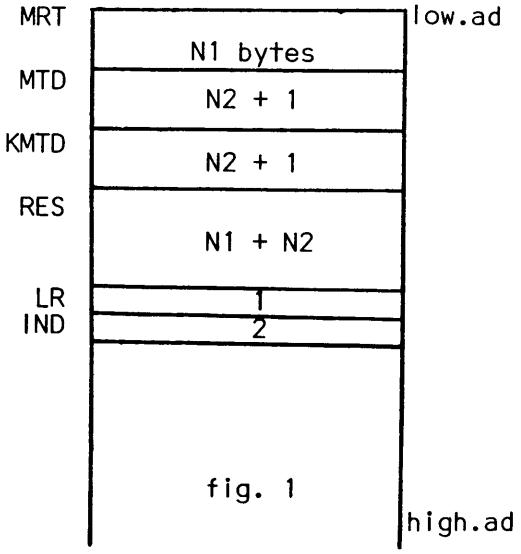


fig. 1

Registers Modified: A,B,C,D,E,H,L	Programmer: Renault C.
RAM Required: (2N1 + 3N1 + 5) bytes	Company: Latecoere
ROM Required: 157 bytes	Address: 135 rue de Periole
Maximum Subroutine Nesting Level: 2	City: Toulouse
Assembler/Compiler Used: MDS Assembler vers. 1-0	State: France

```
                        ;REF.  NO.  BA14
                        ;PROGRAM TITLE MBCD
                        ;
                        ;
                        ;
0020                            ORG     20H
                        ;MBCD:  N1 BYTES * N2 BYTES DECIMAL MULTIPLY SUBROUTINE
                        ;
0020 110702             MBCD:   LXI     D,MTD   ;MOVE MTD INTO KMTD
0023 210A02                     LXI     H,KMTD
0026 0602                       MVI     B,N2
0028 CDB400                     CALL    TBDH
002B AF                         XRA     A
002C 77                         MOV     M,A
002D 12                         STAX    D       ;CLEAR MSB IN MTD AND KMTD
002E 210D02                     LXI     H,RES   ;CLEAR RES AND LR
0031 0606                       MVI     B,(N1+N2)+1
0033 77                 RAZ:    MOV     M,A
0034 23                         INX     H
0035 05                         DCR     B
0036 C23300                     JNZ     RAZ
 ?39 0E01               UN:     MVI     C,1
003B 210000             DEUX:   LXI     H,0     ;CLEAR INDEX
003E 221302                     SHLD    IND
0041 2A1302             TROIS:  LHLD    IND
0044 110402                     LXI     D,MTR
0047 19                         DAD     D
0048 3A1202                     LDA     LR
004B B7                         ORA     A       ;FLAG Z AFFECTED
004C 7E                         MOV     A,M
004D CA5400                     JZ      QUAT
0050 0F                         RRC
0051 0F                         RRC
0052 0F                         RRC
0053 0F                         RRC
0054 E60F               QUAT:   ANI     0FH
0056 B9                         CMP     C
0057 C27500                     JNZ     CINQ
005A 2A1302                     LHLD    IND
005D 110D02                     LXI     D,RES
0060 19                         DAD     D
0061 110A02                     LXI     D,KMTD
0064 0603                       MVI     B,N2+1
0066 CDA800                     CALL    ABDHD
0069 D27500                     JNC     CINQ
006C 3E00               JEAN:   MVI     A,0
 96E 8E                         ADC     M
 96F 27                         DAA
0070 77                         MOV     M,A
0071 23                         INX     H
```

```
0072 DA6C00             JC      JEAN
0075 211302     CINQ:   LXI     H,IND
0078 34                 INR     M
0079 3E03               MVI     A,N1
007B BE                 CMP     M
007C C24100             JNZ     TROIS
007F 0C                 INR     C       ;K=K+1
0080 210A02             LXI     H,KMTD
0083 110702             LXI     D,MTD
0086 0603               MVI     B,N2+1
0088 CDA800             CALL    ABDHD
008B 3E0A               MVI     A,0AH
008D B9                 CMP     C
008E C23B00             JNZ     DEUX
0091 3A1202             LDA     LR
0094 B7                 ORA     A
0095 C0                 RNZ
0096 3C                 INR     A
0097 321202             STA     LR      ;LR=1 LEFT
009A 110A02             LXI     D,KMTD
009D 210702             LXI     H,MTD
00A0 0603               MVI     B,N2+1
00A2 CDB400             CALL    TBDH
00A5 C33900             JMP     UN
                ;ABDHD: THIS SUBROUTINE ADDS (B) BYTES OF MD (MEM AREA
                ;       POINTED BY D,E) TO (B) BYTES OF MH (MEM AREA POINTED
                ;       BY H,L). WHEN RETURN,(H,L) AND (D,E) POINT ON
                ;       FOLLOWING ADDRESS
                ;
                ABDHD:
00A8 AF                 XRA     A       ;CLEAR CARRY
00A9 1A                 LDAX    D
00AA 8E                 ADC     M
00AB 27                 DAA
00AC 77                 MOV     M,A
00AD 13                 INX     D
00AE 23                 INX     H
00AF 05                 DCR     B
00B0 C2A900             JNZ     ABDHD+1
00B3 C9                 RET
                ;TBDH:THIS SUBROUTINE MOVES (B) BYTES OF MD INTO MH
                ;       WHEN RETURN,(H,L) AND (D,E) POINT ON
                ;       FOLLOWING ADDRESS
                ;
                TBDH:
00B4 1A                 LDAX    D
00B5 77                 MOV     M,A
00B6 13                 INX     D
00B7 23                 INX     H
00B8 05                 DCR     B
```

```
00B9 C2B400             JNZ     TBDH
00BC C9                 RET

0200                    ORG     200H
                ;TEST PROGRAM USING MBCD SUBROUTINE
                ;MULTIPLICATION* 71532 * 367 = 26252244
                ;
0003            N1      EQU     3
0002            N2      EQU     2
0200 CD2000             CALL    MBCD
0203 76                 HLT
                ;MEMORY AREA
0204 321507     MTR:    DB      32H,15H,7
0207 6703       MTD:    DB      67H,3
0209                    DS      1
020A            KMTD:   DS      3
020D            RES:    DS      5
0212            LR:     DS      1       ;LEFT/RIGHT FLIP-FLOP
0213            IND:    DS      2       ;INDEX
0000                    END
```

Ref. # BD1

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

HISTOGRAM

Function

This program will plot a histogram graph of numeric data between the limits of 00 to 100. It may be useful for graphical analysis of grade distributions, signal quality, probability or any function which requires analysis of incidence of data.

Required Hardware

Teletype of other printer.

Required Software

TTY routines for input and output, should mask out parity on input, and not effect other registers. Input and output data value should be through accumulator.

Input Parameters

Input data is ASCII, the following commands are accepted:
numbers 0 to 99, and h (for hundred) is accepted as data to be graphed. RETURN on TTY enters data and prompts the next entry.
"e" typed on the TTY will delete the pending entry.
"p" typed on the TTY will print the histogram on the TTY.

Output Results

Graph of incidence of input numeric data.

Registers Modified: ALL	Programmer: Robert A. Mikkelson
RAM Required: 389	Company: System Services
ROM Required:	Address: 12120 Rochester Avenue
Maximum Subroutine Nesting Level: 7	City: West Los Angeles
Assembler/Compiler Used: Microkit assembler ver 1.0	State: California 90025

Ref. # BD2

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other _____ (use additional sheets if necessary)

Program Title
ASCII to EBCDIC and EBCDIC to ASCII converters

Function
To convert ASCII coded data to EBCDIC and vice versa.

Required Hardware
None

Required Software
None

Input Parameters
The accumulator holds the byte to be converted.

Output Results
The accumulator holds the converted byte.

Registers Modified: A	Programmer: W. R. Ott
RAM Required: 411 Bytes (Decimal)	Company: Applied Data Communications
ROM Required: None	Address: 1509 East McFadden Avenue
Maximum Subroutine Nesting Level:	City: Santa Ana
Assembler/Compiler Used: INTEL	State: California 92705

Ref.# BA15

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

Conversion of Scientific to easily readable notation.

Function

By adding leading or trailing zeros and/or a decimal point, this routine converts a BCD number expressed in Scientific notation (4 digit, Sign, exponent, exponent sign) into a format that is more easily read by a mathematically unskilled operator.

Required Hardware

None

Required Software

None for the program itself. The test program uses the CO subroutine of the MDS system.

Input Parameters

Enter subroutine with register pair H L pointing to desired location for the first character in the output buffer. Registers C D E are loaded with the BCD number (see details at beginning of program). This subroutine will accept the output of the FBCD subroutine in program BC-5 directly.

Output Results

A buffer is created with up to eleven ASC11 characters, terminated by an EOT character. The first character is in location pointed to by H, L. The EOT character is at a higher memory location.

Registers Modified: All	Programmer: B. A. Robinson
RAM Required: 12 bytes for output buffer	Company: Du Pont of Canada Ltd.
ROM Required: 157D bytes	Address: P.O. Box 5000
Maximum Subroutine Nesting Level: 2 (3 including the CALL to this subroutine).	City: Kingston
Assembler/Compiler Used: MDS Macro Assembler	State: Ontario, Canada K7L 5A5

```
;REF. NO. BA15
;PROGRAM TITLE "CERN" CONVERSION OF SCIENTIFIC TO
;                     EASILY READABLE NOTATION
;      ****************************************
;      *****CONVERSION OF SCIENTIFIC TO*****
;      ******EASILY READABLE NOTATION*******
;      ****************************************
;
;                   77-03-15
;
;THIS ROUTINE CONVERTS A BCD NUMBER IN REGISTERS
;C,D AND E TO AN ASCII OUTPUT BUFFER, TERMINATED
;BYAN 'E0T'
;
;ENTER THE SUBROUTINE DIRECTLY FROM THE FBCD
;ROUTINE OF PROGRAM BC-5, OR OTHER PROGRAM WITH
;THE C,D AND E REGISTERS ASSIGNED AS FOLLOWS:
;
;   REGISTER                BIT
;                7  6  5  4  3  2  1  0
;      C        SN SE EH EH EL EL EL EL
;      D        X4 X4 X4 X4 X3 X3 X3 X3
;      E        X2 X2 X2 X2 X1 X1 X1 X1
;
;WHERE SN IS THE SIGN OF THE NUMBER (0 IMPLIES
;PLUS), SE IS THE SIGN OF THE EXPONENT, EH IS
;THE TWO BIT BCD UPPER EXPONENT DIGIT, EL IS
;THE BCD LOWER EXPONENT DIGIT, AND X4 TO X1
;ARE THE BCD MANTISSA AS X.XXX.
;ENTER WITH REGISTER PAIR HL POINTING TO THE
;DESIRED LOCATION FOR THE FIRST ASCII CHARACTER.
;OUTPUT BUFFER APPEARS IN THIS AND HIGHER
;MEMORY LOCATIONS.
;
;TYPICAL OUTPUT BUFFER FORMATS ARE:
;  > 10 EXP. 39
;-> 10 EXP. 07
;-1234000
;  123400
;-12340
;  1234
;-123.4
;  12.34
;-1.234
;  .1234
;TO
;-.000001234
;NUMBERS LESS THAN 10 EXP. -07 ARE LOADED AS 0.000.
;
;
```

```
1000                    ORG     1000H
1000 79         CERN:   MOV     A,C     ;FETCH EXP.
1001 E680               ANI     80H     ;MASK
1003 CA0B10             JZ      SPACE
1006 362D               MVI     M,'-'   ;-SIGN LOADED IN BUFFER
1008 C30D10             JMP     CERN1
100B 3620       SPACE:  MVI     M,' '   ;SPACE LOADED
100D 23         CERN1:  INX     H
100E 79         EXNEG:  MOV     A,C     ;IS EXPONENT -VE?
100F E640               ANI     40H
1011 CA5A10             JZ      EXP     ;Z IMPLIES EXP. POSITIVE
1014 79                 MOV     A,C     ;IS EXP. <-6?
1015 E63F               ANI     3FH
1017 FE07               CPI     07H
1019 D25310             JNC     SMALL   ;NUMBER TOO SMALL. SET TO 0
101C 362E               MVI     M,'.'   ;LOAD A DECIMAL PT
101E 23                 INX     H
101F 79                 MOV     A,C     ;MASK SIGNS
1020 E60F               ANI     0FH
1022 4F                 MOV     C,A
1023 0D         CERN2:  DCR     C       ;STORE (C) ZEROS IN BUFFER
1024 CA2D10             JZ      STDE1
1027 3630               MVI     M,'0'
1029 23                 INX     H
102A C32310             JMP     CERN2
                ;
102D 06FF       STDE1:  MVI     B,0FFH  ;STORE (DE), NO DECIMAL PT.
102F 7A         STDE2:  MOV     A,D     ;STORE (DE), WITH DECIMAL PT.
1030 CD4310             CALL    SHSTR   ;STORE MSD FROM D IN BUFFER
1033 7A                 MOV     A,D
1034 CD4710             CALL    MKSTR   ;STORE LSD
1037 7B         STE:    MOV     A,E
1038 CD4310             CALL    SHSTR   ;STORE MSD FROM E IN BUFFER
103B 7B                 MOV     A,E
103C CD4710             CALL    MKSTR   ;STORE LSD
103F 3604       EOTIN:  MVI     M,04H   ;PLACE EOT ON TOP OF BUFFER
1041 23                 INX     H
1042 C9                 RET
                ;
1043 1F         SHSTR:  RAR             ;CONVERT (A) TO ASCII & STORE
1044 1F                 RAR
1045 1F                 RAR
1046 1F                 RAR
1047 E60F       MKSTR:  ANI     0FH     ;CONVERT TO ASCII
1049 F630               ORI     30H
104B 77                 MOV     M,A
104C 23                 INX     H
104D 05                 DCR     B       ;DECREMENT COUNTER
104E C0                 RNZ             ;NZ IMPLIES NO D.P. WANTED
104F 362E               MVI     M,'.'   ;INSERT D.P.
```

```
1051 23                 INX     H
1052 C9                 RET
                ;
1053 AF         SMALL:  XRA     A       ;NUMBER TOO SMALL.  SET TO 0
1054 4F                 MOV     C,A     ;AND RECYCLE
1055 57                 MOV     D,A
1056 5F                 MOV     E,A
1057 C30E10             JMP     EXNEG
                ;
105A 79         EXP:    MOV     A,C     ;EXP.  IS POS.
105B E63F               ANI     3FH     ;IS EXP.  >3
105D 4F                 MOV     C,A
105E FE03               CPI     03H
1060 D26810             JNC     LARGE   ;NC IMPLIES EXP.  >3
1063 3C                 INR     A
1064 47                 MOV     B,A     ;SET COUNTER TO (EXP.  +1)
1065 C32F10             JMP     STDE2   ;STORE (DE) WITH D.P.
                ;
1068 FE07       LARGE:  CPI     07H     ;EXP.>3
106A D27D10             JNC     AWFUL   ;NC IMPLIES EXP.  >6
106D CD2D10             CALL    STDE1   ;STORE NUMB.
1070 2B                 DCX     H       ;WIPE OUT EOT
1071 0D                 DCR     C       ;COMPUTE (EXP.  -3)
1072 0D                 DCR     C
1073 0D         CERN3:  DCR     C
1074 CA3F10             JZ      EOTIN   ;Z IMPLIES EOT SHOULD
                                        ;BE RE-INSERTED
1077 3630               MVI     M,'0'   ;STORE (EXP.  -3) ZEROS
1079 23                 INX     H
107A C37310             JMP     CERN3
                ;
107D C5         AWFUL:  PUSH    B       ;NUMBER TOO LARGE
107E 119410             LXI     D,MSG1  ;TRANSFER MSG TO BUFFER
1081 0E0A               MVI     C,10D   ;SET (C)= NUMB.  OF CHAR.
1083 EB         CERN4:  XCHG
1084 7E                 MOV     A,M
1085 EB                 XCHG
1086 77                 MOV     M,A
1087 23                 INX     H
1088 13                 INX     D
1089 0D                 DCR     C
108A C28310             JNZ     CERN4
108D C1                 POP     B
108E 59                 MOV     E,C     ;STORE EXP.  ON BUFFER
108F 06FF               MVI     B,0FFH
1091 C33710             JMP     STE
                ;
1094 3E203130 MSG1:     DB      '> 10 EXP
1098 20455850
109C 2E20
```

```
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref.#BD3

☐ 4004 ☐ 4040 ☐ 8008 ☐ 8080 ☐ 3000 *8048 (use additional sheets if necessary)

Program Title

DTMF TO HEX CODE

Function

CONVERTS 2-OF-7 DTMF CODE TO HEXADECIMAL.

Required Hardware

NONE

Required Software

NONE

Input Parameters

2-OF-7 DTMF IN THE ACCUMULATOR.

BIT	7	6	5	4	3	2	1	Ø
ROW				4	3	2	1	
COLUMN			3	2	1			

Output Results

RETURNS HEXADECIMAL EQUIVALENT IN THE ACCUMULATOR FOR ZERO THROUGH NINE.

RETURNS ØAH FOR *.

RETURNS ØBH FOR #.

RETURNS 8ØH IF INPUT IS INVALID.

Registers Modified: A, R7	**Assembler/Compiler Used:** ISIS - II 8048, V1.2
RAM Required: NONE	**Programmer:** FRANK FAFF
ROM Required: 4ØH BYTES	**Company:** ATLANTIC RESEARCH CORP.
Maximum Subroutine Nesting Level: ZERO	**Address:** 5390 CHEROKEE AVE. ALEXANDRIA, VA. 22314

98-034C

```
ASM48 :F1:BD3

I..S-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE     1

  LOC  OBJ          SEQ          SOURCE STATEMENT

                      1 ;REF.  NO.  BD3
                      2 ;PROGRAM TITLE DTMHEX
                      3 ;
                      4 ;
                      5 ;
                      6 ;*****************************************************************
                      7 ;
                      8 ;    DTMHEX
                      9 ;
                     10 ;*****************************************************************
                     11 ;
                     12 ;         CONVERTS 2-OF-7 DTMF CODE TO HEX.
                     13 ;         RETURNS 80H IF DTMF IS INVALID.
                     14 ;
                     15 ;         EXPECTS DTMF CODE IN A
                     16 ;         RETURNS HEX IN A
                     17 ;         USES R7
                     18 ;
                     19 ;         INPUT FORMAT
                     20 ;         BIT      7  6  5  4  3  2  1  0
                     21 ;         ROW                  4  3  2  1
                     22 ;         COLUMN      3  2  1
                     23 ;
0000 BF00            24 DTMHEX:   MOV      R7,#0     ; CLEAR HEX VALUE
0002 1213            25           JB0      ROW1      ; JUMP IF ROW 1 TONE PRESENT
0004 1F              26           INC      R7        ; OTHERWISE ADD 3 TO HEX VALUE
0005 1F              27           INC      R7
0006 1F              28           INC      R7
0007 3215            29           JB1      ROW2      ; JUMP IF ROW 2 TONE
0009 1F              30           INC      R7
000A 1F              31           INC      R7
000B 1F              32           INC      R7
000C 5217            33           JB2      ROW3
000E 722A            34           JB3      ROW4      ; ROW 4 IS SPECIAL
0010 2380            35 ERROR:    MOV      A,#80H    ; ERROR VALUE IF NO ROW
0012 83              36           RET
0013 3210            37 ROW1:     JB1      ERROR     ; ERROR IF MORE THAN ONE ROW
0015 5210            38 ROW2:     JB2      ERROR
0017 7210            39 ROW3:     JB3      ERROR
0019 1F              40           INC      R7        ; ADD ONE TO HEX VALUE
001A 9224            41           JB4      COL1      ; FOR EACH COLUMN
001C 1F              42           INC      R7
001D B226            43           JB5      COL2
001F 1F              44           INC      R7
0020 D228            45           JB6      COL3
^922 0410            46           JMP      ERROR     ; ERROR IF NO COLUMN
 J24 B210            47 COL1:     JB5      ERROR     ; ERROR IF MORE THAN ONE COLUMN
0026 D210            48 COL2:     JB6      ERROR
0028 FF              49 COL3:     MOV      A,R7      ; GET HEX VALUE
0029 83              50           RET                ; AND RETURN
002A 9232            51 ROW4:     JB4      STAR      ; TONE = *
002C B239            52           JB5      ZERO      ; ZERO
```

```
  LOC  OBJ          SEQ          SOURCE STATEMENT

  002E D23D           53          JB6     NUMB    ; #
  0030 0410           54          JMP     ERROR
  0032 B210           55 STAR:    JB5     ERROR
  0034 D210           56          JB6     ERROR
  0036 230A           57          MOV     A,#0AH  ; RETURN 0AH IF TONE = *
  0038 83             58          RET
  0039 D210           59 ZERO:    JB6     ERROR
  003B 27             60          CLR     A       ; RETURN 00 IF ZERO
  003C 83             61          RET
  003D 230B           62 NUMB:    MOV     A,#0BH  ; RETURN 0BH IF TONE = #
  003F 83             63          RET
                      64          END

USER SYMBOLS
COL1    0024    COL2    0026    COL3    0028    DTMHEX  0000    ERROR   0010    NUMB
ROW3    0017    ROW4    002A    STAR    0032    ZERO    0039

ASSEMBLY COMPLETE,  NO ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref.#BA16

☐ 4004 ☐ 4040 ☐ 8008 ☐ 8080 ☐ 3000 ☒ MCS-48 (use additional sheets if necessary)

Program Title

8048 BCD MULTIPLY
(Decimal Multiply 6 digit x 4 digit or less)

Function

The program is doing a multiply between a six digit and a four digit BCD value. The six digit value is situated in registers 2 to 4 in order of LSD to MSD and the four digit value in registers 5 and 6. The result is a ten digit number situated in registers 12 to 16 in order of LSD to MSD. Each register contains two BCD digits.

Required Hardware

Prompt 48

Required Software

In a Prompt 48 you can use system calls and by this you get a calculatorlike operation.

Input Parameters

As input parameters we have one six digit value in registers R2 to R4 and another four digit value in registers R5 and R6 These registers contain the value in the format of two four bit BCD digits in order of LSD to MSD. Result registers R12, R13, R14, R15 and R16 has to be cleared before operation.

Output Results

The result is situated in registers R12 to R16 in order of LSD to MSD. Each register contains two four bit BCD digits.

Registers Modified:	Assembler/Compiler Used:
R0 to R7 and R12 to R16	ASM 48
RAM Required:	**Programmer:**
None	Karl-Magnus Heinrichs
ROM Required:	**Company:**
61 bytes	Vaaka-Nyholm
Maximum Subroutine Nesting Level:	**Address:**
None	Finland Oskelantie 1, 00320 Helsinki32

```
 LOC  OBJ          SEQ          SOURCE STATEMENT

                     1
                     2 ;REF.  NO.  BA16
                     3 ;PROGRAM TITLE BCD-MUL 6 DIGIT X 4 DIGIT
                     4 ;
                     5 ;
                     6 ;
                     7 ;*************************************************************
                     8
                     9
                    10 ;        ***************************************************
                    11 ;        *        BCD-MUL    6 DIGIT X 4 DIGIT
                    12 ;        ***************************************************
                    13
                    14
                    15
0060                16                    ORG     60H
                    17
0060 A5             18          SUB:      CLR     F1        ;CLEAR FLAG 1
0061 FD             19                    MOV     A,R5      ;LOAD ACCUMULATOR WITH R
0062 37             20                    CPL     A         ;COMPLEMENT ACCUMULATOR
  )63 0301          21                    ADD     A,#1      ;SUBTRACT ONE FROM ACCUM
0065 E668           22                    JNC     NOBO      ;JUMP ON NO BORROW TO "N
0067 B5             23                    CPL     F1        ;SET FLAG 1
0068 57             24          NOBO:     DA      A         ;DECIMAL ADJUST OF ACCUM
0069 37             25                    CPL     A         ;COMPLEMENT ACCUMULATOR
006A 0300           26                    ADD     A,#0      ;PREPARE ACCUMULATOR FOR
006C 57             27                    DA      A         ;DECIMAL ADJUST OF ACCUM
006D AD             28                    MOV     R5,A      ;LOAD BACK ACCUMULATOR T
006E 7683           29                    JF1     BORROW    ;JUMP IF FLAG 1 IS SET T
0070 97             30          MUL:      CLR     C         ;CLEAR CARRY BIT
0071 BF03           31                    MOV     R7,#3     ;LOAD R7 WITH 3 FOR ADDR
0073 B802           32                    MOV     R0,#2     ;LOAD R0 WITH 2 FOR ADDR
0075 B912           33                    MOV     R1,#18    ;LOAD R1 WITH 18 FOR ADD
0077 F1             34          ADI:      MOV     A,@R1     ;LOAD ACCUMULATOR WITH O
0078 70             35                    ADDC    A,@R0     ;ADD ACCUMULATOR TO MULT
0079 57             36                    DA      A         ;DECIMAL ADJUST OF ACCUM
007A 18             37                    INC     R0        ;INCREMENT REGISER 0 FOR
007B A1             38                    MOV     @R1,A     ;LOAD ACCUMULATOR INTO R
007C 19             39                    INC     R1        ;INCREMENT R1 FOR NEW RE
007D EF77           40                    DJNZ    R7,ADI    ;DECREMENT R7 AMD IF NO
007F F691           41                    JC      CARRY     ;JUMP IF CARRY BIT TO "C
0081 0460           42                    JMP     SUB       ;JUMP TO "SUB" FOR NEXT
0083 FE             43          BORROW:   MOV     A,R6      ;LOAD ACCUMULATOR WITH R
                    44                                      ;OF BORROW
0084 37             45                    CPL     A         ;COMPLEMENT ACCUMULATOR
 )85 0301           46                    ADD     A,#1      ;SUBTRACT ONE FROM ACCUM
  )87 F69B          47                    JC      WAIT      ;JUMP ON BORROW TO "WAIT
0089 57             48                    DA      A         ;DECIMAL ADJUST OF ACCUM
008A 37             49                    CPL     A         ;COMPLEMENT ACCUMULATOR
008B 0300           50                    ADD     A,#0      ;PREPARE ACCUMULATOR FOR
008D 57             51                    DA      A         ;DECIMAL ADJUST OF ACCUM
008E AE             52                    MOV     R6,A      ;LOAD BACK ACCUMULATOR T
```

```
 LOC  OBJ          SEQ          SOURCE STATEMENT

                    53                    JMP     MUL     ;JUMP TO "MUL"
008F 0470           54          CARRY:    CLR     A       ;CLEAR ACCUMULATOR
0091 27             55                    RLC     A       ;ROTATE ACCUMULATOR LEFT
0092 F7             56                    ADD     A,@R1   ;ADD ACCUMULATOR TO RESU
0093 61             57                    DA      A       ;DECIMAL ADJUST OF ACCUM
0094 57             58                    MOV     @R1,A   ;LOAD BACK ACCUMULATOR
0095 A1             59                    INC     R1      ;INCREMENT R1 FOR NEXT R
0096 19             60                    JC      CARRY   ;JUMP ON "CARRY" TO "CAR
0097 F691           61                    JMP     SUB     ;JUMP TO "SUB" FOR NEXT
0099 0460           62          WAIT:     JMP     WAIT    ;RESULT IN REG. R12,R13,
009B 049B           63                    END
```

USER SYMBOLS

```
ADI    0077    BORROW 0083    CARRY  0091    MUL    0070    NOBO   0068    SUB
```

ASSEMBLY COMPLETE, NO ERRORS

Ref. #BB46

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

Hex to ASCII Conversion

Function

To convert a string of hexadecimal bytes in memory (string length variable up to 255) into an ASCII character string in memory for display or transmission.

Required Hardware

None

Required Software

Subroutine call with input parameters initialized.
Stack with pointer SP initialized to support subroutine call/return

Input Parameters

1- Hex input string in memory
2- Address of input string in H,L - registers
3- Address of output buffer in D,E - registers
4- Input byte count in B - register

(Note: Output buffer must be at least twice as long as input string)

Output Results

1- Output ASCII string in memory(output buffer)
2- Input string in memory unchanged
3- B - register = 0
4- H,L registers point 1 byte past last input byte
5- D,E registers point 1 byte past last character in output buffer

Registers Modified: A,H,L,D,E,B	Programmer: Mike Lippman
RAM Required: Depends on input string length	Company: Fluke Trendar
ROM Required: 49 bytes	Address: 630 Clyde Ave.
Maximum Subroutine Nesting Level: 2	City: Mt. View
Assembler/Compiler Used: ASM80 V1.0	State: Calif. 94043

```
                    ;REF. NO. BB46
                    ;PROGRAM TITLE HEX TO ASCII CONVERSION
                    ;
                    ;
                    ;
                    ;       SUBROUTINE HTOA
                    ;          CONVERTS A HEX INPUT STRING IN MEMORY TO AN ASCII CHARA
                    ;          IN MEMORY.  INPUT STRING LENGTH IS VARIABLE UP TO 255 B
                    ;          INPUT PARAMETERS:
                    ;              STARTING ADDRESS OF INPUT STRING IN H,L REGISTERS
                    ;              STARTING ADDRESS OF OUTPUT BUFFER IN D,E REGISTERS
                    ;              INPUT STRING BYTE COUNT IN B REGISTER
                    ;          OUTPUT PARAMETERS:
                    ;              ASCII CHARACTER EQUIVALENT OF INPUT STRING HEX DIGI
                    ;                   TO RIGHT) IN OUTPUT BUFFER
                    ;              INPUT STRING UNCHANGED
                    ;              B REGISTER = 0
                    ;              H,L REGISTERS POINT 1 BYTE PAST LAST INPUT BYTE
                    ;              D,E REGISTERS POINT 1 BYTE PAST LAST ASCII CHARACTE
                    ;           SUBROUTINE NESTING:
                    ;              2 LEVELS
                    ;
0000 7E             HTOA:   MOV     A,M     ;LOAD HEX INPUT BYTE
0001 E6F0                   ANI     0F0H    ;MASK OUT LEAST SIGNIF HEX DIGIT
0003 0F                     RRC             ;RIGHT
0004 0F                     RRC             ;  JUSTIFY
0005 0F                     RRC             ;    MOST SIGNIF
0006 0F                     RRC             ;      HEX DIGIT
0007 CD1D00                 CALL    CONV    ;CONVERT MOST SIGNIF HEX DIGIT OF BYTE T
000A 12                     STAX    D       ;STORE RESULT IN OUTPUT BUFFER
000B 13                     INX     D       ;INCREMENT OUTPUT BUFFER PTR
000C 7E                     MOV     A,M     ;RE-LOAD HEX INPUT BYTE
000D E60F                   ANI     0FH     ;MASK OUT MOST SIGNIF DIGIT
000F CD1D00                 CALL    CONV    ;CONVERS LEAST SIGNIF HEX DIGIT OF BYTE
0012 12                     STAX    D       ;STORE RESULT IN OUTPUT BUFFER
0013 13                     INX     D       ;INCREMENT OUTPUT BUFFER PTR
0014 23                     INX     H       ;INCREMENT INPUT STRING PTR
0015 05                     DCR     B       ;DECREMENT INPUT BYTE COUNTER
0016 78                     MOV     A,B     ;LOAD INPUT BYTE COUNTER
0017 FE00                   CPI     0       ;CHECK FOR 0
0019 C20000                 JNZ     HTOA    ;LOOP FOR NEXT INPUT BYTE IF COUNT NOT 0
001C C9                     RET             ;RETURN TO MAIN PROGRAM WHEN DONE
                    ;
                    ;
                    ;       SUBROUTINE CONV(CALLED BY HTOA)
                    ;           CONVERTS A SINGLE HEX DIGIT (RIGHT JUSTIFIED IN A REG)
                    ;           ASCII CHARACTER (RETURNED IN A REG)
                    ;           CONVERSION ALGORITHM:
                    ;              INPUT DIGIT 0-9 - - "OR" 30H INTO DIGIT TO FORM ASC
                    ;              INPUT DIGIT A-F - - MASK OUT MSB OF DIGIT,SUBTRACT
```

```
                        ;                                       RESULT, AND OR WITH 40H TO FORM
                        ;   NOTE: CONV PRODUCES ODD PARITY AS SHOWN, FOR EVEN PARITY CHAN
                        ;           JPE
                        ;
001D FE0A               CONV:   CPI     10      ;IS HEX DIGIT < 10 ?
001F DA2A00                     JC      NUM     ;JUMP IF 0-9
0022 E6F7                       ANI     0F7H    ;CLR MSB OF HEX DIGIT
0024 3D                         DCR     A       ;SUBTRACT 1 FROM RESULT
0025 F640                       ORI     40H     ;OR 40H TO COMPLETE ASCII CHAR
0027 C32C00                     JMP     PAR     ;GO TO SET PARITY
002A F630               NUM:    ORI     30H     ;OR WITH 30H TO COMPLETE ASCII CHAR
002C E23100             PAR:    JPO     $+5     ;IF PARITY ALREADY ODD, SKIP NEXT LINE
002F F680                       ORI     80H     ;SET ODD PARITY
0031 C9                         RET             ;RETURN TO SUBR HTOA
                        ;
                        ;
                        ;
                        ;
0000                            END
```

Ref. #BC18

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/40 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ 3000 ☐ Other _____ (use additional sheets if necessary)

Program Title

FLOATING POINT PACKAGE - OPTIMISED ULTRA FAST

Function

Performs Floating Point addition, subtraction, multiplication, division, square and square root. (16 Bit mantissa, 8 Bit exponent). square root time typically 1ms.

Required Hardware

SBC 80/10 or similar

Required Software

SBC 80/10 P Monitor or similar

Input Parameters

ASC11 String entered from console forms F.P. numbers in B, D & E and A, H & L. Numbers in ASC11 are terminated by ',' or 'RET' and operators '+' '*' '/' 's' 'r' used to select appropriate subroutines.

Output Results

F. P. number is returned in B, D & E and converted to an ASC11 string for O/P to console.

Program offered on diskette only.

Registers Modified: ALL REGISTERS USED	Programmer: S. N. COPE & S. E. EVANS
RAM Required: 1 BYTE (PLUS STACK)	Company: OXFORD UNIVERSITY
ROM Required: 1055 BYTES (INC. TEST ROUTINE)	Address: DEPT. OF ENGINEERING SCIENCE
Maximum Subroutine Nesting Level: 5 LEVELS (10 BYTES)	City: PARKS ROAD, OXFORD
Assembler/Compiler Used: MACRO ASSEMBLER VER. 1	State: ENGLAND

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

"AS2FL" - ASCII STRING TO INTEL FLOATING POINT NUMBER

Function

Converts ASCII String (variable length) to floating point number in floating point record (FPR).

Required Hardware

MDS 800 with ISIS II Linking Loader

Required Software

Intel "FPAL.LIB", Intel "LINK" & "LOCATE", PL/M80 Compiler or ASM80 Assembler

Input Parameters

Pass Pointer to 18-Byte 'FPR' in B, C Registers
Pass Pointer to String in D, E Registers.

String Format:

␣␣ ±xxxxx.xxxx ␣␣ E ± xx #

Revised 2/78

WHERE: ␣ = Optional Spaces
± = Sign ('+' Optional)
x = ASCII Characters
# = Terminating Characters (See comments in listing)

Output Results

PL/M Call : "Call AS2FL (.FPR, .STR)"

NOTES:
(1) Has not been compiled as "REENTRANT".
(2) May be added to "FPAL.LIB" if desired.
(3) For users without PL/M-80, request version which is pre-linked with PLM80.LIB (AS2FL.OBX) Otherwise, order FL2AS.OBJ

BC19A is offered as one program with BC19B.

Registers Modified: ASSUME ALL	Programmer: BART EVANS
RAM Required: VARIABLES: 24 STACK: 4	Company: DURRUM INSTRUMENT
ROM Required: CODE: 814	Address: 1228 Titan Way
Maximum Subroutine Nesting Level: 1 Level after entry	City: Sunnyvale
Assembler/Compiler Used: PL/M-80 V3.0	State: CA

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE AS2FL
OBJECT MODULE PLACED IN :F1:AS2FL.OBJ
COMPILER INVOKED BY: PLM80 :F1:AS2FL.PLM DATE(0926JAN09)

```
            $TITLE('              ***** ASCII STRING TO FLOATING POINT *****')

  1         AS2FL: DO;

  2   1     FLOAD: PROCEDURE (A,B) EXTERNAL;
  3   2         DECLARE (A,B) ADDRESS;
  4   2         END FLOAD;

  5   1     FSTOR: PROCEDURE (A,B) EXTERNAL;
  6   2         DECLARE (A,B) ADDRESS;
  7   2         END FSTOR;

  8   1     FADD: PROCEDURE (A,B) EXTERNAL;
  9   2         DECLARE (A,B) ADDRESS;
 10   2         END FADD;

 11   1     FMUL: PROCEDURE (A,B) EXTERNAL;
 12   2         DECLARE (A,B) ADDRESS;
 13   2         END FMUL;

 14   1     FDIV: PROCEDURE (A,B) EXTERNAL;
 15   2         DECLARE (A,B) ADDRESS;
 16   2         END FDIV;

 17   1     FLTDS: PROCEDURE (A,B) EXTERNAL;
 18   2         DECLARE (A,B) ADDRESS;
 19   2         END FLTDS;

            /*************************************************************************************************************/

 20   1     AS2FL: PROCEDURE (FPR$ADR, STR$ADR) PUBLIC;
 21   2         DECLARE (FPR$ADR, STR$ADR) ADDRESS;
 22   2         DECLARE FPR BASED FPR$ADR (18) BYTE;
 23   2         DECLARE STRING  BASED STR$ADR (128) BYTE;

                /* ROUTINE WILL "READ" A STRING OF ARBITRARY LENGTH,
                STOPPING WHEN AN "ILLEGAL", OR TERMINATING, CHARACTER
                OCCURS.  A TERMINATING CHARACTER IS ALWAYS ANY
                CHARACTER EXCEPT '+', '-', '.', 'E', '0' THRU '9', ALTHOUGH
                '+', '-', '.', AND 'E' MAY TERMINATE IF THEY APPEAR WHERE
                THEY SHOULD NOT.

                EXAMPLES:
                    '1*'
                    '-1.000*'
                    ' 12345.67891234*'
                    '     1 E -23*'

                NOTES:
                (1) SPACES AHEAD OF NUMBERS ARE IGNORED.
                (2) NUMBER IS ASSUMED TO BE POSITIVE UNLESS A MINUS SIGN EXISTS.
```

```
                (3) NO SPACES ALLOWED BETWEEN DIGITS.
                (4) EXPONENT IS ASSUMED TO BE POSITIVE UNLESS A MINUS SIGN EXISTS.

                */

 24   2         DECLARE

                    TRUE    LITERALLY   '11111111B',
                    FALSE   LITERALLY   '00000000B',

                    DIG  (4) BYTE,
                    NUM  (4) BYTE,
                    FRAC (4) BYTE,
                    DIV  (4) BYTE,
                    TEN  (4) BYTE DATA (0, 0, 20H, 41H),

                    EXP BYTE,
                    POS$NUM BYTE,
                    POS$EXP BYTE,
                    I BYTE;

 25   2      DIGIT: PROCEDURE;
 26   3         DIG(0) = STRING(I) - '0';
 27   3         DIG(1), DIG(2), DIG(3) = 0;
 28   3         CALL FLTDS  (.FPR, .DIG);
 29   3         CALL FSTOR  (.FPR, .DIG);
 30   3         END DIGIT;

           /*****************************************************************************************************/

 31   2         DO I = 0 TO 3;
 32   3            NUM(I), FRAC(I), DIV(I) = 0;
 33   3            END;

 34   2         DIV(0) = 1;
 35   2         CALL FLTDS  (.FPR, .DIV);
 36   2         CALL FSTOR  (.FPR, .DIV);

 37   2         EXP = 0;
 38   2         POS$NUM, POS$EXP = TRUE;
 39   2         I = 0;

 40   2         CHAR1:  /* DETERMINE SIGN OF MANTISSA */

                IF STRING(I) = ' ' THEN DO;
 42   3            I = I+1;
 43   3            GO TO CHAR1;
 44   3            END;
 45   2         ELSE IF STRING(I) = '-' THEN DO;
 47   3            POS$NUM = FALSE;
 48   3            I = I+1;
 49   3            END;
 50   2         ELSE IF STRING(I) = '+' THEN DO;
 52   3            I = I+1;
 53   3            END;

                CHAR2:  /* BRANCH ON '.' OR 'E'; ELSE COMPUTE MANTISSA */
```

```
                IF STRING(I) = '.' THEN DO;
 56   3             I = I+1;
 57   3             GO TO POINT;
 58   3             END;
 59   2          ELSE IF STRING(I) = ' ' OR STRING(I) = 'E' THEN DO;
 61   3             I = I+1;
 62   3             GO TO EXPO;
 63   3             END;
 64   2          ELSE IF STRING(I) < '0' OR STRING(I) > '9' THEN DO;
 66   3             GO TO FINAL;
 67   3             END;
 68   2          ELSE DO;
 69   3             CALL DIGIT;
 70   3             CALL FLOAD  (.FPR, .NUM);
 71   3             CALL FMUL   (.FPR, .TEN);
 72   3             CALL FADD   (.FPR, .DIG);
 73   3             CALL FSTOR  (.FPR, .NUM);
 74   3             I = I+1;
 75   3             GO TO CHAR2;
 76   3             END;

 77   2          POINT:  /* COMPUTE FRACTIONAL PART */

                IF STRING(I) >= '0' AND STRING(I) <= '9' THEN DO;
 79   3             CALL DIGIT;
 80   3             CALL FLOAD  (.FPR, .DIV);
 81   3             CALL FMUL   (.FPR, .TEN);
 82   3             CALL FSTOR  (.FPR, .DIV);
 83   3             CALL FLOAD  (.FPR, .DIG);
 84   3             CALL FDIV   (.FPR, .DIV);
 85   3             CALL FADD   (.FPR, .FRAC);
 86   3             CALL FSTOR  (.FPR, .FRAC);
 87   3             I = I+1;
 88   3             GO TO POINT;
 89   3             END;
 90   2          ELSE IF STRING(I) = ' ' OR STRING(I) = 'E' THEN DO;
 92   3             I = I+1;
 93   3             GO TO EXPO;
 94   3             END;
 95   2          ELSE GO TO FINAL;

 96   2          EXPO:   /* COMPUTE EXPONENT */

                IF STRING(I) = ' ' OR STRING(I) = 'E' THEN DO;
 98   3             I = I+1;
 99   3             GO TO EXPO;
100   3             END;
101   2          ELSE IF STRING(I) = '-' THEN DO;
103   3             POS$EXP = FALSE;
104   3             I = I+1;
105   3             END;
106   2          ELSE IF STRING(I) = '+' THEN DO;
108   3             I = I+1;
109   3             END;

                DO WHILE STRING(I) >= '0' AND STRING(I) <= '9';
```

```
111   3             EXP = (10*EXP) + (STRING(I) - '0');
112   3             I = I+1;
113   3             END;

114   2          FINAL:  /* COMBINE AND PERFORM NORMALIZATION */

                  CALL FLOAD  (.FPR, .NUM);
115   2          CALL FADD   (.FPR, .FRAC);

116   2          IF NOT POS$NUM THEN FPR(17) = 10000000B;

118   2          IF POS$EXP THEN DO I = 1 TO EXP;
120   3             CALL FMUL   (.FPR, .TEN);
121   3             END;
122   2          ELSE DO I = 1 TO EXP;
123   3             CALL FDIV   (.FPR, .TEN);
124   3             END;

125   2          END AS2FL;

126   1       END AS2FL;



MODULE INFORMATION:

     CODE AREA SIZE     = 032EH    814D
     VARIABLE AREA SIZE = 0018H     24D
     MAXIMUM STACK SIZE = 0004H      4D
     186 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

Ref.# BC19B

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

"FL2AS" - INTEL FLOATING POINT NUMBER TO ASCII STRING

Function

Converts number in floating point record (FPR) to ASCII string.

Required Hardware

Intel MDS 800 with ISIS II Linking Loader

Required Software

Intel "FPAL.LIB", Intel "LINK" & "LOCATE", PL/M-80 Compiler or ASM-80 Assembler.

Input Parameters

Pass Pointer to 18-Byte 'FPR' in B, C Registers
Pass Pointer to 14-Byte String Buffer in D, E Registers

PL/M Call : "Call FL2AS (.FPR, .STR)"

Revised 2/78

Output Results

String is filled with 14 spaces and then digits are filled in as necessary.

Format: -xxxxxxx. ␣ E -xx
See format notes in Listing.

NOTES:
(1) Has not been compiled as "REENTRANT"
(2) May be added to 'FPAL.LIB' if desired.
(3) For users without PL/M-80, request version which is pre-linked with PL/M80.LIB (FL2AS.OBX) otherwise, order FL2AS.OBJ.

BC19B is offered as one program with BC19A.

Registers Modified: ASSUME ALL	Programmer: Bart Evans
RAM Required: VARIABLE: 28 STACK: 4	Company: Durrum Instrument
ROM Required: CODE: 828	Address: 1228 Titan Way
Maximum Subroutine Nesting Level: 1 LEVEL AFTER ENTRY	City: Sunnyvale,
Assembler/Compiler Used: PL/M-80 Ver.3.0	State: CA 94086

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE FL2AS
OBJECT MODULE PLACED IN :F1:FL2AS.OBJ
COMPILER INVOKED BY: PLM80 :F1:FL2AS.PLM DATE(0931JAN09)

```
                $TITLE('            ***** FLOATNG POINT TO ASCII STRING *****')

  1             FL2AS: DO;

  2   1         FLOAD: PROCEDURE (A,B) EXTERNAL;
  3   2             DECLARE (A,B) ADDRESS;
  4   2             END FLOAD;

  5   1         FSTOR: PROCEDURE (A,B) EXTERNAL;
  6   2             DECLARE (A,B) ADDRESS;
  7   2             END FSTOR;

  8   1         FADD: PROCEDURE (A,B) EXTERNAL;
  9   2             DECLARE (A,B) ADDRESS;
 10   2             END FADD;

 11   1         FSUB: PROCEDURE (A,B) EXTERNAL;
 12   2             DECLARE (A,B) ADDRESS;
 13   2             END FSUB;

 14   1         FMUL: PROCEDURE (A,B) EXTERNAL;
 15   2             DECLARE (A,B) ADDRESS;
 16   2             END FMUL;

 17   1         FDIV: PROCEDURE (A,B) EXTERNAL;
 18   2             DECLARE (A,B) ADDRESS;
 19   2             END FDIV;

 20   1         FIXSD: PROCEDURE (A,B) EXTERNAL;
 21   2             DECLARE (A,B) ADDRESS;
 22   2             END FIXSD;

 23   1         FLTDS: PROCEDURE (A,B) EXTERNAL;
 24   2             DECLARE (A,B) ADDRESS;
 25   2             END FLTDS;

 26   1         FCMPR: PROCEDURE (A,B) EXTERNAL;
 27   2             DECLARE (A,B) ADDRESS;
 28   2             END FCMPR;

                /*************************************************************************************************/

 29   1         FL2AS: PROCEDURE (FPR$ADR, STR$ADR) PUBLIC;

                    /* ROUTINE FILLS FIRST 14 LOCATIONS OF STRING POINTED TO UPON
                    ENTRY WITH ASCII CHARACTERS REPRESENTING FLOATING POINT VALUE
                    IN THE FPR.  FORMAT IS AS FOLLOWS:

                    STRING ELEMENT NUMBER: 0  1  2  3  4  5  6  7  8  9 10 11 12 13

                    CONTENTS               -  M     M  M  M  M  M  M     E  -  X  X
```

```
                WHERE 'M' IS 7 DIGITS WITH AN ACCURACY OF +/- 1 IN THE LAST
                DIGIT AND 'X' IS 2 DIGITS OF EXPONENT

                NOTES:
                (1) LARGEST NUMBER (INFINITY) IS 3.402823 E 38
                (2) SMALLEST NUMBER GREATER THAN ZERO IS 1.175495 E-38
                (3) SIGNS ARE ONLY RETURNED IF NEGATIVE.
                (4) IF VALUE IS WITH THE RANGE OF .1 <= N <= 9999999., NO
                    EXPONENTIAL PORTION IS RETURNED. THE SPACE IS FILLED
                    WITH ZEROS.  LOCATION OF DECIMAL POINT WILL BE
                    ADJUSTED AS REQUIRED.

                (5) WHEREAS LEADING ZEROS ARE NONEXISTANT, TRAILING ZEROS
                    WILL BE FILLED IN IN ORDER TO ALWAYS RETURN A 7 DIGIT
                    STRING.

                */

30  2           DECLARE (FPR$ADR, STR$ADR) ADDRESS;
31  2           DECLARE FPR BASED FPR$ADR (18) BYTE;
32  2           DECLARE STR BASED STR$ADR (14) BYTE;

33  2           DECLARE

                    TRUE LITERALLY '11111111B',
                    FALSE LITERALLY '00000000B',

                    TMP$STR (8) BYTE,
                    NUM (4) BYTE,
                    INT (4) BYTE,
                    REM (4) BYTE,

                    P100E6 (4) BYTE DATA (20H, 0BCH, 0BEH, 4CH),
                    P10E6  (4) BYTE DATA (7FH,  96H,  18H, 4BH),
                    TEN    (4) BYTE DATA (  0,    0,  20H, 41H),

                    EXP BYTE,
                    POS$EXP BYTE,
                    (I, J) BYTE;

               /*****************************************************************************************************/

34  2       MAIN:   /* CHECK FOR ZERO; IF SO, LOAD ' 0.000000     ' AND RETURN */

                   DO I = 0 TO 13;
35  3                  STR (I) = ' ';
36  3                  END;

37  2              IF FPR (16) = 0 THEN DO;
39  3                  DO I = 1 TO 8;
40  4                      STR (I) = '0';
41  4                      END;
42  3                  STR (2) = '.';
43  3                  RETURN;
44  3                  END;
```

```
 45  2          MANTISSA$SIGN:  /* IF NEGATIVE, LOAD '-' AND NEGATE */

                IF FPR (17) = 10000000B THEN DO;
 47  3              FPR (17) = 0;
 48  3              STR (0) = '-';
 49  3              END;

 50  2          CALL FSTOR  (.FPR, .NUM);

 51  2          EXP = 100;

 52  2          NORMALIZE1: /* MULTIPLY/DIVIDE UNTIL NUMBER IS IN RANGE
                           OF 10,000,000 <= N < 100,000,000 AND KEEP
                           TRACK OF RESULTING EXPONENT        */

                CALL FLOAD  (.FPR, .NUM);
 53  2          CALL FCMPR  (.FPR, .P100E6);
 54  2          IF (FPR (0) AND 11000000B) <> 0 THEN DO; /* X >= 100,000,000 */
 56  3              EXP = EXP+1;
 57  3              CALL FLOAD  (.FPR, .NUM);
 58  3              CALL FDIV   (.FPR, .TEN);
 59  3              CALL FSTOR  (.FPR, .NUM);
 60  3              GO TO NORMALIZE1;
 61  3              END;

 62  2          NORMALIZE2:

                CALL FLOAD  (.FPR, .NUM);
 63  2          CALL FCMPR  (.FPR, .P10E6);
 64  2          IF (FPR (0) AND 00100000B) <> 0 THEN DO; /* X < 10,000,000 */
 66  3              EXP = EXP-1;
 67  3              CALL FLOAD  (.FPR, .NUM);
 68  3              CALL FMUL   (.FPR, .TEN);
 69  3              CALL FSTOR  (.FPR, .NUM);
 70  3              GO TO NORMALIZE2;
 71  3              END;

 72  2          EXTRACT$DIGITS: /* REMOVE DIGITS FROM RIGHT TO LEFT -
                               DIGIT = N - 10 * INT(N/10)
                               N' = INT(N/10)  */

                DO I = 0 TO 7;
 73  3              CALL FLOAD  (.FPR, .NUM);
 74  3              CALL FDIV   (.FPR, .TEN);
 75  3              CALL FIXSD  (.FPR, .INT);
 76  3              CALL FLTDS  (.FPR, .INT);
 77  3              CALL FMUL   (.FPR, .TEN);
 78  3              CALL FSTOR  (.FPR, .REM);
 79  3              CALL FLOAD  (.FPR, .NUM);
 80  3              CALL FSUB   (.FPR, .REM);
 81  3              CALL FIXSD  (.FPR, .REM);

 82  3              TMP$STR (7-I) = REM (0) + 90H;

 83  3              CALL FLTDS  (.FPR, .INT);
 84  3              CALL FSTOR  (.FPR, .NUM);
 85  3              END;
```

```
 86   2          ROUND$OFF:  /* ROUND TO NEAREST 7 DIGITS */

                 TMP$STR (7) = DEC (TMP$STR (7)  +   5);
 87   2          TMP$STR (6) = DEC (TMP$STR (6) PLUS 0);
 88   2          TMP$STR (5) = DEC (TMP$STR (5) PLUS 0);
 89   2          TMP$STR (4) = DEC (TMP$STR (4) PLUS 0);
 90   2          TMP$STR (3) = DEC (TMP$STR (3) PLUS 0);
 91   2          TMP$STR (2) = DEC (TMP$STR (2) PLUS 0);
 92   2          TMP$STR (1) = DEC (TMP$STR (1) PLUS 0);
 93   2          TMP$STR (0) = DEC (TMP$STR (0) PLUS 0);

 94   2          RESOLVE:    /* IF .1 <= N <= 9999999. THEN LOAD WITH SPACES
                            INSTEAD OF ' E XX' ELSE COMPUTE EXPONENT AND LOAD
                            IN 'E' FORMAT.  */

                 J = 1;

 95   2          IF EXP >= 92 AND EXP <= 99 THEN DO;
 97   3              DO I = 0 TO 6;
 98   4                  IF EXP-92 = I THEN DO;
100   5                      STR (J) = '.';
101   5                      J = J+1;
102   5                      END;
103   4                  STR (J) = (TMP$STR (I) AND 0FH) + '0';
104   4                  J = J+1;
105   4                  END;
106   3              RETURN;
107   3              END;
108   2          ELSE DO;
109   3              POS$EXP = TRUE;
110   3              IF EXP < 93 THEN DO;
112   4                  EXP = 93 - EXP;
113   4                  POS$EXP = FALSE;
114   4                  END;
115   3              ELSE EXP = EXP - 93;

116   3              DO I = 0 TO 6;
117   4                  IF I = 1 THEN DO;
119   5                      STR (J) = '.';
120   5                      J = J+1;
121   5                      END;
122   4                  STR (J) = (TMP$STR (I) AND 0FH) + '0';
123   4                  J = J+1;
124   4                  END;

125   3              STR (10) = 'E';

126   3              IF NOT POS$EXP THEN STR (11) = '-';
128   3              I = EXP/10;
129   3              STR (12) = I + '0';
130   3              STR (13) = (EXP - 10*I) + '0';
131   3              END;

132   2          END FL2AS;

133   1      END FL2AS;
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 033CH    828D
     VARIABLE AREA SIZE = 001CH     28D
     MAXIMUM STACK SIZE = 0004H      4D
     222 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

8080 FLOATING POINT A^b, EXP AND NATURAL LOG FUNCTIONS FOR USE WITH FLOATING POINT MATH PKG. BC1 and BC2

Function

A^b is evaluated by X = EXP (B*Ln(A)), and has been checked to 9.9^{99} (vs. an HP-35 calculator). A typical evaluation consumes 80-90 msec (2 MHz clock). Evaluation of Exponential and Natural Log functions over argument range with approx 7-8 digit accuracy, using scaled series evaluation.

Required Hardware

Required Software

BC1 & BC2 Floating Point Math Pkg., (Insite Section 5).

Input Parameters

A**b HL = address of Floating Point Value A
DE = address of Floating Point Value b
EXP(X) uses value in F.P. accum or at HL
Ln(X)

Returned in F.P. accum and in A, B, C, D (for call to STR sub)
In EXP (X), X is scaled by dividing by 2**N so 0.<X<0.5
EXP is evaluated using the standard series:

$EXP = 1 + X + X^2/2 + \ldots.$ which is very accurate for X<0.5

Output Results

The result is them squared N times to give the correct magnitude result. A typical evaluation consumes approx 35-50 Msec (2 MHz clock).

$$LN = 2\left[\frac{X-1}{X+1} \quad \frac{1}{3}\left(\frac{X-1}{X+1}\right)^3 \quad \frac{1}{5}\left(\frac{X-1}{X+1}\right)^5 \quad +\ldots.\right]$$

scaled by dividing by 2^N, so that $1 \leq X < 2$.

The result is then added to N*Ln (2) which represents the scaling offset. A typical evaluation also consumes 35-50 Msec. Both series are expandable to a greater number of terms by changing the operand in a CPI Instruction. Currently 7 terms for EXP and 6 for Ln.

Registers Modified: ALL	Programmer: Edward G. Perkins
RAM Required: 18-Bytes + FP math RAM	Company: Adams-Smith, Inc.
ROM Required: 161 H Bytes	Address: Summer Road
Maximum Subroutine Nesting Level: 4 + FP pkg.	City: Boxboro
Assembler/Compiler Used: MAC80 V2.3	State: MASSACHUSETTS 01719

```
                ;REF. NO. BC21
                ;PROGRAM TITLE 8080 FLOATING PT. A ,EXP & NATURAL LOG
                ;              FUNCTION FOR USE W/F.P. MATH PKG. REF. NO. BC4
                ;
                ;
                ;
                ;
                ;FLOATING EXP AND LN FUNCTIONS
                ;AND TEST ROUTINES

0800                    ORG 800H  ;TEMP LOC
0800 111C00     TOEX:   LXI D,FLOAT+4 ;EXPONENT
0803 211800             LXI H,FLOAT ;VALUE
0806 CD3208             CALL FPOWR ;VALUE**EXPONENT
0809 DB01       NDEX:   IN 1  ;PAUSE W/RESULT IN ACCS
080B 211800     TOEEX:  LXI H,FLOAT  ;X
080E CD6F12             CALL FLOD  ;LOAD
0811 CD4308             CALL FEXP  ;EXP(X)
0814 211C00             LXI H,FLOAT+4 ;STORE
0817 CD3E12             CALL FSTR
081A DB01       NDEEX:  IN  1  ;PAUSE
081C 211C00     TOLN:   LXI H,FLOAT+4  ;EXP(X)
081F CD6F12             CALL FLOD  ;LOAD IT
0822 CDE308             CALL FNLOG ;LN(EXP(X))
0825 DB01       NDLN:   IN  1  ;PAUSE W/RESULT IN ACCS
0827 76                 HLT  ;STOP RUNNING OFF END
0018            FLOAT EQU  0018H ;VALUE

                ;FLTAC - FLOAT VALUE IN A REG
0828 1E08       FLTAC:  MVI E,8  ;IND DATA IN A
082A 0600               MVI B,0  ;CLEAR B,C,D
082C 48                 MOV C,B
082D 50                 MOV D,B
082E CD0015             CALL FLT ;FLOAT
0831 C9                 RET ;DONE

                ;FPOWR - RAISE VALUE AT HL TO POWER AT DE (A**B)
                ;METHOD - X=EXP(B * LN(A))

0832 D5         FPOWR:  PUSH D ;SAVE B
0833 CD6F12             CALL FLOD ;LOAD A
0836 C4E308             CNZ FNLOG ;A NOT 0, SO TAKE LN(A)
0839 E1                 POP H  ;B
083A CD8C12             CALL FMUL ;B*LN(A)
083D C34308             JMP FEXP  ;EXP TO GET A**B

                ;FEXP - EXP(FP ACCUM), USING SERIES EVALATION
                ; EXP=1+X+X**2/2!+....

                ;ENTRY WHEN HL=ADDR OF X
```

```
0840 CD6F12   FEXPH:  CALL FLOD  ;LOAD X

0843 AF       FEXP:   XRA A  ;RESET SCALE COUNT
0844 325118           STA TXSCL
0847 CD5A12           CALL FTST  ;CHECK SIGN OF X
084A F26008           JP  FEXPP ;X IS +
084D CD5212           CALL FABS ;MAKE X +
0850 CD6008           CALL FEXPP  ;EXP(+X)



0853 21DC08           LXI H,FPONE ;EXP(-X)=1/EXP(X)
0856 CD6F12           CALL FLOD   ;1.0
0859 214418           LXI H,TEXP  ;EXP(+X)
085C CDB412           CALL FDIV   ;EXP(-X)
085F C9               RET ;DONE

              ;EXP(+X)
0860 FE80     FEXPP:  CPI 80H  ;SEE IF X>.5
0862 DA6C08           JC  FEXOK ;X<.5
              ;FOR ACCURACY, WITH FEW TERMS, X IS SCALED /2**N
              ;SO 0<=X<.5, THEN EXP(X) IS SQUARED N TIMES
0865 D67F             SUI 7FH  ;GET N FOR /2**N
0867 325118           STA TXSCL ;SAVE SCALE
086A 3E7F             MVI A,7FH  ;X<.5
086C 214018   FEXOK:  LXI H,TMPX  ;SAVE X
086F E5               PUSH H  ;SAVE
0870 CD3E12           CALL FSTR
0873 E1               POP H
0874 CD6F12           CALL FLOD  ;LOAD SCALED X, REPLACING ORIG X
0877 214818           LXI H,TERMX ;INIT TERM
087A CD3E12           CALL FSTR
087D 215018           LXI H,TDNOM ;DENOMINATOR
0880 3601             MVI M,1  ;SET AT 1
0882 21DC08           LXI H,FPONE ;1.0
0885 CDD812   FXSUM:  CALL FADD   ;1+X
0888 214418           LXI H,TEXP ;SUM
088B CD3E12           CALL FSTR
088E 3A5018           LDA TDNOM ;INC DENOM
0891 3C               INR A
0892 325018           STA TDNOM
0895 FE08             CPI MXTRM ;MAX # TERMS?
0897 CAC108           JZ  FEXPX ;YES-EXIT SCALING EXP(X)
              ;NOW CALC NEXT TERM
089A CD2808           CALL FLTAC ;FLOAT DENOM
089D 214C18           LXI H,TMPD ;SAVE IT
08A0 CD3E12           CALL FSTR
08A3 214818           LXI H,TERMX ;TERM
08A6 CD6F12           CALL FLOD
08A9 214018           LXI H,TMPX  ;X
08AC CD8C12           CALL FMUL  ;TERM*X
08AF 214C18           LXI H,TMPD ;DENOM
08B2 CDB412           CALL FDIV  ;TERM/DENOM
```

```
08B5 214818             LXI H,TERMX
08B8 CD3E12             CALL FSTR  ;STORE NEW TERM
08BB 214418             LXI H,TEXP ;SUM
08BE C38508             JMP FXSUM ;SUM IT

                    ;EXP(X) CALCULATED, SCALE IF NEEDED
08C1 3A5118         FEXPX:  LDA TXSCL ;SCALE COUNT
08C4 FE00               CPI 0
08C6 CA5A12             JZ  FTST  ;NO SCALING, X WAS <.5
                    ;SCALE EXP BY SQUARING
08C9 67                 MOV H,A   ;SET COUNT
08CA E5             FEXSQ:  PUSH H    ;SAVE COUNT
08CB 214418             LXI H,TEXP ;EXP
08CE E5                 PUSH H
08CF CD8C12             CALL FMUL  ;SQUARE IT
08D2 E1                 POP H
08D3 CD3E12             CALL FSTR  ;SAVE
08D6 E1                 POP H
08D7 25                 DCR H    ;COUNT
08D8 C2CA08             JNZ FEXSQ ;MORE
08DB C9                 RET ;DONE

0008                MXTRM EQU 8  ;GO TO X**7, GIVES 7 PLACES FOR X<.5
                        ;FOR ACCURACY 1.E-7 ERROR

08DC 81000000       FPONE:  DB 81H,0,0,0 ;1.0



1840                TMPX  EQU 1840H ;X HOLD
1844                TEXP  EQU TMPX+4 ;SERIES SUM
1848                TERMX EQU TEXP+4  ;TERM
184C                TMPD  EQU TERMX+4 ;DENOM
1850                TDNOM EQU TMPD+4  ;8 BIT DENOM INTEGER
1851                TXSCL EQU TDNOM+1 ;8 BIT SCALE COUNT

                    ;FNLOG - LN(FP ACCUM) USING SERIES FOR 1<= X < 2
                    ;LN=2*(X-1/X+1+1/3(X-1/X+!)**3+1/5(X-1/X+1)**5+..)
                    ;ACCURATE TO 6 DECIMALS OVER INTERVAL 1 => 2
                    ;RETURNS LN IN FP ACCUM

                    ;ENTRY WHEN HL=ADDR OF X
08E0 CD6F12         FNLGH:  CALL FLOD  ;LOAD X

08E3 5F             FNLOG:  MOV E,A  ;GET EXP
08E4 3E81               MVI A,81H ;SCALE X, 1<=X<2
08E6 214018             LXI H,TMPX ;STORE X
08E9 CD3E12             CALL FSTR
08EC 7B                 MOV A,E  ;ORIG EXP
08ED D681               SUI 81H  ;ADJ
08EF CD2808             CALL FLTAC ;POWER OF 2
08F2 218209             LXI H,LOG2 ;LN(2)
```

```
08F5 CD8C12             CALL FMUL  ;INIT LN
08F8 214418             LXI H,TMPY ;Y
08FB CD3E12             CALL FSTR  ;STORE
08FE 3E01               MVI A,1    ;K=1
0900 325018             STA TFNLK
0903 21DC08             LXI H,FPONE ;1.0
0906 CD6F12             CALL FLOD  ;LOAD IT
                    ;CALC TERM X-1/X+1
0909 214018             LXI H,TMPX
090C E5                 PUSH H
090D CDD812             CALL FADD  ;X+1
0910 E1                 POP H      ;SAVE
0911 E5                 PUSH H
0912 CD3E12             CALL FSTR
0915 218609             LXI H,FPTWO  ;NOW X-1
0918 CDD312             CALL FSUB
091B E1                 POP H    ;X+1
091C E5                 PUSH H
091D CDB412             CALL FDIV  ;TERM
0920 E1                 POP H    ;X
0921 E5                 PUSH H
0922 CD3E12             CALL FSTR  ;SET INIT TERM
0925 E1                 POP H      ;AND ITS SQUARE
0926 CD8C12             CALL FMUL
0929 214818             LXI H,TMPZ
092C CD3E12             CALL FSTR
092F 214018             LXI H,TMPX  ;LOAD INIT TERM
0932 CD6F12             CALL FLOD
0935 C34909             JMP FNLSM ;START SUMMING

                    ;NEW TERM = TERM * Z  (INIT TERM**2)
0938 214018         FNLC2:  LXI H,TMPX  ;TERM
093B E5                 PUSH H    ;SAVE
093C CD6F12             CALL FLOD  ;LOAD IT
093F 214818             LXI H,TMPZ  ;NEXT ODD POWER
0942 CD8C12             CALL FMUL
0945 E1                 POP H     ;STORE IT
0946 CD3E12             CALL FSTR

                    ;SUM=SUM + 2*TERM /K
0949 3C             FNLSM:  INR A    ;2*TERM
094A 215118             LXI H,TMPN  ;SAVE 2*TERM
094D E5                 PUSH H
094E CD3E12             CALL FSTR



0951 3A5018             LDA TFNLK ;GET K
0954 CD2808             CALL FLTAC  ;FLOAT IT
0957 214C18             LXI H,TMPD  ;SAVE
095A E5                 PUSH H
095B CD3E12             CALL FSTR
095E E1                 POP H     ;K
```

```
095F E3              XTHL  ;SWAP W/2*TERM
0960 CD6F12          CALL FLOD  ;2*TERM
0963 E1              POP H ;K
0964 CDB412          CALL FDIV  ;SET FOR SUM
0967 214418          LXI H,TMPY ;SUM
096A E5              PUSH H
096B CDD812          CALL FADD
096E E1              POP H
096F CD3E12          CALL FSTR  ;NEW SUM
0972 3A5018          LDA TFNLK  ;NEXT K=K+2
0975 C602            ADI 2  ;K=1,3,5,...
0977 325018          STA TFNLK
097A FE0C            CPI 12  ;6 TERMS?
097C DA3809          JC  FNLG2 ;NO-MORE
097F C35A12          JMP FTST  ;RELOAD REGS, SET CONDS


0982 80317217 LOG2:   DB 80H,31H,72H,17H ;LN(2)
0986 82000000 FPTWO:  DB 82H,0,0,0  ;2.0

1844          TMPY  EQU TEXP ;Y
1848          TMPZ  EQU TERMX ;Z
1850          TFNLK EQU TDNOM ;K
1851          TMPN  EQU TFNLK+1 ;TEMP STORE

              ;EXTERNALS
1500          FLT   EQU 1500H ;FLOAT ROUTINE
12D8          FADD  EQU 12D8H ;ADD
12D3          FSUB  EQU 12D3H ;SUBTRACT
128C          FMUL  EQU 128CH ;MULTIPLY
12B4          FDIV  EQU 12B4H ;DIVIDE
126F          FLOD  EQU 126FH ;LOAD FP ACCUM
123E          FSTR  EQU 123EH ;STORE FP ACCUM
125A          FTST  EQU 125AH ;RELOAD REGS FROM FP ACCUM
1252          FABS  EQU 1252H ;ABSOLUTE VALUE
0000                  END
```

Ref.#BC22A

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

Floating Point Utility Programs for Use with FPAL.LIB

Function

FFBCD - Converts floating point number in FAC to BCD in buffer memory: fixed point for values .1000000 to 9999999. Otherwise floating point BCD such as -1.234567E+09

FFLOAT - Converts BCD input string to floating point value in FAC. Accepts fixed and floating formats.

FLN, FLOG, FLOG2 - Natural Log, Common Log and Log Base 2 of FAC replaces FAC

FALN, FALOG, FALOG2 - Antilog Base E, 10 and 2 of FAC replaces FAC

FPWR - Raises FAC to power in memory which is pointed to by D & E registers.

FEXCH - Places FAC in temporary storage, loads memory pointed to by D & E registers, and loads D & E with temp. (used to exchange operator/operand)

IRMATH - Initializes conditions before math functions are called within the service routine and restores prior conditions after the service routine is finished with the math processor.

Required Software

FPAL.LIB

Required Hardware

No special hardware required

Output Results

BC22A is offered as one program with BC22B.

Available on diskette only for $35.00.

Registers Modified: All registers restored unless result in that register	Programmer: James C. Follansbee
RAM Required: Varies with function	Company: J F Microsystems
ROM Required: Varies with function	Address: 5617 W Argent Rd
Maximum Subroutine Nesting Level: Varies with function	City: Pasco
Assembler/Compiler Used: ISIS-II MACRO V2.0	State: Washington 99301

Ref.# BC22B

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

SBC-310 FLOATING POINT SYSTEM FOR USE WITH SINGLE OR MULTIPLE SBC-80/20 PROCESSORS

Function

INTERFACES SBC-80/20 PROGRAMS WITH HIGH SPEED MATH BOARD, SBC-310

Required Hardware

SBC-310, AT LEAST 1 SBC-80/20, MULTIBUS SUCH AS SBC-604/SBC-614
(SEE COMMENTS ON MODULE 'SBC310' FOR JUMPERS, ADDRESS SELECTIONS, INTERRUPT LEVEL, ETC.)

Required Software

NONE REQUIRED (SYSTEM IS COMPATIBLE WITH FPAL.LIB AND MAY BE USED AT THE SAME TIME BY THE SBC-80/20 WITH SOFTWARE MATH BEING DONE BY THE SBC-80/20 AND HARDWARE MATH BEING DONE BY THE SBC-310)

Input Parameters

FFSET	INITIALIZE SBC-310 MATH SYSTEM
FFLOAD	LOAD ACCUMULATOR *
FFSTOR	STORE ACCUMULATOR *
FFSTAT	STATUS BYTE INTO 'A' REGISTER
FFADD	ADD OPERATOR TO ACCUMULATOR *
FFSUB	SUBTRACT OPERATOR FROM ACCUMULATOR *
FFMUL	MULTIPLY ACCUMULATOR BY OPERATOR *
FFDIV	DIVIDE ACCUMULATOR BY OPERATOR *
FFIXSD	CONVERT FLOATINT ACCUMULATOR TO FIXED VALUE IN MEMORY *
FFLTDS	CONVERT FIXED MEMORY TO FLOATING ACCUMULATOR *
FFCMPR	COMPARE ACCUMULATOR TO MEMORY VALUE *
FFZTST	COMPARE ACCUMULATOR TO ZERO
FFNEG	CHANGE SIGN OF ACCUMULATOR
FFCLR	CLEAR ACCUMULATOR TO ZERO
FFABS	ABSOLUTE VALUE OF ACCUMULATOR
FFSQR	SQUARE ACCUMULATOR
FFSQRT	SQUARE ROOT OF ACCUMULATOR
FFSAVE	ACCUMLATOR INTO TEMPORARY REGISTER
FFRSTR	RESTORE TEMPORARY REGISTER TO ACCUMULATOR

Output Results

* OPERATOR OR MEMORY POINTED TO BY D & E REGISTERS

Note: BC22B is offered as one program with BC22A.

Registers Modified: ALL REGISTERS/FAC RESTORED UNLESS RESULT THEREIN	Programmer: JAMES C. FOLLANSBEE
RAM Required: VARIES WITH FUNCTION	Company: J F MICROSYSTEMS
ROM Required: VARIES WITH FUNCTION	Address: 5617 W ARGENT RD
Maximum Subroutine Nesting Level: VARIES WITH FUNCTION	City: PASCO
Assembler/Compiler Used: ISIS-II MACRO V2.0	State: WASHINGTON 99301

Ref.# BC23

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☐8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

RELOCATABLE FMATH AND XMATH, 8085 FLOATING POINT PACKAGE

Function

Floating Point Routines
Integer/Fractional Part
Square Root
Log Base E
Exponential, E↑X
Log Base 1Ø
10↑X
Real Base to Real Exponent A↑X
Trig SIN, COS, and TAN
ARCSIN, ARCCOS, AND ARCTAN
Polynomial Expander
Degrees <- -> Radians Conversions

Package contains subroutines for Addition, Subtraction, Multiplication, Division, Negate, Absolute Value and Test of Floating Point Numbers.

Required Hardware

8080

Required Software

ISIS II for system generation

Input Parameters

Output Results

Source available on diskette for $35.00.
Source Listing available for $15.00.

Registers Modified: See documentation	Programmer: Richard Allen
RAM Required:	Company: Texas Microsystems, Inc.
ROM Required:	Address: 6610 Harwin, Suite 125
Maximum Subroutine Nesting Level:	City: Houston
Assembler/Compiler Used: ISIS II MACRO ASSEMBLER	State: Texas 77036

2/6/78

98-034D

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☒ 8048 ☐ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

8048 - DIV -- DIVISION ROUTINE

Function

R34 = R23/A = remainder

Required Hardware

8748 or 8035

Required Software

None

Input Parameters

R23 = 16 bit dividend
A = 8 bit divisor

Output Results

R34 = 16 bit result
R2 = 8 bit remainder

Registers Modified: A, R0, 2, 3, 4, 5	Programmer: H. Serindat
RAM Required: None	Company: Societe ECA
ROM Required: 47 Bytes	Address: Z.I. Toulon-EST
Maximum Subroutine Nesting Level: 1	City: 83087 Toulon-Cedex
Assembler/Compiler Used:	State: France

```
;REF.  NO.  BB47
;PROGRAM TITLE 8048-DIV -- DIVISION SUBROUTINE
;
;
;
;  DIV - DIVISION SUBROUTINE  R34=R23/A
;                DIVIDEND = R2,3
;                DIVISOR = A
;                QUOTIENT = R3,4
;                REMAINDER = R2
;                REGISTERS MODIFIED: A, R0, R2, R3, R4, R5
;
DIV:     MOV     R5,#9    ;INIT COUNTER
         MOV     R0,A     ;R0 = DIVISOR
DV0:     CLR     C        ;DIVISOR NORMALIZATION
         JB7      DV1       ;IF BIT7 = 0
         CPL     A        ;OR
         ADD     A,R2     ;IF R2 > DIVISOR:
         JNC     DV1
         CPL      A
         INC     R5       ;         R5 = R5+1
         MOV      A,R0      ;          DIVISOR * 2
         RL       A
         MOV     R0,A
         JMP     DV0
DV1:     MOV     R4,#0    ;ELSE : DIVISION
         MOV     A,R2
DV2:     CPL      A
         JC       DV3       ;IF C = 1: SUBTRACTION AND
         ADD      A,R0      ;                    C = 0, ELSE:
         JC       DV5       ;IF R2 < R0: ROTATE
         JMP      DV4       ;ELSE:
DV3:     ADD      A,R0
         CLR      C
DV4:     CPL      A         ;SUBTRACTION
         MOV      R2,A      ;          R2 = R2-R0
DV5:     CPL      C         ;C = RESULT LSB PARTIAL
         MOV      A,R4      ;ROTATE LEFT R2,3,4 WITH C
         RLC      A
         MOV      R4,A
         MOV      A,R3
         RLC      A
         MOV      R3,A
         MOV      A,R2
         RLC      A
         MOV      R2,A
         DJNZ     R5,DV2
         RRC      A         ;REMAINDER NORMALIZATION
         MOV      R2,A
         RET
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

ARRAY ADDRESSING SUBROUTINE AND CALLING MACRO

Function

Addresses individual array elements (1, 2 or 3 dimensional arrays) via subscripts. See description on separate sheet.

Required Hardware

MDS with console output.

Required Software

Uses MDS console output subroutine CO.

Input Parameters

The calling macro (ARSET) defines all the three array dimensions (ISIZE, JSIZE, KSIZE) and the number of bytes occupied by each array element (IW). Locations I, J, K contain the subscripts of the array element whose address is required.

Output Results

Subroutine ARRAY returns the address of the (I, J, K) element of the array in the H, L registers.

ARRAY SIZE LIMITATIONS

ZERO, ISIZE, JSIZE, KSIZE, IW must each lie between 0 and 255 dec.
ZERO + (ISIZE x JSIZE x KSIZE x IW) product must not exceed 64K dec.

LIMITATIONS ON LOCATION OF ARRAY IN RAM OR ROM

None.

Registers Modified: H, L	Programmer: Roy G. Witton
RAM Required: 10 bytes dec.	Company: GKN Sankey Limited
ROM Required: ARRAY subroutine uses 139 bytes dec.	Address: Tweedale Industrial Site, Madeley,
Maximum Subroutine Nesting Level: 1 level used	City: Telford,
Assembler/Compiler Used: 8080 MDS Macro Assembler Ver. 1.0	State: Salop TF7 4JR ,England. Tel: Telford 586261

Macro for Initialisation and Calling of ARRAY Subroutine (ARSET)

The array initialisation macro is used to transfer arguments to the ARRAY subroutine via the PLIST array and the H, L registers. The macro is entered in the program as shown below:-

```
ARSET   ZERO, IW, ISIZE, JSIZE, KSIZE
```

ARSET is the macro name.
ZERO represents the base address (name) of the array being addressed.
IW represents the word length of the data contained in the array being addressed.
ISIZE, JSIZE, KSIZE represent the dimensions of the array being addressed, giving the maximum values that the subscripts (I, J, K) can attain.

The macro stores the values given to IW, ISIZE, JSIZE, KSIZE in the PLIST array and loads the H, L registers with the base address of the array being addressed, finally calling the ARRAY subroutine which returns the array address, indexed by I, J and K to the H, L registers.

One dimensional arrays may be given ISIZE, JSIZE, KSIZE values such as 1, 10, 0 or 1, 1, 10 or 10, 0, 0 depending which subscript is being used (J, K or I in these examples). Two dimensional arrays may be given values such as 1, 10, 5 or 10, 5 0 or 10, 1, 5 depending which two subscripts are being used (J, K or I, J or I, K in these examples). Three dimensional arrays must have all three values quoted, such as 20, 10, 5 or 5, 20, 10 or 10, 5, 20 as all three subscripts are being used.

Subroutine to Address Array Elements via Subscripts (ARRAY, INDX)

This subroutine determines the address of the data word in an array that corresponds to the subscripts held in the locations I, J and K. Arguments are transfered via the PLIST array and the H, L registers by the ARSET macro. These arguments transfer the values of the data word length, the array dimensions and the base address of the array. The subroutine will handle up to three dimensions, the algorithm used being:-

$$ADDR = ZER\emptyset + (I \times IW)^{*} + (J \times IW \times ISIZE)^{**} + (K \times IW \times ISIZE \times JSIZE)$$

ADDR represents the final indexed address returned by the subroutine.
ZERØ represents the base address (name) of the array.
IW represents the word length of the data in the array.
ISIZE, JSIZE, KSIZE represent the array dimensions
I, J, K represent the subscripts.

To save operating time the algorithm may be terminated at * for one dimensional arrays by passing JSIZE and/or KSIZE dimensions of zero. It may be terminated at ** for two dimensional arrays by passing a KSIZE dimension of zero.

.... Continued

As a program development aid an error message (*M) is output if the ISIZE argument is set to zero (no dimensions), or if any of the indices (I, J, or K) exceeds the array dimensions (ISIZE, JSIZE, KSIZE) indicating that a subscript has exceeded the maximum allowed array size.

The indexing is done, by double precision addition, adding the H, L registers to the D, E registers, in the INDX subroutine, the final indexed address being returned in the H, L registers.

```
ASM80 :F1:ARRAY MACROFILE

ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0          MODULE   PAGE    1

 LOC  OBJ          SEQ          SOURCE STATEMENT

                     1 ;         ARRAY ADDRESSING VIA SUBSCRIPTS
                     2 ;
                     3 ;
                     4 ;
                     5 ;         MDS ADDRESSES USED
                     6 ;
F809                 7 CO        EQU     0F809H  ; CONSOLE OUTPUT
                     8 ;
                     9 ;
                    10 ;
                    11 ;         RAM LOCATIONS USED
                    12 ;
1000                13 I         EQU     1000H   ; I SUBSCRIPT (INDEX)
1002                14 J         EQU     I+2     ; J SUBSCRIPT (INDEX)
1004                15 K         EQU     J+2     ; K SUBSCRIPT (INDEX)
1006                16 PLIST     EQU     K+2     ; SUBROUTIN ARGUMENTS
2000                17 TABLE     EQU     2000H   ; ARRAY BEING ADDRESSED
                    18 ;
                    19 ;
                    20 ;
                    21 ;         MACRO TO INITIALISE & CALL ARRAY SUBRTN
                    22 ARSET     MACRO   ZERO,IW,ISIZE,JSIZE,KSIZE
                    23 ;
                    24           LXI     H,(IW SHL 8 XOR ISIZE) AND 0FFFFH
                    25           SHLD    PLIST   ; STORE ARGS IN PARAMETER LIST
                    26           LXI     H,(JSIZE SHL 8 XOR KSIZE) AND 0FFFFH
                    27           SHLD    PLIST+2 ; STORE ARGS IN PARAMETER LIST
                    28           LXI     H,ZERO  ; TO ADDR ZERO LOCN OF ARRAY
                    29           CALL    ARRAY   ; CALL ARRAY SUBRTN
                    30           ENDM
                    31 ;
                    32 ;
                    33 ;
                    34 ;         TEST PROGRAM
                    35 ;
0100                36           ORG     0100H
0100 310030         37           LXI     SP,3000H        ; SET STACK POINTER
0103 C32001         38           JMP     TEST    ; BEGIN TEST PROGRAM
0120                39           ORG     0120H
                    40 TEST:     ARSET   TABLE,4,4,4,4
                    41+;
0120 210404         42+          LXI     H,(4 SHL 8 XOR 4) AND 0FFFFH
0123 220610         43+          SHLD    PLIST   ; STORE ARGS IN PARAMETER LIST
0126 210404         44+          LXI     H,(4 SHL 8 XOR 4) AND 0FFFFH
0129 220810         45+          SHLD    PLIST+2 ; STORE ARGS IN PARAMETER LIST
012C 210020         46+          LXI     H,TABLE ; TO ADDR ZERO LOCN OF ARRAY
012F CD0002         47+          CALL    ARRAY   ; CALL ARRAY SUBRTN
0132 76             48           HLT
0140                49           ORG     0140H
                    50           ARSET   TABLE,4,4,4,0
                    51+;
0140 210404         52+          LXI     H,(4 SHL 8 XOR 4) AND 0FFFFH
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

0143 220610         53+        SHLD    PLIST   ; STORE ARGS IN PARAMETER LIST
0146 210004         54+        LXI     H,(4 SHL 8 XOR 0) AND 0FFFFH
0149 220810         55+        SHLD    PLIST+2 ; STORE ARGS IN PARAMETER LIST
014C 210020         56+        LXI     H,TABLE ; TO ADDR ZERO LOCN OF ARRAY
014F CD0002         57+        CALL    ARRAY   ; CALL ARRAY SUBRTN
0152 76             58         HLT
0160                59         ORG     0160H
                    60         ARSET   TABLE,4,4,0,0
                    61+;
0160 210404         62+        LXI     H,(4 SHL 8 XOR 4) AND 0FFFFH
0163 220610         63+        SHLD    PLIST   ; STORE ARGS IN PARAMETER LIST
0166 210000         64+        LXI     H,(0 SHL 8 XOR 0) AND 0FFFFH
0169 220810         65+        SHLD    PLIST+2 ; STORE ARGS IN PARAMETER LIST
016C 210020         66+        LXI     H,TABLE ; TO ADDR ZERO LOCN OF ARRAY
016F CD0002         67+        CALL    ARRAY   ; CALL ARRAY SUBRTN
0172 76             68         HLT
0180                69         ORG     0180H
                    70         ARSET   TABLE,4,4,1,4
                    71+;
0180 210404         72+        LXI     H,(4 SHL 8 XOR 4) AND 0FFFFH
0183 220610         73+        SHLD    PLIST   ; STORE ARGS IN PARAMETER LIST
0186 210401         74+        LXI     H,(1 SHL 8 XOR 4) AND 0FFFFH
0189 220810         75+        SHLD    PLIST+2 ; STORE ARGS IN PARAMETER LIST
018C 210020         76+        LXI     H,TABLE ; TO ADDR ZERO LOCN OF ARRAY
018F CD0002         77+        CALL    ARRAY   ; CALL ARRAY SUBRTN
0192 76             78         HLT
01A0                79         ORG     01A0H
                    80         ARSET   TABLE,4,1,4,4
                    81+;
01A0 210104         82+        LXI     H,(4 SHL 8 XOR 1) AND 0FFFFH
01A3 220610         83+        SHLD    PLIST   ; STORE ARGS IN PARAMETER LIST
01A6 210404         84+        LXI     H,(4 SHL 8 XOR 4) AND 0FFFFH
01A9 220810         85+        SHLD    PLIST+2 ; STORE ARGS IN PARAMETER LIST
01AC 210020         86+        LXI     H,TABLE ; TO ADDR ZERO LOCN OF ARRAY
01AF CD0002         87+        CALL    ARRAY   ; CALL ARRAY SUBRTN
01B2 76             88         HLT
01C0                89         ORG     01C0H
                    90         ARSET   TABLE,4,1,4,0
                    91+;
01C0 210104         92+        LXI     H,(4 SHL 8 XOR 1) AND 0FFFFH
01C3 220610         93+        SHLD    PLIST   ; STORE ARGS IN PARAMETER LIST
01C6 210004         94+        LXI     H,(4 SHL 8 XOR 0) AND 0FFFFH
01C9 220810         95+        SHLD    PLIST+2 ; STORE ARGS IN PARAMETER LIST
01CC 210020         96+        LXI     H,TABLE ; TO ADDR ZERO LOCN OF ARRAY
01CF CD0002         97+        CALL    ARRAY   ; CALL ARRAY SUBRTN
01D2 76             98         HLT
01E0                99         ORG     01E0H
                   100         ARSET   TABLE,4,1,1,4
                   101+;
01E0 210104        102+        LXI     H,(4 SHL 8 XOR 1) AND 0FFFFH
01E3 220610        103+        SHLD    PLIST   ; STORE ARGS IN PARAMETER LIST
01E6 210401        104+        LXI     H,(1 SHL 8 XOR 4) AND 0FFFFH
01E9 220810        105+        SHLD    PLIST+2 ; STORE ARGS IN PARAMETER LIST
01EC 210020        106+        LXI     H,TABLE ; TO ADDR ZERO LOCN OF ARRAY
01EF CD0002        107+        CALL    ARRAY   ; CALL ARRAY SUBRTN
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

01F2 76            108          HLT
                   109 ;
                   110 ;
                   111 ;
                   112 ;        SUBRTN TO ADDR ARRAY ELEMENTS VIA SUBSCRIPTS
                   113 ;
0200               114          ORG     0200H
0200 F5            115 ARRAY:   PUSH    PSW      ; SAVE A REG
0201 C5            116          PUSH    B        ; SAVE B,C REGS
0202 D5            117          PUSH    D        ; SAVE D,E REGS
0203 E5            118          PUSH    H        ; SAVE ADDR OF ZERO LOCN
0204 3A0610        119          LDA     PLIST    ; I SIZE TO A REG
0207 FE00          120          CPI     0        ; I SIZE = ZERO ?
0209 CA7402        121          JZ      ERRX     ; NO 1ST DIMENSION - ERROR
020C 57            122          MOV     D,A      ; I SIZE TO D REG
020D 3A0010        123          LDA     I        ; I INDEX TO A REG
0210 BA            124          CMP     D        ; COMPARE INDEX WITH SIZE
0211 D27402        125          JNC     ERRX     ; INDEX >= SIZE
0214 2A0010        126          LHLD    I        ; I INDEX TO H,L REGS
0217 EB            127          XCHG             ; I INDEX TO D,E REGS
0218 3A0710        128          LDA     PLIST+1  ; WORD LENGTH (IW) TO A REG
021B CD8102        129          CALL    INDX     ; INDEX TO (I*IW)
021E E1            130          POP     H        ; RESTORE ZERO ADDR
021F 19            131          DAD     D        ; INDEX ZERO ADDR
0220 E5            132          PUSH    H        ; SAVE CURRENT ADDR
0221 3A0910        133          LDA     PLIST+3  ; J SIZE TO A REG
0224 FE00          134          CPI     0        ; J SIZE = ZERO ?
0226 C22E02        135          JNZ     AR1      ; 2ND DIMENSION
0229 E1            136 RET1:    POP     H        ; RETURN CURRENT ADDR - FINISHED
022A D1            137 RET2:    POP     D        ; RESTORE D,E REGS
022B C1            138          POP     B        ; RESTORE B,C REGS
022C F1            139          POP     PSW      ; RESTORE A REG
022D C9            140          RET              ; RETURN TO CALLER
022E 57            141 AR1:     MOV     D,A      ; J SIZE TO D REG
022F 3A0210        142          LDA     J        ; J INDEX TO A REG
0232 BA            143          CMP     D        ; COMPARE INDEX WITH SIZE
0233 D27402        144          JNC     ERRX     ; INDEX >= SIZE
0236 2A0210        145          LHLD    J        ; J INDEX TO H,L REGS
0239 EB            146          XCHG             ; J INDEX TO D,E REGS
023A 3A0710        147          LDA     PLIST+1  ; WORD LENGTH TO A REG
023D CD8102        148          CALL    INDX     ; INDEX TO (J*IW)
0240 3A0610        149          LDA     PLIST    ; I SIZE (ISZ) TO A REG
0243 CD8102        150          CALL    INDX     ; INDEX TO ((J*IW)*ISZ)
0246 E1            151          POP     H        ; RESTORE CURRENT ADDR
0247 19            152          DAD     D        ; INDEX CURRENT ADDR
0248 E5            153          PUSH    H        ; SAVE CURRENT ADDR
0249 3A0810        154          LDA     PLIST+2  ; K SIZE TO A REG
024C FE00          155          CPI     0        ; K SIZE = ZERO ?
024E CA2902        156          JZ      RET1     ; NO 3RD DIMENSION - FINISHED
0251 57            157 AR2:     MOV     D,A      ; K SIZE TO D REG
0252 3A0410        158          LDA     K        ; K INDEX TO A REG
0255 BA            159          CMP     D        ; COMPARE INDEX WITH SIZE
0256 D27402        160          JNC     ERRX     ; INDEX >= SIZE
0259 2A0410        161          LHLD    K        ; K INDEX TO H,L REGS
025C EB            162          XCHG             ; K INDEX TO D,E REGS
```

```
LOC  OBJ       SEQ          SOURCE STATEMENT

025D 3A0710      163          LDA     PLIST+1 ; WORD LENGTH TO A REG
0260 CD8102      164          CALL    INDX    ; INDEX TO (K*IW)
0263 3A0610      165          LDA     PLIST   ; I SIZE TO A REG
0266 CD8102      166          CALL    INDX    ; INDEX TO ((K*IW)*ISZ)
0269 3A0910      167          LDA     PLIST+3 ; J SIZE (JSZ) TO A REG
026C CD8102      168          CALL    INDX    ; INDEX TO (((K*IW)*ISZ)*JSZ)
026F E1          169          POP     H       ; RESTORE CURRENT ADDR
0270 19          170          DAD     D       ; INDEX CURRENT ADDR
0271 C32A02      171          JMP     RET2    ; FINISHED 3RD DIMENSION;
0274 0E2A        172 ERRX:    MVI     C,'*'   ; OUTPUT *
0276 CD09F8      173          CALL    CO
0279 0E4D        174          MVI     C,'M'   ; OUTPUT M
027B CD09F8      175          CALL    CO
027E C32902      176          JMP     RET1    ; RETURN CURRENT ADDR
                 177 ;
                 178 ;
                 179 ;
                 180 ;        SUBRTN TO CALCULATE INDEX
                 181 ;
0281 210000      182 INDX:    LXI     H,0     ; ZERO BASE
0284 19          183 IND1:    DAD     D       ; ADD INDEX TO BASE,
0285 3D          184          DCR     A       ; NO. OF TIMES SHOWN,
0286 C28402      185          JNZ     IND1    ; IN A REG
0289 EB          186          XCHG            ; RESULTANT INDEX TO D,E REGS
028A C9          187          RET             ; RETURN TO CALLER
                 188          END
```

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

AR1	A 022E	AR2	A 0251	ARRAY	A 0200	ARSET	+ 0000	CO	A F80		
IND1	A 0284	INDX	A 0281	J	A 1002	K	A 1004	PLIST	A 100		
TABLE	A 2000	TEST	A 0120								

ASSEMBLY COMPLETE, NO ERRORS

Ref.# BC24

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

FLOATING POINT CONVERSION ROUTINE (for use with FPAL.LIB)

Function

CONVERT FREE FORM ASCII STRING TO FLOATING POINT NUMBER. CONVERT FLOATING NUMBER TO ASCII STRING WITH FORMAT CONTROL.

Required Hardware

STANDARD 8080 OR 8085 HARDWARE

Required Software

INTEL FPAL.L1B

Input Parameters

THE INPUT PARAMETERS CONFORM TO PLM CONVENTION. THE ASCII TO FLOATING POINT ROUTINE, ENCODE, IS CALLED WITH THREE ARGUMENTS:
B-C=STRING ADDRESS
D-E=FP NUMBER ADDRESS
STACK+2=FP RECORD ADDRESS

THE FLOATING POINT TO ASCII ROUTINE, DECODE, IS CALLED WITH FIVE ARGUMENTS:
B-C=STRING ADDRESS
D-E=FP NUMBER ADDRESS
STACK+2=FP RECORD ADDRESS
STACK+4=FRACTIONAL PRECISION
STACK+6=STRING LENGTH
SEE ATTACHED DOCUMENTATION INCLUDED WITH LISTING

Output Results

THE OUTPUT RESULTS CONFORM TO PLM CONVENTION:
A=FP STATUS FIELD
H-L=FP ERROR FIELD
ENCODE LOADS THE FP NUMBER AT THE FP NUMBER ADDRESS. DECODE FILLS THE STRING TO THE LENGTH AND PRECISION SPECIFIED. LEADING ZERO'S ARE SUPPRESSED.
SEE ATTACHED DOCUMENTATION INCLUDED WITH LISTING

Registers Modified: A,D,E,H,L	Programmer: P. M. Callihan
RAM Required: 23 Bytes	Company: Goodyear Atomic Corp.
ROM Required: 587 Bytes	Address: P. O. Box 628
Maximum Subroutine Nesting Level: 3 Max stack 36-Includes caller&FPAL.L.1B	City: Piketon
Assembler/Compiler Used: ISIS ASSEMBLER	State: Ohio, 45661

Ref.#BC25

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

LISY -- Linear System (Gauss Elimination) - (FPAL)

Function

The routine solves the linear system [A] x = B of order N with the Gauss elimination method.

Required Hardware

None (Math. routine)

Required Software

8080/8085 Floating point arithmetic library - FPAL
Help : Diagnostic routine called in case of math error.

Input Parameters

[A] Coefficient matrix = N.N.4 Locations ; symbolic name ACO
B Coefficient vector = N.4 Locations ; symbolic name BCO
System order = 1 Location ; symbolic name SYSORD

Output Results

x unknown vector = N x 4 Locations ; symbolic name XV

Registers Modified: ALL	Programmer: G. DE GRANDI
RAM Required: 12H	Company: COMMISSION OF EUROPEAN COM-
ROM Required: 2A8H	Address: MUNITIES, JRC EURATOM, ELECTRONIC DIVISION
Maximum Subroutine Nesting Level: 2	City: 20127 ISPRA (VARESE)
Assembler/Compiler Used: ISIS II 8080/8085 MACRO ASSEM.	State: ITALY

Ref.#BC26

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 XX8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

Gamma Function Subroutine

Function

Performs evaluation of the gamma function with the interval (-34, 34) as its domain.

Required Hardware

NONE

Required Software

8008 Floating Point Math Package, Insite User's Library Ref.#BC1, reassembled for the 8080.

Input Parameters

Argument of Gamma Function (ARGX) in Floating Point Format (4 bytes).

Output Results

Result of Gamma Function (GAMMA) in Floating Point Format (4 bytes). ERROR CODE:

0 - No error.

1 - The absolute value of the argument is greater than 34. The result of Gamma Function is set to zero.

2 - The argument is within 0.000001 of a pole (i.e., zero and negative integers). The result of Gamma Function is set to zero.

Registers Modified: ALL	Programmer: William D. Becher & Joseph W. Wilhelm
RAM Required: 18 bytes	Company: The Univ. of Michigan-Dearborn
ROM Required: 494 bytes	Address: 4901 Evergreen Rd.
Maximum Subroutine Nesting Level: 1 (exclusive of Floating Point Math Package)	City: Dearborn,
Assembler/Compiler Used: 8080 MACRO Assembler, V.2.4	State: Michigan 48128

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

PLM MULTIPLE PRECISION ARITHMETIC

Function

Multiple precision twos complement arithmetic package includes addition, subtraction, multiplication, division and convert to decimal. The maximum precision is defined by assembly equates.

Required Hardware

8080/8085

REVISED 3/25/79

Required Software

Input Parameters

Each subroutine requires the address and length of each field. The parameters are passed using PLM conventions. See the program listing for interface for each subroutine.

Output Results

The package performs the required function and stores the output in the user designated memory locations.

Registers Modified: ALL	Programmer: J. Hiley
RAM Required: 36 Bytes	Company: Vector International
ROM Required: 488 Bytes	Address: Research Park
Maximum Subroutine Nesting Level: 2	City: B-3044 Haasrode
Assembler/Compiler Used: ISIS-II ASM80	State: Belgium

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 XX 8048 ☐ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

MATH48

Function

Performs multi-precision arithmetic using the MCS-48 family of microprocessors. The operations ADD, SUBTRACT, MULTIPLY and DIVIDE are supported using a memory to memory operational format. Other functions provided as part of the subroutine package, which can be used separately, include two's complement, shift left, shift right and the setting of values to zero.

Required Hardware

Any MCS-48 processor.

Required Software

None

Input Parameters

The precision of the data to be operated on plus the areas in RAM for variable storage must be defined, as per the program EQUATES, before assembly is carried out. All routines are called as subroutines and linkage registers are defined in program documentation.

Output Results

Computed values are deposited in the defined memory locations

REVISED 7/79

Registers Modified: A, R0, R1, R2, R3, R4 depending on the function performed	Programmer: D. Holden
RAM Required: bytes=4 X precision of data	Company: Miltope
ROM Required: 96H	Address: 16 Hancock Street
Maximum Subroutine Nesting Level: 2	City: Plainville
Assembler/Compiler Used: ASM48 V2.1	State: MA 02762

Ref.#BC27

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

Acquisition of a Decimal Number from MDS Console, Conversion to FPAL Floating Point Number and Vice Versa

Function

(See reverse page 5-312)

Required Hardware

MDS 800 System

Required Software

Intel FPAL Library

Input Parameters

Decimal number from MDS Console

Output Results

Floating point decimal number on MDS Console

Available on non-system diskette only for $35.00 (source & object code included)

Registers Modified:	ALL	**Programmer:**	G.DeGrandi, N.Coppo
RAM Required:	5FH	**Company:**	Commission of European Communities JRC Est. of Ispra
ROM Required:	D9EH	**Address:**	Electronics Division
Maximum Subroutine Nesting Level:		**City:**	21020 Ispra (Varese)
Assembler/Compiler Used:	ISIS-II 8080/8085 Macro Assembler V2.0	**State:**	ITALY

The program consists of a set of relocatable modules and performs a conversion from an ASCII string to a floating point number in the Intel floating point format and vice versa. The converted ASCII string represents a floating point decimal number. The modules can be linked together to implement a particular function. A test program is included which acquires a decimal number from the console of an MDS system, converts it into a floating point number in the Intel FPAL format, then converts it back into a ASCII string which represents a floating point decimal number to be output on the console. The program entails the following modules:

TCOFLO : a test module which calls all the routines which are necessary for the above mentioned conversion.

ASCFL : the routine acquires a decimal number from console and converts it into an Intel FPAL floating point number.

COFLO : the routine converts an Intel FPAL floating point number into an ASCII string representing a decimal floating point number.

FBFBCD : the routine converts an Intel FPAL floating point number into a packed BCD floating point number.

BCDASC : the routine converts a packed BCD string into an ASCII string.

Moreover, the modules use : a mathematical library which contains a multibyte shift routine, multibyte clear, multibyte addition and BCD adjust ; I/O routines.

Ref.#BC28

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

Double Precision Floating Point Package (DFPAL.LIB)

Function

To expand FPAL.LIB to include double precision functions

Required Hardware

MDS-800 with DOS

Required Software

ISIS II, FPAL.LIB

Input Parameters

See manual

Output Results

See manual

Available on non-system diskette only with manual for $45.00.

Registers Modified:	Programmer: Larry Brookwell & M. Master
RAM Required:	Company: University of Ottawa Electrical Engrg. Dept.
ROM Required:	Address: 770 King Edward Ave.
Maximum Subroutine Nesting Level:	City: Ottawa, Ontario
Assembler/Compiler Used: ISIS-II 8080/8085 Assembler, V.2.0	State: Canada K1N 6N5

Ref.#BC29

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

Discrete Fourier Transform

Function

DFT.SR is an 8080 Assembly language subroutine implementing the forward and inverse Fourier transform of a complex data vector. The subroutine executes a Fast Fourier Transform algorithm originally written in FORTRAN by Cooley, Lewis and Welch.

Required Hardware

8080 CPU

Required Software

Floating Point Packages references BC1, BC2 and BC14

Input Parameters

DFT: entry point for the forward transform,
IDFT: entry point for the inverse transform,
XR: real part of the input vector (size N array),
XI: imaginary part of the input vector (size N array),
N: number of data points in the input vector (must be a power of 2).

Output Results

XR: real part of the output vector (size N array)
XI: imaginary part of the output vector (size N array)
N: number of data points in the output vector

Registers Modified:	All	**Programmer:**	Louis Gilles Durand
RAM Required:	2052H	**Company:**	Institut de Recherches Cliniques de Montreal
ROM Required:	377H	**Address:**	110 ouest Avenue des Pins
Maximum Subroutine Nesting Level:	3 without nesting due to FPP	**City:**	Montreal, Quebec
Assembler/Compiler Used:	16K ISIS Assembler Version 1.1	**State:**	Canada H2W 1R7

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

FCONST -- Floating Point Constant Calculator

Function

The routine calculates and displays on the console 4 byte floating point constants equivalent to fixed-point or scientific notation numbers entered by the operator. These can then be used in PLM data statements or assembeler DB statements.

Required Hardware

MDS, Console, Floppy Disk System

Required Software

ISIS-II. Program is located above ISIS and is called by entering FCONST

Input Parameters

Fixed point or scientific notation numbers, entered on console
e.g. 1.09, 10.5E-20

Output Results

4 Bytes HEX constant displayed on console
e.g. F2, 04, 35, 3F

Data is displayed as it would be used in PLM or Assembler
e.g. DCL CONST (4) BYTE DATA (0F2H, 04H, 35H, 3FH);
or CONST: DB 0F 2H, 04H, 35H, 3FH

Registers Modified:	Programmer: J. B. Page
RAM Required: 3743 BYTES	Company: Plessey Marine
ROM Required:	Address: Templecombe
Maximum Subroutine Nesting Level:	City: Somerset
Assembler/Compiler Used: PLM80 V3.1	State: U.K.

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☐ 8080/8085 ☒ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

FPAL86.LIB - FLOATING POINT LIBRARY

Function

Provides the exact same operations as 8080/8085 Floating-Point arithmetic library for 8086 based software. The routines must be called from and linked to an 8086 assembly language or PL/M-86 language program.

Required Hardware

An MDS or Series-II for assembling, compiling, linking, locating 8086 based programs. Plus hardware necessary to execute 8086 code.

Required Software

MDS-311 8086 software support package.

Input Parameters

Same as 8080/8085 FPAL.

Output Results

Same as 8080/8085 FPAL.

$50.00 (object code only on diskette)

Registers Modified: All	Programmer:
RAM Required: Total FPAL.86LIB uses 6811 bytes	Company:
ROM Required:	Address:
Maximum Subroutine Nesting Level:	City:
Assembler/Compiler Used: ASM86, PLM86	State:

Ref. # BB52

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

BINGRY - BINARY TO GRAY CODE CONVERSION SUBROUTINE

Function

Converts binary number to Gray cyclic code. Number can be 16 bits long / unused bits = Ø /.

Algorithm: $G_i = B_i + B_{i+1}$, where i=Ø, 1,2,..,15

G_i...ith bit of Gray number

B_i...ith bit of binary number

Required Hardware

any 8080 Microcomputer

Required Software

None

Input Parameters

Binary number stored in regs. D, E
/EØ ... LSB, unused bits = Ø /

Output Results

Gray cyclic code in D,E

Registers Modified: A,D,E	Programmer: Nemec D.
RAM Required: None	Company: VSE
ROM Required: ØCH	Address: Pelhrimovska 339/9
Maximum Subroutine Nesting Level: Ø	City: 145 00 Praha 4, Michle
Assembler/Compiler Used: Macro Assembler V2.Ø	State: Czechoslovakia

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

"FLN" - NATURAL LOG OF INTEL FLOATING POINT NUMBER

Function

Takes natural LOG (LN(X)) of number in floating point record (FPR)

Required Hardware

MDS 800 with ISIS-II linking loader

Required Software

Intel "FPAL.LIB", Intel LINK & LOCATE, PL/M80 Compiler

Input Parameters

Pass pointer to 18-byte "FPR" in B,C registers
PL/M80 call: CALL FLN (.FPR)

Output Results

Places natural log of contents of "FPR" into the "FPR"

NOTES:

1) Has not been compiled as "REENTRANT"
2) May be added to "FPAL.LIB" if desired
3) Returns $-\infty$ for LN(Ø)
4) Does not set error/stat bits for "Ø" operand

REF.NOS. BC32A,B,C
ORDERED AS ONE
PROGRAM

Registers Modified:	ALL	Programmer:	Bart Evans
RAM Required:	Variables: 28 Stack:2	Company:	Durrum Instrument
ROM Required:	Code: 462	Address:	1228 Titan Way
Maximum Subroutine Nesting Level:	1	City:	Sunnyvale
Assembler/Compiler Used:	PLM80, V3.0	State:	CA 94086

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

"FSQR" - SQUARE ROOT OF INTEL FLOATING POINT NUMBER

Function

Extracts square root of number in floating point record (FPR)

Required Hardware

MDS 800 with ISIS-II linking loader

Required Software

Intel "FPAL.LIB", Intel "LINK" & "LOCATE", PL/M80 Compiler

Input Parameters

Pass pointer to 18-byte "FPR" in B,C registers
PLM80 CALL: CALL FSQR (.FPR)

Output Results

Places square root of absolute value of contents of "FPR" into the "FPQ"

NOTES:

1) Has not been compiled as "REENTRANT"
2) May be added to "FPAL.LIB" if desired
3) Does not set error/stat bits for negative operand

REF.NOS. BC32A,B,C
ORDERED AS ONE
PROGRAM

Registers Modified: ALL	Programmer: Bart Evans
RAM Required: VARIABLES: 13 STACK: 4	Company: Durrum Instrument
ROM Required: CODE: 330	Address: 1228 Titan Way
Maximum Subroutine Nesting Level: 1	City: Sunnyvale
Assembler/Compiler Used: PLM80, V3.0	State: CA 94086

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

"FEXP" - NATURAL EXPONENT OF INTEL FLOATING POINT NUMBER

Function

Takes natural exponential (e^K) of number in floating point record (FPR)

Required Hardware

MDS 800 with ISIS-II linking loader

Required Software

Intel "FPAL.LIB", Intel "LINK" & "LOCATE", PLM80 Compiler

Input Parameters

Pass pointer to 18-byte "FPR" in B,C registers

PLM80 CALL: CALL FEXP (.FPR)

Output Results

Places natural exponential of contents of "FPR" into the "FPR"

NOTES:

1) Has not been compiled as "REENTRANT"
2) May be added to :FPAL.LIB" if desired

REF.NOS. BC32A,B,C
ORDERED AS ONE
PROGRAM

Registers Modified: ALL	Programmer: Bart Evans
RAM Required: VARIABLES: 25 STACK: 2	Company: Durrum Instruments
ROM Required: CODE: 498	Address: 1228 Titan Way
Maximum Subroutine Nesting Level: 1	City: Sunnyvale
Assembler/Compiler Used: PLM80, V3.0	State: CA 94086

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

BINGRY - BINARY TO GRAY CODE CONVERSION SUBROUTINE

Function

Converts binary number to Gray cyclic code. Number can be 16 bits long / unused bits = Ø /.

Algorithm: $G_i = B_i + B_{1+1}$, where i=Ø, 1,2,...,15

G_i...ith bit of Gray number

B_i...ith bit of binary number

Required Hardware

Any 8080 Microcomputer

Required Software

None

Input Parameters

Binary number stored in regs. D, E
/EØ ... LSB, unused bits = Ø /

Output Results

Gray cyclic code in D, E

Registers Modified: A, D, E	Programmer: Nemec D.
RAM Required: None	Company: VSE
ROM Required: ØCH	Address: Pelhrimovska 339/9
Maximum Subroutine Nesting Level: Ø	City: 145 00 Praha 4, Michle
Assembler/Compiler Used: Macro Assembler V2.0	State: Czechoslovakia

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

MTL DIV - SUBROUTINES

Function

1. Multiplication of two 24 bit Binary numbers to give a 48 Bit result.
2. Integer Division of a 48 Bit Binary number by a 24 Bit Binary number.

Required Hardware

Any 8080 or 8085 CPU

Required Software

Input Parameters

See Listing

Output Results

See Listing

Registers Modified:	ALL	Programmer:	Ken Bartlett
RAM Required:	12_{10} Bytes	Company:	Acurex Corporation
ROM Required:	259_{10} Bytes	Address:	485 Clyde Avenue
Maximum Subroutine Nesting Level:	1	City:	Mountain View
Assembler/Compiler Used:	8080/8085 Macro Assembler V2.0	State:	California 94042

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☒ 8048 ☐ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

SQUARE ROOT FOR MCS-48

Function

This routine generates an 8 bits root of a 16 bits number.

Required Hardware

None

Required Software

None

Input Parameters

16 bits binary number in R2, R3.
MSBYTE in R2, LSBYTE in R3.

Output Results

8 bits root in R6

Registers Modified: ALL	Programmer: Susumu Urata
RAM Required: 4 bytes	Company: TOA Microcomputer Inc.
ROM Required: 96 bytes	Address: 5-61 Nipponbashi, Naniwa
Maximum Subroutine Nesting Level: 2	City: Osaka 556
Assembler/Compiler Used: ISIS-II MCS-48/UPI-41 Macro Assembler V2.0	State: Japan

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

FLOATING POINT LOAD AND STORE SUBROUTINES

Function

Six routines to assist in storing and loading floating point numbers as defined in the INSITE floating point packages (Ref. #BC5)

Required Hardware

None

Required Software

None

Input Parameters

See Program Listings for:

1. SHIFT - Shift contents of CDE into BHL registers.
2. EXCH - Exchange BHL and CDE registers.
3. STONE - Store one floating point number in a buffer.
4. STARY - Store a floating point in an array of numbers.
5. LDONE - Load one floating point number from a buffer.
6. LDARY - Load a floating point number from an array of numbers.

Output Results

Registers Modified: CDE, BHL	Programmer: F.M. Cady
RAM Required: 8 Bytes on stack	Company: Dept. of Elec. Engineering
ROM Required: 98H bytes	Address: University of Canterbury
Maximum Subroutine Nesting Level: 1	City: Christchurch 1
Assembler/Compiler Used: 8080/8085 Macro Assembler, V2.0	State: New Zealand

SECTION 6

CROSS PRODUCTS SOFTWARE

**** SPECIAL NOTE ****

ALL DISKETTES PROCESSED BY INSITE USERS LIBRARY ARE ISIS-II FORMATTED.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. C2

☐ 4004 ☐ 8008 ☒ 8080 ☐ 4040

(use additional sheets if necessary)

Program Title

CROSS ASSEMBLER ON PDP-11.

Function

PERFORMS SYMBOLIC ASSEMBLY FOR 8080 ASSEMBLY LANGUAGE PROGRAMS. PROGRAM RUNS ON ANY DEC PDP-11 WHICH SUPPORTS THE MACRO ASSEMBLER.

Required Hardware

DEC PDP-11 CAPABLE OF SUPPORTING THE MACRO ASSEMBLER AND 4K WORDS OF MEMORY FOR MACRO DEFINITIONS.

Required Software

PDP-11 MACRO ASSEMBLER, PLUS SUPPORT SYSTEM; E.G., DOS OR RSX.

Input Parameters

ASSEMBLY LANGUAGE PROGRAM FOR THE INTEL 8080.

Output Results

ASSEMBLED LISTING AND/OR PDP-11 FORMAT BINARY LOAD MODULE.

Registers Modified: N.A.	**Maximum Subroutine Nesting Level:** N.A.
RAM Required: N.A.	**Assembler/Compiler Used:** PDP-11 MACRO ASSEMBLER
ROM Required: N.A.	**Programmer:** JOHN ANDERSON & WILLIAM GALWAY
	Company: UNIVERSITY OF UTAH DEPARTMENT OF CHEMISTRY

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. C5

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

8008 Macro Definition Set

Function

Permits cross assembly of 8008 programs on a PDP-11.

Required Hardware

Any PDP-11 with 16K core and the Macro Assembler.

Required Software

Macro Assembler minimum, and RT-11 operating system optional (instructions assume RT-11).

Input Parameters

Program source using PDP-11 delimiters (colon after labels, semicolon preceding comments). Assembled program must be in absolute form. See listing for appropriate pseudo-ops required in program.

Output Results

Assembled program. Linking loader can be used to generate an absolute paper tape of the object code.

NOTE: Remove macro comments to make more memory available.

Registers Modified: N.A.	**Assembler/Compiler Used:**
RAM Required: N.A.	**Programmer:** Tom Seim
ROM Required: N.A.	**Company:** Battelle-Northwest
Maximum Subroutine Nesting Level: N.A.	**Address:** Box 999 Richland, Wash. 99352

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. C8

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Cross Assembler for the PDP-11

Function

The program accepts a source file from any device on the system which has been prepared by the user according to the syntactical rules outlined in the accompanying document, and converts the 8080 mnemonics into two output files—a hexadecimal file for loading into the microprocessor memory, and a listing file.

Required Hardware

Digital Equipment Corp. PDP 11 with RSTS BASIC-plus.
-could be modified to operate under any extended BASIC

Required Software

An editor for preparation of the source file

Input Parameters

ASCII source file

Output Results

Hexadecimal file—if punched on paper-tape, suitable for loading into the microprocessor memory via loader

Listing file

Registers Modified: N/A	**Assembler/Compiler Used:** RSTS BASIC-plus
RAM Required: N/A	**Programmer:** G.D. Young
ROM Required: N/A	**Company:** Aanderaa Instruments Ltd.
Maximum Subroutine Nesting Level: N/A	**Address:** 560 Alpha Street Victoria, B.C., Canada V8Z 1B2

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☒ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

Cross Assembler for Nova 1200

Function

To serve as a cross assembler enabling the user to assemble programs written for the 8008 or 8080 microcomputers, without using a time sharing service.

Required Hardware

The program is designed to run on a Data General Nova 1200 mini-computer with 16K of memory utilizing Data General Fortran and the SECOS operating system. However, little or no modifications would be needed to implement the assemblers on any 16 bit computer with Data General Fortran, disk capability, and at least 16K of memory.

Required Software

Simple to moderate modifications would be required to enable the 8008 or 8080 assemblers to run on most other 16 bit computers with 16K memory, disk, and Fortran capability.

Input Parameters

Input file name, Intermediate file name, listing options

Output Results

An assembled listing with machine code given in octal

A paper tape of the object code in HEX for use in loading programs

REVISED 8/8/77

Registers Modified:	Programmer: Paul Mennen
RAM Required:	Company: Sierra Research
ROM Required:	Address: 247 Cayuga Road
Maximum Subroutine Nesting Level:	City: Cheektowaga, New York
Assembler/Compiler Used:	State:

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

CROSS ASSEMBLER FOR NOVA 1220, IBM 360/40 and CDC 3300

Function

The CP1 is a one pass cross assembler that will assemble programs written using the Intel 8008 or 8080 Microprocessor Assembly Language Instruction Sets, any subset of the two, or any user-defined instruction set similar to the Intel Assembly Language Syntax.

Required Hardware

A principal objective in the development of the assembler was transporability. To achieve this, it is written in a subset of ANSI Standard Fortran that allows it to be complied on a wide variety of computers ranging from large general purpose systems to minicomputers. It has been tested on an IBM 360/40, CDC 3300, and a Nova 1220. Providing such generality required that avoidance of certain special features of fortran that are compiler dependent but would have increased execution speed. In addition to generality with respect to compiling, we allow complete freedom of character sets by reading in character code definitions each time that the assembler is executed.

Required Software

Input Parameters

Another main objective of this assembler was the ability to assemble object code for a variety of present and future microprocessors. In order to achieve this goal it was necessary to characterize certain types of instructions and allow for new types in the future.

Because the assembler is a one-pass assembler, the source program only has to be read once which offers another advantage when assembling on a minicomputer equipped with teletype 10 character/sec paper tape input. The assembler reads the opcodes at execution time in order to have the capacity of producing object code for more than one microprocessor. Of course, frequent users of any one instruction set can build a permanent data base with the character representations and opcodes available prior to execution.

Output Results

A search technique called a TRIE-TREE (1,2,3) is used to store the identifiers and symbols in the most efficient method possible. This search technique requires a minimum of space and it searches and inserts very quickly. It was important to have an efficient search method because the opcodes are read in at execution time. The object program is currently output in the Intel hexadecimal tape format.

Registers Modified:	Programmer: Bernard Evans, Ph.D.
RAM Required:	Company: Cal Poly
ROM Required:	Address: Computer Sciences Dept
Maximum Subroutine Nesting Level:	City: San Luis Obispo
Assembler/Compiler Used:	State: California 93407

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

NOVA Cross Assembler for INTEL 8080

Function

To assemble programs written in the INTEL 8080 language and output a listing and a paper tape of the assembly.

Required Hardware

DATA GENERAL NOVA minicomputer with 8K or larger memory and an SOS or better operating system; teletype and line printer.

Required Software

The following DATA GENERAL software:

1. SOS software drivers for the TTY, LPT or other input device.
2. Trigger produced by SYSGEN (091-000070-03).
3. RELOCATABLE MATH LIBRARY (099-000001-02).
4. OPTIONAL - SOS CASSETTE LIBRARY (099-000041).
   or SOS MAGNETIC TAPE LIBRARY (099-000042).

Input Parameters

This is a two pass assembler. The only inputs in addition to the source program are user responses to the questions put forth by the assembler with respect to input and output.

The operating procedure is described in the first several pages of the enclosed printout.

Output Results

A listing of the source is output to the line printer. A tape of the assembly results is output to the teletype paper tape punch. This tape may be in BINARY, ASCII OCTAL or BNPF.

NOTE: The test program can consist of any 8080 source program. However, complete verification would require use of a source that contains all of the 8080 instructions.

Registers Modified: NA	Programmer: Greg A. Head, D.834 (317/353-3243)
RAM Required: NA	Company: Naval Avionics Facility
ROM Required: NA	Address: 6000 E. 21st Street
Maximum Subroutine Nesting Level:	City: Indianapolis, Ind. 46218
Assembler/Compiler Used:	State:

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. C13

☐ 4004 ☒ 8008 ☐ 8080 ☐ 4040

(use additional sheets if necessary)

Program Title

Intel 8008 Cross Inverse Assembler for HP-2100

Function

This program accepts as input BNPF object code. From this object code, it produces a pseudo-assembler listing containing: memory location, NP object input and the mnemonic derived from the BNPF object code.

Required Hardware

Hewlett-Packard 2100 family computer, 4K, Teleprinter, Photoreader.

Required Software

HP Assembler for assembly.
Program is self-contained.

Input Parameters

BNPF object code (see attached sheet)

Output Results

Pseudo-Assembler output in the following format:
(page #)(space)(character #)(space)(NP constant)(space)(source mnemonic)

Registers Modified:	**Maximum Subroutine Nesting Level:**
RAM Required:	**Assembler/Compiler Used:** HP Assembler
ROM Required:	**Programmer:** Roger J. Walker
847 ORTNAC, Concord, Mass. 01742	**Company:** c/o Concord-Carlisle High School

PROGRAM DELETION

Please note that page 6-15 thru 6-20 have been deleted from your Insite Manual.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. C17

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

PLM 83 (PLM/80 PASS 3)

Function

Inter-lists PLM source lines with the resulting assembly language and hex code.

Required Hardware

None

Required Software

PLM 81 & PLM 82 (PLM/80 PASS 1&2)

Input Parameters

PLM/80 pass 1 listing, PLM/80 pass 2 listing & Hex file

Output Results

1) inter-listed file

2) Alphabetical symbol table

TEST METHOD

Sample input & output files for Intel Square Root Program attached.

Registers Modified: N.A.	**Assembler/Compiler Used:** Fortran (IBM/360, SIGMA/9, PDP/10)
RAM Required: N.A.	**Programmer:** B. Searle
ROM Required: N.A.	**Company:** Ministry of Transport (CANADA)
Maximum Subroutine Nesting Level: N.A.	**Address:** TACD, Tower C, Place de Ville OTTAWA, ONTARIO, CANADA K1A ON8

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. C18

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

CROSS ASSEMBLER FOR VARIAN DATA MACHINES

Function

TRANSLATES SYMBOLIC 8080 CODE BY USING MACROS FOR EACH INSTRUCTION

Required Hardware

V73, V74 OR V75 VARIAN DATA MACHINE

Required Software

DASMR VARIAN ASSEMBLER LANGUAGE PROCESSOR
VORTEX VARIAN OMNITASK REAL-TIME EXECUTIVE

Input Parameters

SYMBOLIC CODE FOR 8080 CPU

Output Results

CODE IN VARIAN OBJECT MODULE FORMAT, ONE WORD OF 16 BITS IS RESERVED FOR ONE BYTE.

ASSEMBLER LISTING

Registers Modified: /	**Assembler/Compiler Used:** DASMR
RAM Required: /	**Programmer:** WALTER HEIL
ROM Required: /	**Company:** GESELLSCHAFT FÜR KERNFORSCHUNG/ADI
Maximum Subroutine Nesting Level: /	**Address:** 7500 KARLSRUHE WEST GERMANY P.B. 3640

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. C19

☐ 4004 ☐ 4040 ☒ 8008 ☒ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

Intel to Octal Number Conversion (OCT)

Function

This Fortran IV program converts source code containing Intel numbers (binary, octal, decimal & hexadecimal) into source code containing only octal numbers. The program is designed to run on a PDP-11 & can produce a source code which can be assembled by the PDP-11 Macro Assembler.

Required Hardware

PDP-11 computer with disk

Required Software

RT-11
Fortran IV compiler
System Library Subroutines

Input Parameters

INPUT FILE* (enter name of file with Intel numbers)
OUTPUT FILE*(enter name of file to contain octal number source code)

Output Results

An output file will be created which will contain the octal number source code

Registers Modified:	**Assembler/Compiler Used:** PDP-11 Fortran IV
RAM Required:	**Programmer:** Jim Ahern
ROM Required:	**Company:** Standard Memories
Maximum Subroutine Nesting Level:	**Address:** 2221 So. Anne St. Santa Ana, Ca. 92704

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. C20

☐ 4004 ☒ 4040 ☒ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Hexcnvrt (Hex Convert)

Function

Convert the Hexadecimal object file produced by the Intel cross product assemblers to the Hexadecimal format required by the Prolog Corp. Model 810 and series 90 Prom Programmers.

Required Hardware

Time sharing system or other computer system with capability of running Intel cross product software.

Required Software

Fortran IV
(Source program may have to be modified for other compilers).

Input Parameters

"LOGBIN" File produced by Intel Macro assembler (see attached sheets).

Output Results

"Prolog" File (see attached sheets).

Registers Modified: ----------	**Assembler/Compiler Used:** GE Mark III Timesharing
RAM Required: ----------	**Programmer:** Joseph Parchesky
ROM Required: ----------	**Company:** Datatype Corporation
Maximum Subroutine Nesting Level: ----------	**Address:** Miami, Florida 33169

98-0348

REF. NO. C20

PROGRAM TITLE HEXCONVERT

```
100*PROGRAM TO CONVERT THE INTEL HEXADECIMAL
110*OBJECT FILE (LOGBIN) FORMAT TO A HEXADECIMAL
120*FORMAT SUITABLE FOR INPUT TO THE PROLOG
130*MODEL 810 PROGRAMMER - PROLOG REQUIRES A 4
140*CHARACTER CODE FOLLOWED BY A CARRIAGE RETURN
150*THE INTEL MAC4 ASSEMBLER PROVIDES A BLOCKED
160*FORMAT AS DESCRIBED IN THE EXTERNAL REFERENCE
170*SPECIFICATION 4004/4040 MACRO ASSEMBLER NOV74
180* (ALSO NOTE SPACE BEFORE COLON)
190*
200*INPUT FILE IS LOGBIN
210*OUTPUT FILE IS PROLOG (WHICH USER MUST CREATE)
220*
230*ON GE MARKIII TIME SHARING, WHEN LISTING FILE
240*TO TELETYPE PUNCH, USE "TYPE 6" COMMAND TO
250*INHIBIT DELETE CODES ON TAPE AND KEEP IT AS
260*SHORT AS POSSIBLE.
270*
280*
290*INITIALIZE THE HEX VARIABLE
300         BLOCK DATA
310         ALPHA HEXA(16)
320         COMMON/LABEL/HEXA
330         DATA HEXA/"0","1","2","3","4","5","6","7","8","9","A","B",
340        &"C","D","E","F"/
350         END
360*
370*MAIN PROGRAM
380         ALPHA CHAR(24),PAGE,A(16),B(16),C
390         DATA C/0015000000000/
400         PAGE="  "
410*INPUT LINE FROM LOGBIN FILE
420    420  READ("LOGBIN",430)CHAR
430    430  FORMAT(A2,2A1,A2,2A1,18A2)
440*CHECK FOR END OF FILE
450         IF(CHAR(1).NE." :")GOTO 690
460         IF(CHAR(2).NE."0")GOTO 490
470         IF(CHAR(3).EQ."0")GOTO 690
480*CHECK FOR CHANGE IN MEMORY PAGE
490    490  IF(CHAR(4).EQ.PAGE)GOTO 540
500         PAGE=CHAR(4)
510         WRITE("PROLOG",520)PAGE
520    520  FORMAT("/",//,"PROM PAGE ",A2,//)
530*J1 IS NUMBER OF BYTES IN LINE (DECIMAL)
540    540  CALL CNVRTHEX(CHAR(2),CHAR(3),J1)
550*J2 IS STARTING ADDRESS OF LINE (DECIMAL)
560         CALL CNVRTHEX(CHAR(5),CHAR(6),J2)
```

```
570*WRITE LINE TO OUTPUT FILE
580    580 DO 630 I=1,J1
590        J3=J2+I-1
600*J3 IS BYTE ADDRESS (DECIMAL) AND IS CONVERTED
610*BACK TO HEX
620        CALL CNVRTDEC(J3,A(I),B(I))
630    630 CONTINUE
640        WRITE("PROLOG",650)(A(I),B(I),CHAR(I+7),C,I=1,J1)
650    650 FORMAT(16(A1,A1,A2,A1))
660*REPEAT NEXT LINE
670    670 GOTO 420
680*END OF FILE
690    690 WRITE("PROLOG",700)
700    700 FORMAT("/",//,"EOF",//)
710        ENDFILE "PROLOG"
720        STOP
730        END
740*
750*SUBROUTINE TO CONVERT HEX CHARACTERS TO DEC
760        SUBROUTINE CNVRTHEX(G,H,K)
770        ALPHA G,H,HEXA(16)
780        COMMON/LABEL/HEXA
790        INTEGER IHEX(16)
800        DATA IHEX/0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15/
810        DO 860 I=1,16
820        IF(G.NE.HEXA(I))GOTO 860
830        DO 850 J=1,16
840        IF(H.EQ.HEXA(J))GOTO 870
850    850 CONTINUE
860    860 CONTINUE
870    870 K=IHEX(I)*16+IHEX(J)
880        RETURN
890        END
900*
910*SUBROUTINE TO CONVERT DECIMAL NUMBER TO HEX
920        SUBROUTINE CNVRTDEC(J,X,Y)
930        COMMON/LABEL/HEXA(16)
940        ALPHA X,Y,HEXA
950    950 DO 970 N=1,17
960        IF((J+16)-(N*16))1010,980,970
970    970 CONTINUE
980    980 X=HEXA(N)
990        Y=HEXA(1)
1000        RETURN
1010   1010 X=HEXA(N-1)
1020        Y=HEXA((J+17)-(N-1)*16)
1030        RETURN
1040        END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. C21

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

PDP-11 binary file to INTEL HEX file converter.

Function

Using PDP-11 MACRO to assemble 8080 programs leaves the user with a PDP-11 binary file containing the object code. This program converts the absolute binary file to an INTEL HEX file suitable for input to either INTERP/80 or an INTELLEC system.

Required Hardware

PDP-11 with an operating system
disk, papre tape reader/punch

Required Software

a set of macro definitions defining the 8080 instruction set
(X8-2) is a possible set
PDP-11 FORTRAN and MACRO

Input Parameters

PDP-11 absolute binary tape (program could be modified to read from another source)
output file name
number of bytes per line in the output file

Output Results

INTEL HEX file for use with INTELLEC or INTERP/80

Registers Modified: -------	**Assembler/Compiler Used:** PDP-11 FORTRAN and MACRO
RAM Required: -------	**Programmer:** Rex Tracy
ROM Required: -------	**Company:** Colorado State University Electrical Engr. Dept/
Maximum Subroutine Nesting Level: -------	**Address:** Ft.Collins, Co. 80523

98-034B

Ref. # C22

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

PDP-11 Program to load Intel Hex programs and dump and verify the programs in binary format suitable for PROM programming.

Function

The program has three parts. LOADHX reads the Hex format programs into the PDP-11 core store. DMPBIN punches out in PROM binary format parts of the program in the PDP-11 core store. VERIFY reads and compares the tape produced by DMPBIN with the core store contents.

Required Hardware

DEC PDP-11 computer with 8K store and a high speed reader and punch. (Could be used with ASR 33 Teletype only with minor modifications)

Required Software

Only requires Absolute Loader. The program occupies absolute core locations 200-676 and uses the rest of the store as a buffer area to load the Intel Hex programs.

Input Parameters

LOADHX With Intel Hex tape in reader

SR	Press	Comment
000200	ADDRESS LOAD	Starting address
XXXXXX	START	Buffer offset in SR

DMPBIN Punch out binary tape

SR	Press	Comment
000500	ADDRESS LOAD	Starting address
XXXXXX	START	Buffer offset in SR
	When the computer halts	
YYYYYY	CONT	No. of bytes to be punched

VERIFY With binary tape in the reader

SR	Press	Comment
000600	ADDRESS LOAD	Starting address
XXXXXX	START	Buffer offset in SR
	When computer halts	
YYYYYY	CONT	No. of bytes to be verified

Output Results

Halts at the following locations are faults:-

LOCATION	REASON
246	Checksum error in reading Intel Hex code
652	Verification error
446	Reader error, taut tape or no tape
472	Punch error, punch no ON or out of tape

Registers Modified:	Programmer: A. W. J. Griffin
RAM Required:	Company: Simulation Systems
ROM Required:	Address: 18 Park Avenue
Maximum Subroutine Nesting Level:	City: Paisley
Assembler/Compiler Used: MAC 80 to generate Intel Hex code; PAL11A to produce LDHEX/DMPBIN/VERIFY	State: Strathclyde PA2 6HL U.K.

Ref. # C23

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other _____ (use additional sheets if necessary)

Program Title

FORMAT

Function

Converts Varian object format produced by DASMR for the INTEL 8080 CPU to HEX file format. The program is used in conjunction with the set of macros, which define the 8080 instructions for the Varian assembler.
(see contribution C18 in this library)

Required Hardware

V73, V74, or V75 Varian Data Machine

Required Software

DASMR Varian Assembler Language Processor
VORTEX Varian Omnitask Realtime Executive

Input Parameters

Inputfile: binary output of DASMR

Output Results

INTEL HEX file of objectcode
on LP and/or PTP for use with INTELLEC

Registers Modified: --	Programmer: R. G. Knoepker
RAM Required: --	Company: Gesellschaft für Kernforschung/ IDT
ROM Required: --	Address: P.O.B. 3640
Maximum Subroutine Nesting Level: --	City: 7500 KARLSRUHE
Assembler/Compiler Used: DASMR	State: West Germany

Ref. # C24

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

8080 CROSS-ASSEMBLER FOR TEKTRONIX 4051

Function

This program assembler Intel 8080 source code producing a listing and (optionally) an object file.

Required Hardware

Tektronix 4051 Graphic System with 16K or larger memory.

Tektronix 4924 tape unit (optional)

Required Software

Input Parameters

See Attachment

Output Results

Source Listing available only. Program not available on Paper Tape or Diskette.

Registers Modified:	Programmer: Bruce W. Bomar
RAM Required:	Company: ARO, Inc.
ROM Required:	Address: Arnold AF Station
Maximum Subroutine Nesting Level:	City:
Assembler/Compiler Used: Extended BASIC	State: TN 37389

8080 CROSS-ASSEMBLER FOR TEKTRONIX 4051

HARDWARE REQUIREMENTS:

A Tektronix 4051 Graphic System with 16K or larger memory. A Tektronix 4924 tape unit (optional) may be included to save the hexadecimal output code from the assembler. An extended function editor ROM (Number 4051R06) (optional) may be included to edit the assembly language program.

DESCRIPTION:

This cross-assembler accepts Intel 8080 microprocessor assembly language instructions (as described in the Intel 8080 Microcomputer Systems User's Manual) and provides an output listing with line number, address, and hexadecimal code. With an optional 4924 tape unit, the hexadecimal code may be saved on magnetic tape for listing in table format at the end of program assembly and for entry into an 8080 based microcomputer.

Input to the assembler is from the 4051 internal magnetic tape unit. Output is to the 4051 CRT diaplay. (Hexadecimal code is also output to a magnetic tape file in systems using a 4924).

The assembly language program must be placed on a file, F, of magnetic tape within the 4051 internal tape unit prior to using the assembler, This can be accomplished using either the optional ROM editor or the following 4051 program:

```
100 FIND F
110 INPUT X$
120 PRINT @33:X$
130 GO TO 110
140 END
```

This program is initiated by typing RUN and is terminated by pressing the "break" key after the program has been typed in. Note that the program file, F, must be marked large enough to hold the assembly language program (approximately 72 times the number of program lines = number of bytes required) prior to running the above program.

The format for the assembly language program must be as follows:

Columns 1 - 6	Symbol followed by ":"
Column 7	Space
Columns 8 - 11	Instruction mnemonic
Column 12	Space
Columns 13 -	Operand

An entire line of the assembly language program may be a comment if a ";" is placed in column 1. The portion of a program line after the operand may also represent a comment by following the operand with one or more spaces and a ";" followed by the comment.

Valid entries in the symbol field must begin with a letter of the alphabet and may contain alphanumeric characters but must not contain special characters (such as punctuation characters or spaces and tabs). Each symbol must be five characters or less in length followed by a ":". The assembler is dimensioned to permit the use of as many as 150 symbols on a 16K machine. On larger memory machines this number can be increased by adding 6 to the dimension of S$ and 1 to the dimenion on S for each additional symbol table entry permitted. The value for comparison against K in line 285 of the assembler should be set to the maximum permissible number of symbol table entries.

Valid entries in the instruction mnemonic field are elements of the 8Ø8Ø instruction set and the following assembler-control mnemonics:

ORG - Set the address counter to the operand field value.

EQU - Equate a symbol to the operand field value.

DW - Define word (two bytes) equal to the operand field value.

DB - Define byte (8 bits) equal to the operand field value.

DS - Define block storage for the number of bytes specified by the operand field.

DC - Define consecutive ASCII character bytes for the characters (up to 31) specified in the operand field.

END - End assembly.

Valid entries in the operand field are:

1) "#" followed by a hexadecimal string.

2) A positive or negative decimal number (fractional values are truncated to integer value--negative values are converted to two's complement hexadecimal code).

3) "'"followed by ASCII character(s) and closed by "'"

4) Register identifiers A, B, C, D, E, H, L, M, SP, or PSW or two of these separated by a comma. An error message is given if register identifier(s) which are not appropriate for a given operation are specified.

5) A valid symbol. The symbol must be defined elsewhere in the program.

6) "$" (which represents the current value of the address counter).

A single level of arithmetic expression is allowed in the operand field on quantities involving symbols and "$". For example "BUFFER + 18", "$-1Ø", and "$-BUFFER" would all be permissible operands as long as the symbol "BUFFER" is defined elsewhere in the program.

Errors encountered during assembly are flagged on pass 1 by printing the line number of the error, an error mnemonic, and the error-containing line. On pass 2, the error mnemonic is printed in the hexadecimal code output field. Error mnemonics and their meaning are:

*D - Double defined symbol.

*I - Unrecognizable mnemonic.

*N - Invalid numeric or character string operand.

*O - Operand out of range (too large or small) for the instruction being processed.

*R - Unrecognizable or illegal register identifier in the operand field.

*U - Undefined or illegal symbol.

OPERATION:

Before using the assembler for the first time, the following preparations should be made. First, if a 4924 tape unit is used, the address of this unit must be set to 2. If a 4924 is not used the following changes must be made in the assembler:

Delete lines 1Ø5, 155, 16Ø, 445, 545, 55Ø, 565, 57Ø, 6ØØ, 99Ø through 1165, and 1175 through 1185.

Change line 535 to -

535 IF B = 2 THEN 555.

Second, place the assembler (modified as indicated above if a 4924 is not used) on magnetic tape for future access.

To use the assembler, first locate the file on magnetic tape where it is stored by executing a FIND statement. Then load the assembler into memory with an OLD statement. If a 4924 is used, mark a tape file large enough to hold the expected hexadecimal code output and place the tape containing this file into the 4924. Place the tape containing the assembly language program into the 4Ø51. Type RUN and enter the file numbers of the program input and, if used, hexadecimal output files as requested.

PROGRAM DOCUMENTATION:

To conserve memory in the 4Ø51 for symbol table use, few remark statements were included to document the assembler. This section will describe the different functions of the assembler program.

First, a table of program variables and constants along with their function is given. This table is followed by a description of operations performed by each major division of the assembler program.

PROGRAM VARIABLES & CONSTANTS:

A$ = Current operand field.

C$ = Current instruction's hexadecimal code.

L$ = Current input line.

M$ = Current mnemonic field.

O$ = Current output line.

R$ - Register identification table.

S$ = Symbol table.

T$ = Mnemonic identification table.

U$ = Current symbol field.

A = Current operand field length.

E = Error counter.

F = Program input file.

F1 = Hexadecimal output file.

K = Current symbol index.

L = Current line number.

M = Current mnemonic instruction code index.

N = Number of bytes in current instruction.

P = Current address.

R = Current register code.

S = Symbol address array.

T = Instruction code array.

U = Current symbol length.

V = Current operand value.

Lines 1ØØ through 175 request input and output files, dimension variables, and set up register and mnemonic lookup tables.

Pass 1 (lines 18Ø through 42Ø) constructs and prints the symbol table.

Pass 2 (lines 425 through 985) determines the hexadecimal code required to execute each program instruction, prints and numbers each line giving current address and hexadecimal codes, and stores the code on magnetic tape in systems where a 4924 is used.

Lines 99Ø through 118Ø print a hexadecimal code table by address in systems where a 4924 is used.

The subroutine beginning at line 12Ø5 inputs a program line and separates it into symbol, mnemonic, operand and comment fields. It then determines the type of instruction contained in the line along with the number of bytes, N, of memory required for the instruction. On return from this subroutine, the value of C indicates the type of instruction as follows:

C = 1	error	C = 5	DC
C = 2	8Ø8Ø instruction,	C = 6	DS
	M determines the type	C = 7	ORG
	of instruction in order	C = 8	EQU
	of its occurrence within	C = 9	END
	data statements which	C = 1Ø	Comment
	start at line 223Ø.		
C = 3	DW		
C = 4	DB		

The subroutine beginning at line 159Ø evaluates the operand field and returns its value in V. On return, if C = 1 an undefined symbol was encountered. If C = 2, an invalid argument was encountered. A value of C = 3 corresponds to a normal return from the subroutine.

The subroutine beginning at line 191Ø converts a hexadecimal string, A$, to a decimal number, V. On return, if C = 1 an invalid string was encountered. A value of C = 2 corresponds to a normal return.

The subroutine beginning at line 1995 converts a decimal number, G, to a hexadecimal string, N$. On return, B is the number of 8-bit bytes contained in N$.

The subroutine beginning at line 21Ø5 returns a value of Ø through 7 for R, corresponding to the binary equivalent of the current operand's register identifier. On return, C = 1 indicates that an invalid or illegal register identifier was encountered. A value of C = 2 corresponds to a normal return from the subroutine.

Research reported in this paper was conducted by the Arnold Engineering Development Center, Air Force Systems Command. Research results were obtained by personnel of ARO, Inc., contract operator of AEDC. Further reproduction is authorized to satisfy needs of the U. S. Government.

Ref. #C25

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

18080 Cross Assembler for INTEL 8080/8085 Microprocessors.

Function

The Cross Assembler is written in HP ALGOL for execution on Hewlett-Packard 2100 series computers. The program is designed for the RTE-2 and RTE-3 operating systems. The Assembler accepts 8080/8085 source language statements (very close to INTEL Assembly) and generates a listing (including a symbol table) and a hexadecimal file format.

Required Hardware

Computer: Hewlett-Packard 2100 series.
Memory size: 32K-words.
Operating system: RTE-2/RTE-3

Required Software

HP-ALGOL For the Hewlett-Packard 2000 computer series.

Input Parameters

Source language program stored on a disc file.

Output Results

1. Listing of source code.
2. Symbol table.
3. Listing of generated object code.
4. Possible error messages.
5. Hexadecimal disc file.

Registers Modified:	Programmer: Lara Thrane
RAM Required:	Company: Electromagnetics Institute
ROM Required:	Address: Technical University,Bldg 348
Maximum Subroutine Nesting Level:	City: DK-2800 Lyngby
Assembler/Compiler Used:	State: Denmark

Ref.# C26

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

MACRO ASSEMBLER FOR DG NOVA

Function

Assembles standard 8080 instruction set plus allows a full macro facility. Consists of a number of user definitions and macros which define the 8080 instruction set which are passed to the NOVA macro assembler.

Required Hardware

DG NOVA or ECLIPSE with at least 16K and a disk operating system.

Required Software

RDOS or equivalent operating system able to run the NOVA macro assembler.

Input Parameters

Source code in NOVA assembly format using 8080 mnemonics and register definitions.

Output Results

Listing and absolute binary (in a .RB format) with the 8080 code assembled in the right side of each NOVA word. Note; due to the address structure of the NOVA, code above adr 077777 will not be assembled ... and address error will be flagged.

Registers Modified:	Programmer: Tom Rust
RAM Required:	Company: Computer Tools
ROM Required:	Address: 205 W. Eureka
Maximum Subroutine Nesting Level:	City: Champaign
Assembler/Compiler Used: DG NOVA MACRO ASSEMBLER	State: Illinois 61820

CROSS-ASSEMBLER FOR THE 8080 ON A DG NOVA

THIS PROGRAM CONSISTS OF A SET OF USER DEFINITIONS (.DUSR) AND MACRO DEFINITIONS THAT DEFINE THE 8080 INSTRUCTION SET. A NUMBER OF ADDITIONAL PSUEDO-OPERATIONS HAVE ALSO BEEN DEFINED FOR COMMONLY USED FUNCTIONS. THESE INCLUDE:

.WORD -ALLOCATES 2 BYTES OF STORAGE FOR EACH OPERAND

.BYTE -ALLOCATES 1 BYTE

FROM 1-64 OPERANDS CAN BE SPECIFIED IN EACH OF THE ABOVE PSUEDO-OPS, CAUSING EACH OPERAND TO BE ASSEMBLED INTO SEQUENTIAL MEMORY LOCATIONS.

.Z -GENERATES THE HI AND LO BYTES FOR JUST ONE OPERAND

.PAGE -BUMPS UP THE ASSEMBLY COUNTER TO THE NEXT PAGE OF MEMORY

CLA -CLEAR REG A AND CARRY..GENERATES AN XRA A INSTRUCTION

NOTE THE NOVA INSTRUCTION SET HAS BEEN .XPNG(ED) FROM THE ASSEMBLER.. IT IS POSSIBLE TO ASSEMBLE NOVA CODE ALSO IN-LINE, THOUGH THERE ARE SOME CONFLICTS BETWEEN INSTRUCTIONS.

THIS ASSEMBLER HAS ONE STRONG DISADVANTAGE IN THAT IT IS DIFFICULT TO ASSEMBLE CODE ABOVE ADDRESS 077777 DUE TO THE FACT THAT THE ADDRESS STRUCTURE OF THE NOVA ONLY ALLOWS UP TO 32K WORDS OF MEMORY TO BE ACCESSED. HOWEVER, MANY PROGRAMS ARE SMALL ENOUGH TO FIT IN THE LOWER HALF OF THE 8080 ADDRESS SPACE, AND THERE ARE WAYS OF BYPASSING THIS PROBLEM BY ASSIGNING LABELS RATHER THAN USING THE : SUFFIX. A RELOCATABLE LOADER HAS BEEN WRITTEN THAT ALLOWS MODULER CODE TO BE GENERATED AND SEPERATELY LINKED TOGETHER. THIS LOADER, WHEN USING RELOCATABLE CODE SEGMENTS, CAN PLACE THE SEGMENTS ANYWHERE IN THE FULL 64K OF THE 8080 MEMORY SPACE.

ALL THE CODE IS ASSEMBLED INTO THE RIGHT BYTE OF THE NOVA WORD, AS THERE ARE NO BYTE LISTING FORMATS. THIS ALSO WASTES THE LEFT BYTE GENERATED IN THE BINARY (.RB), BUT IS UNAVOIDABLE WITHOUT REWRITING THE MACRO ASSEMBLER ITSELF.

THE ASSEMBLER SUPPORTS A WIDE VARIETY OF PSUEDO-OPS AND A VERY POWERFUL RECURSIVE MACRO STRUCTURE. CONDITIONAL ASSEMBLY STATEMENTS, CONDITIONAL LOOP STATEMENTS, ASSEMBLY BRANCH STATEMENTS, AND A FULL SET OF OPERATORS INCLUDING +,-,*,/,!,(OR), & (AND), BINARY SHIFT,

PARENS. FULL CONDITIONAL OPERATORS--

>	GREATER THAN
<	LESS THAN
> =	GREATER THAN OR EQUAL TO
< =	LESS THAN OR EQUAL TO
==	EQUAL TO
<>	NOT EQUAL TO

SPECIAL RADIX PSUEDO OPS ALLOW ANY RADIX FROM 2-20 TO BE USED AS DEFAULT ON INPUT (AS WELL AS LOCALLY REDEFINED), AND ANY RADIX FROM 8-20 (ALLOWING HEX LISTINGS AS WELL AS OCTAL) ON OUTPUT.

TO USE THE ASSEMBLER, THE DEFINITIONS ARE READ INTO A FILE WHICH WILL THEN NEED ONLY BE READ ON THE FIRST PASS OF THE ASSEMBLY. AS AN EXAMPLE, ASSUME THE DEFINITIONS ARE IN FILE 80DEF. THE SOURCE TO BE ASSEMBLED IS IN FILE TEST AND THE OUTPUT IS TO GO TO FILE TEST.LS. THE COMMAND LINE INTERPRETER INPUT TO BE TYPED WOULD BE:

```
.MAC 80DEF/S TEST TEST.LS/L
```

THE /S INDICATES THE DEFINITIONS NEED ONLY BE READ IN ON THE FIRST PASS. USING THE /S SPEEDS THE ASSEMBLY CONSIDERABLY AND REMOVES THE DEFINITIONS FROM THE ASSEMBLY LISTING.

*****CAUTION ! *****

THE FOLLOWING SYMBOLS SHOULD NOT BE REDEFINED BY THE USER SOURCE:

I	USED IN ALMOST EVERY INSTRUCTION
L	USED IN .WORD

PLUS OF COURSE THE MNEUMONICS THEMSELVES AND THE REGISTER NAMES. HOWEVER, IF ANY OF THE REGISTERS ARE REDEFINED AN ERROR MESSAGE WILL FLAG THE BAD CODE. THE ABOVE 2, IF CHANGED IN THE CODE STREAM, MAY CAUSE UNPREDICTABLE RESULTS.
ALSO INCLUDED IS A TEST LIST OF ALL THE MNEUMONICS FOR INITIAL PROGRAM CHECKOUT.

ANY INQUIRIES SHOULD BE DIRECTED TO:

TOM RUST
COMPUTER TOOLS
205 W. EUREKA
CHAMPAIGN, IL 61820
217-359-5534

Ref. #C27

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

SYMBOL CROSS-REFERENCE

Function

Produce a cross-reference listing of tag names vs referenced locations for 8080 assembly language programs on a PDP-11. Uses the symbol table output of the Intel paper tape assembler.

Required Hardware

PDP-11 with a disk
CRT
Paper Tape Reader
Line Printer

Required Software

No additional software required

Input Parameters

1. The symbol table - hex location as output on paper tape by the mds paper tape assembler.
2. The user program source tapes.

Output Results

1. The cross-reference listing on the line printer
2. Coding errors listed on the CRT

Registers Modified: N/A	Programmer: Eugene Bidwell
RAM Required: N/A	Company: Supreme Eqpt. & Systems
ROM Required: N/A	Address: 170 53rd Street
Maximum Subroutine Nesting Level: None	City: Brooklyn
Assembler/Compiler Used: PDP-11 Assembler	State: New York 11232

1. Disk Data Base Format

The program is assembled to save the SYMBOL, its location and cross-references on the disk starting at disk address 3000 octal.

Each symbol is allocated 60 words on the disk in the following sequence:

Byte 0 - 4 The symbol name in ASCII
" 5 The number of references plus one
" 6 - 9 The symbol address in ASCII
" 10 - 11 1st reference location in binary
" 12 - 13 2nd reference location in binary
" 126,127,58th " " " "

As can be seen, there is a maximum of 58 references per symbol which will be listed.

2. Core memory data base format - for the SYMBOL vs. disk location of its data.

A map of the disk data is maintained in core at the address labeled (SYM).

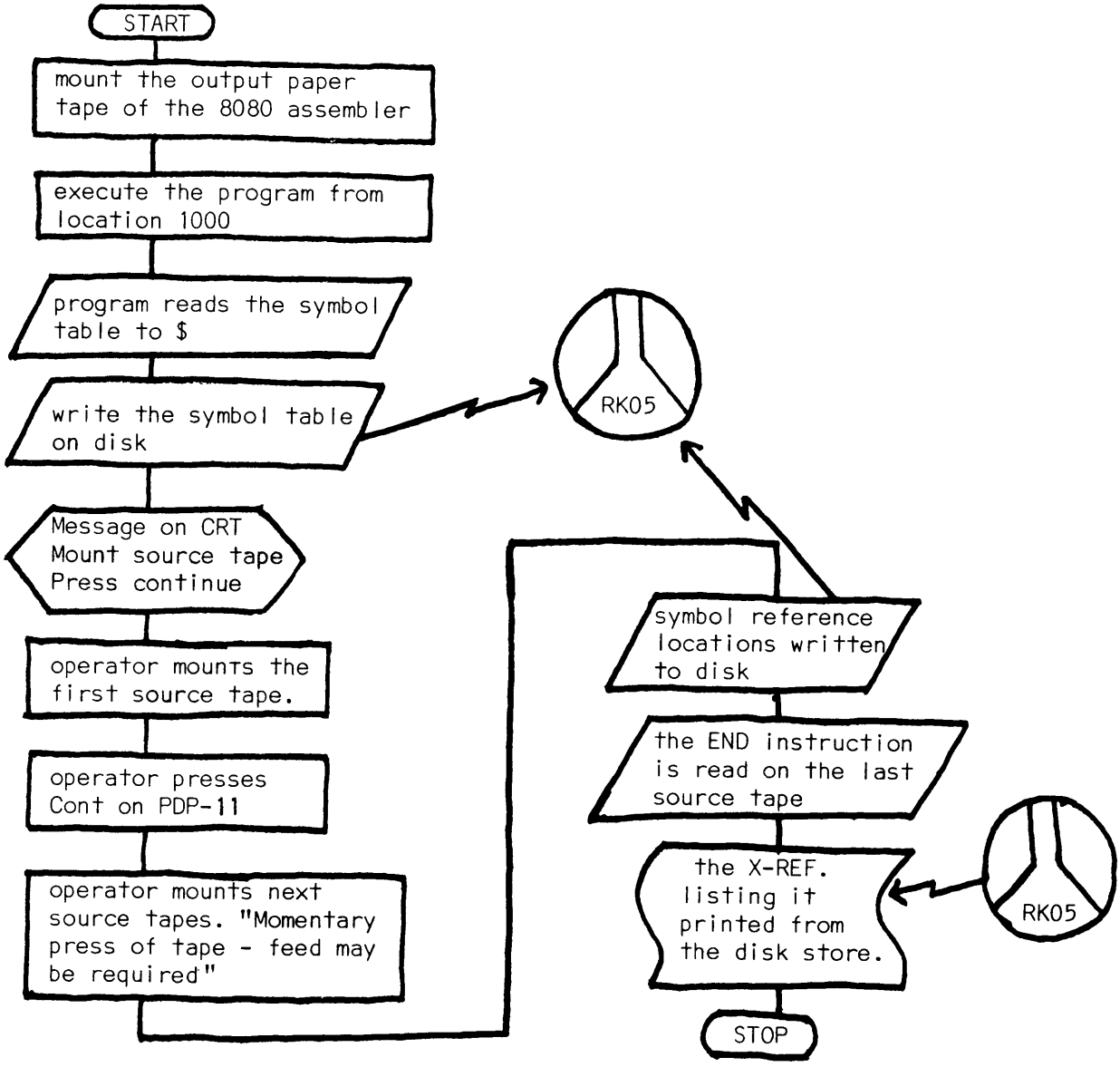
A disk sector has 256 words. Each symbol is given 60 words, as mentioned on the previous page. Thus the data for 256/60 = 4 symbols is kept on one sector. These 4 records are distinguished by an index (0 - 3) in the disk memory map.

The disk memory map has 8 bytes per symbol as follows:

Byte	0 - 4	The symbol name in ASCII
"	5	The sector index (0 - 3)
"	6 - 7	The disk address

The disk memory map is in sort on symbol name.

BASIC OPERATION



Ref. #C28

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

Align Program - Intermediate Pass between PLM Pass 1 and 2.

Function

Generate consistent storage assignment for global variables in separately compiled program segments.

Required Hardware

Computer that supports the PLM Cross Compiler and FORTRAN IV

Required Software

PLM Cross Compiler, FORTRAN Compiler

Input Parameters

Origin (within RAM bank specified by $V in Pass 2) via FORTRAN Unit 9 (decimal, free format)

PLM intermediate File 21 as generated by Pass 1, via FORTRAN Unit 23.

Output Results

Message Output via FORTRAN Unit 10 (Conversational Terminal Output File)

PLM Intermediate File 21 as reprocessed, via FORTRAN Unit 21.

NOTE: The function ICON included here is an Arithmetic Statement Function specific to IBM System/360/370. It is functionally equivalent to the Function ICON furnished with the PLM Cross Compiler.

Program available in source listing only.

Registers Modified: NA	Programmer: R. L. Mahn & O. C. Juelich
RAM Required: NA	Company: Missile Systems Division Rockwell International Corp.
ROM Required: NA	Address: 4300 E. Fifth Avenue
Maximum Subroutine Nesting Level: NA	City: Columbus
Assembler/Compiler Used: IBM 370·Fortran IV	State: Ohio 43216

The ALIGN program is an intermediate pass between passes One and Two of the PLM Cross Compiler. It allows separate compilation of program segments arising from compiler table limitations, from memory overlay design, or for any other reason.

The variables global to the segments must be declared at the start of the main program block before declarations of procedures. Address variables should be declared ahead of Byte variables.

Since the PLM Cross Compiler allocates compiler temporaries in the space preceding the global variables in the RAM bank specified by $V in Pass Two, an origin or offset for the global variables must be specified by the user. The ALIGN program adds a dummy declaration to the intermediate file 21 to bring the beginning of the global variables down to the specified origin or offset, which should normally be an even number. The ALIGN program may be left in place between Passes One and Two of the PLM Cross Compiler even when its services are not needed. An offset specification of -2 will direct that no dummy declaration be added to the intermediate file 21.

When the ALIGN program is in place, completion of Pass One is followed by the message:

ALIGN PROGRAM - ENTER OFFSET

The user responds by typing the desired offset as a decimal number, e.g., 128, -2. The ALIGN program is a normal FORTRAN program, executed between Passes One and Two of the PLM Cross Compiler. The File 21 output of Pass One is read from unit 23. The change in unit numbers can be effected via job control in many host systems. Alternatively, the reference to unit 21 in Pass One subroutine WRITEL could be changed to refer to unit 23. No change is needed in Pass Two.

Ref. #C29

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

8085 Cross Assembler for the DEC PDP8 and PDPII

Function

Utilizing the file access features of the PDPII under RTII or the PDP8 under OS8, assembly of programs in standrad Intel format for the 8080 or the 8085 is performed. The few differences between MAC80 is performed. The few differences between MAC80 and this assembler are outlined in the program comments.

Required Hardware

DEC PDP8 capable of running OS8 and Fortran II or DEC PDPII capable of running RTII and Fortran IV (16K words of memory in either machine is enough memory for 300 symbols in the symbol table)

Required Software

OS8 or RTII and at least Fortran II (although specifically designed for these machines/operating systems, conversion to other machines would be straight forward)

Input Parameters

Source file

Output Results

Listing of assembled program with errors also printed on the console device. And a symbol table.

Intel compatible HEX object file

Registers Modified:	Programmer: Rex Tracy
RAM Required:	Company: Colorado State University
ROM Required:	Address: Elec. Engr. Dept. - CSU
Maximum Subroutine Nesting Level:	City: Ft. Collins
Assembler/Compiler Used: DEC Fortran II(pdp8) Fortran II(pdp11)	State: Colorado, 80523

Ref. #C30 A

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

8008 Cross Assembler for 8085 - MACRO Definition Set -- M8008.SRC

Function

Permits assembly of programs written in 8008 assembly language with an 8080 Macro Assembler.

Required Hardware

ISIS II System

Required Software

8080/8085 Macro Assembler

Input Parameters

A source program written in 8008 assembly language with only slight modifications which are specifically detailed in the source listing of M8008.SRC.

Output Results

An assembly of the source program to be used as is to create an absolute paper tape or the list as input to the 8080 program LPSTPR which follows to output a more readable object listing on a lineprinter.

C30A is offered as one program with C30B.

Registers Modified:	Programmer: H. Webster
RAM Required:	Company: Bedford Computer Systems
ROM Required:	Address: 3 Preston Court
Maximum Subroutine Nesting Level:	City: Bedford
Assembler/Compiler Used: ISIS-II 8080/8085 Macro Ass.,V2.0	State: Massachusetts 01730

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

LPSTPR Location Included Post Assembly Processor

Function

This program reads the list file resulting from assembling a source file written in Pseudo 8008 assembly language and outputs a very readable object listing to the lineprinter

Required Hardware

ISIS II System

Required Software

8080/8085 Macro Assembler

Input Parameters

List file of an assembled source file written in 8008 assembly language and including at start M8008.SRC, the macro definition set.

Output Results

Very readable object listing on the lineprinter

C30B is offered as one program with C30A.

Registers Modified:	Programmer: H. Webster
RAM Required: 0389H Bytes	Company: Bedford Computer Systems
ROM Required: 0C24H Bytes	Address: 3 Preston Court
Maximum Subroutine Nesting Level:	City: Bedford
Assembler/Compiler Used: ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0	State: Massachusetts 01730

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☒ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

CROSS ASSEMBLER USING HONEYWELL H316/516/716 (Rev.B)

Function

To generate full program listing and object tape from a source tape written in Intel 8080/8085 Assembly Language.

Required Hardware

Honeywell Mini H316/516/716 4K minimum.
Paper Tape Reader, Punch and Teletype.
Line Printer is optional.

Required Software

Honeywell DAP-16 Assembler

Revised 12/78

Input Parameters

Source Tape written in Intel 8080/8085 Assembly Language using ASCII code forced 8 parity (Honeywell Standard) or Even parity

Output Results

Full assembly listing with HEX or Octal machine code optional.
Binary output tape if required.

Registers Modified: N/A	Programmer: Robert G. Yarwood
RAM Required: N/A	Company: John Player & Sons
ROM Required: N/A	Address: Radford Boulevard
Maximum Subroutine Nesting Level: N/A	City: Nottingham
Assembler/Compiler Used: Honeywell DAP-16 Assembler	State: ENGLAND

Ref. #C32

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

MCS-48 Interpretive Cross Assembler.

Function

MCS-48 Interpretive Cross Assembler that runs on an Intellec 8/MOD80.

Required Hardware

Intellec 8/MOD80.
Teletype.

Required Software

Intellec system monitor Vers. 3.0

Input Parameters

Starting address of the program to be assembled.
Assembly language program.

Output Results

Assembled program in RAM at address specified.
Complete listing of address, machine code, and assembly language mnemonic for each instruction.

Registers Modified: ALL	Programmer: M.A. PORDES
RAM Required: 11 Bytes + Stack	Company: GEC Hirst Research Centre
ROM Required: 2412 Bytes	Address: East Lane, Wembley, Middx.
Maximum Subroutine Nesting Level: 8	City: London
Assembler/Compiler Used: MDS 8080/8085 ASSEMBLER V2.0	State: ENGLAND

MCS-48 INTERPRETIVE CROSS ASSEMBLER

The assembler is designed to run on an Intellec 8/MOD 80 and will assemble all of the standard MCS-48 instructions as defined in the MCS-48 Assembly Language Manual with the following exceptions:

1) No labels are allowed
2) The only pseudo-instructions allowed are ORG, END, DS, DB, and DW
3) No operators are allowed
4) The assembler will not accept decimal, binary or octal numbers

All numbers input to the assembler are assumed to be in hexadecimal, typing an H after any hexadecimal number is optional. Also, it is not necessary to type a Ø before a number that starts with A - F.

Operating Instructions

On starting the program the assembler will print the pseudo-instruction ORG, at which point the user must type in the address of the area in RAM into which the assembled program will be placed. The user can then proceed to enter his program.

When entering an instruciton the code part of the instruction must be separated from the operand (if any), by a single space.

If an illegal instruction is entered it is either thrown out immediately or when the appropriate delimiter is typed in, allowing the user to re-enter his instruction.

The data part of all appropriate instructions can be any ASCII character, as well as being a HEX number, providing the character is enclosed within single quotes (').

DB instructions can consist of a sequence of HEX numbers, a string of ASCII characters (providing the string is enclosed within single quotes), or a combination of the two types. The maximum length of "list" is ten HEX numbers or ASCII characters.

Comments can be added to any instruction by typing a semicolon (;) immediately after entering the instruction. The comment can be of any length and is terminated by typing carriage return (CR).

The pseudo-instruction ORG, followed by an address, can be entered at any time during the assembly allowing the user to assemble different parts of his program at different addresses.

When all the instructions in the program have been entered, the pseudo-instruction END is typed to terminate the assembly.

Examples of all the above points are given in the test program.

Ref.#C33

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

SIM48 VERSION 1.3-8048 SIMULATOR

Function

Simulates an 8048/49 microprocessor with 8243 I/O expander.

Required Hardware

8080 Intellec Microcomputer Development System
32K bytes of RAM memory
Flexible diskette drive and controller
Console device

Required Software

ISIS-II Operating System
ISIS-II 8048 Assembler

Input Parameters

Hexadecimal coded file containing the 8048 machine instructions.

Output Results

Interactive simulation.

Available on diskette only

Registers Modified: All	Programmer: E. L. Jones
RAM Required: 0A27H	Company: Wits. University
ROM Required: 219BH	Address: 1 Jan Smuts Ave.
Maximum Subroutine Nesting Level: 3	City: Johannesburg
Assembler/Compiler Used: 8080 Assembler PL/M-80 Compiler	State: South Africa

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

8085 CROSS ASSEMBLER FOR NOVA 1200

Function

Utilizing file access features of the NOVA 1200 under RDOS 5, assembly of programs in standard INTEL format for the 8080 or 8085 is performed. The few differences between MAC80 and this assembler are outlined in the program comments

Required Hardware

NOVA 1200, 4047 single or dual disk drives and 16K words of memory

Required Software

RDOS 5 and FORTRAN IV

Input Parameters

Source file name

Output Results

(SOURCE FILE).LS Assembly listing and symbol table
(SOURCE FILE).RB 8085 object code
Error listing on console

NOTE: THIS IS A MODIFICATION TO PROGRAM C29 WRITTEN BY REX TRACY

Registers Modified:	Programmer: William P. Weber
RAM Required:	Company: Page Communications Engineers,Inc
ROM Required:	Address: 801 Follin Lane
Maximum Subroutine Nesting Level:	City: Vienna
Assembler/Compiler Used: 093-000053-8 REV.05.10NOVA FORTRAN IV	State: Virginia 22180

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 XX 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

RTCOPY

Function

Copies first file from a PDP-11 diskette (RT-11) to an MDS ISIS-II diskette file.

Required Hardware

MDS 800 System running ISIS-II

Required Software

ISIS-II

Input Parameters

A valid ISIS file name. The PDP-11 RT-11 diskette is placed in drive 1 and the ISIS-II diskette is placed in drive Ø. Key 'RTCOPY' followed by the file name desired.

Output Results

The first file on the RT-11 diskette will be copied to the ISIS diskette. Since only the first file is copied, the user should initialize his RT-11 diskette prior to placing the file to be copied onto it.

Registers Modified: N/A	Programmer: Steve Freeman
RAM Required: Less than 1K	Company: Amer-O-Matic Corp.
ROM Required: None	Address: 804-4th Avenue N.
Maximum Subroutine Nesting Level: N/A	City: Birmingham
Assembler/Compiler Used: 8080/8085 Macro Assembler V2.0	State: Alabama

SECTION 7

UNCLASSIFIED PROGRAMS

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D1

☐ 4004 ☒ 8008 ☒ 8080 ☐ 4040

(use additional sheets if necessary)

Program Title

QUICKSORT PROGRAM

Function

SORT AN ARRAY INTO ASCENDING ORDER

Required Hardware

8008/8080 SYSTEM

Required Software

PL/M COMPILER

Input Parameters

1. ADDRESS OF ARRAY (16 BITS)
2. LENGTH OF ARRAY (8 BITS)

Output Results

THE ELEMENTS OF THE ARRAY PASSED ARE SORTED INTO ASCENDING ORDER

Registers Modified: ALL	**Maximum Subroutine Nesting Level:** 2
RAM Required: 546 BYTES	**Assembler/Compiler Used:** PLM80 or PLM
ROM Required: 368 BYTES	**Programmer:** KEN BURGETT
	Company:

```
00001  1
00002  1   /* REF. NO. D1 */
00003  1   /* PROGRAM TITLE QUICKSORT */
00004  1
00005  1   /* QUICKSORT PROCEDURE.
0(  6  1
00007  1   THIS PL/M PROCEDURE SORTS AN ARRAY INTO ASCENDING ORDER
00008  1   USING THE QUICKSORT ALGORITHM. INCLUDED IN THIS LISTING
00009  1   IS THE PROCEDURE, QUICKSORT, AND A TEST DRIVER PROGRAM
00010  1   TO DEMONSTRATE THE CALLING SEQUENCE. NOTE THAT THE
00011  1   PROCEDURE IS WRITTEN WITH AN ASSUMPTION THAT THE NUMBER
00012  1   OF ELEMENTS TO BE SORTED IS LESS THAN OR EQUAL TO 256
00013  1   (LOW,HIGH,UPTR,DPTR,LSTACK,HSTACK,ARRAY$SIZE,A1,
00014  1   AND A2 ARE BYTE VARIABLES) AND THAT THE PRECISION OF
00015  1   THE ARRAY ELEMENTS IS 8 BITS (LIST,TEMP, AND REF ARE
00016  1   BYTE VARIABLES). THESE RESTRICTIONS MAY BE LIFTED BY
00017  1   CHANGING THE DECLARATIONS. NOTE ALSO THAT THE
00018  1   WORKING ARRAYS (LSTACK AND HSTACK) ARE DIMENSIONED
00019  1   BY STACK$SIZE WHERE
00020  1
00021  1        STACK$SIZE >= ARRAY$SIZE.
00022  1
00023  1   */
00024  1
00025  1   QUICKSORT:
00026  1     PROCEDURE (ARRAY,ARRAY$SIZE);
00027  2       DECLARE STACK$SIZE LITERALLY '256';
00028  2       DECLARE TRUE LITERALLY '0FFH', FALSE LITERALLY '0';
00029  2       DECLARE ARRAY ADDRESS;
00030  2       DECLARE ARRAY$SIZE BYTE;
00031  2       DECLARE LIST BASED ARRAY BYTE;
0(  32 2       DECLARE LSTACK(STACK$SIZE) BYTE, HSTACK(STACK$SIZE) BYTE;
00033  2       DECLARE TOP BYTE;
00034  2       DECLARE (LOW,DPTR,UPTR,HIGH) BYTE;
00035  2       DECLARE (REF, TEMP,DECREMENTING) BYTE;
00036  2
00037  2       PUSH:
00038  2         PROCEDURE (A1,A2);
00039  3           DECLARE (A1,A2) BYTE;
00040  3
00041  3           LSTACK(TOP) = A1;
00042  3           HSTACK(TOP) = A2;
00043  3           TOP = TOP + 1;
00044  3         END PUSH;
00045  2
00046  2       /* MAIN PROGRAM */
00047  2
00048  2       TOP = 0;
00049  2       CALL PUSH(0,ARRAY$SIZE);
00050  2       DO WHILE TOP <> 0;
00051  2         TOP = TOP - 1;
00052  3         IF (DPTR:=(LOW:=LSTACK(TOP)))<>(UPTR:=(HIGH:=HSTACK(TOP))) THEN
00053  3         DO;
00054  3           REF = LIST(LOW);
00055  4           DECREMENTING = TRUE;
00056  4           DO WHILE DECREMENTING;
0(  57 4             DO WHILE LIST(DPTR) <= REF AND HIGH > DPTR;
00058  5               DPTR = DPTR + 1;
00059  6             END;
00060  5             DO WHILE LIST(UPTR) >= REF AND LOW < UPTR;
```

```
00061  5                 UPTR = UPTR - 1;
00062  6               END;
00063  5               IF DPTR < UPTR THEN
00064  5               DO;
00065  5               TEMP = LIST(UPTR);
00066  6               LIST(UPTR) = LIST(DPTR);
00067  6               LIST(DPTR) = TEMP;
00068  6               DPTR = DPTR + 1;
00069  6               UPTR = UPTR - 1;
00070  6             END;
00071  5             ELSE
00072  5             DO;
00073  5               IF UPTR > LOW THEN
00074  6               DO;
00075  6                 LIST(LOW) = LIST(UPTR);
00076  7                 LIST(UPTR) = REF;
00077  7                 UPTR = UPTR - 1;
00078  7               END;
00079  6               CALL PUSH(LOW,UPTR);
00080  6               CALL PUSH(DPTR,HIGH);
00081  6               DECREMENTING = FALSE;
00082  6             END;
00083  5           END;
00084  4         END;
00085  3       END;
00086  2     END QUICKSORT;
00087  1
00088  1     /* BEGIN TEST DRIVER */
00089  1
00090  1     DECLARE TEST$ARRAY (16) BYTE INITIAL
00091  1       (0,15,1,14,2,13,3,12,4,11,5,10,6,9,7,8);
00092  1
00093  1       CALL QUICKSORT(.TEST$ARRAY,LAST(TEST$ARRAY));
00094  1
00095  1     EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. D2

☐ 4004 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

BINARY SEARCH ROUTINE

Function

Uses a binary search method to find a character in a table of characters.

Required Hardware

None.

Revised 6/78

Required Software

None.

Input Parameters

E register = character searching for
D register = length of table (1 to 255 characters)
H, L registers = address of first character in the table
Note: The table must be arranged in ascending order.

Output Results

If the character is found,
A register = 1
B register = index of character in the table (0 to 254)

If the character is not found,
A register = 0

Registers Modified:	Maximum Subroutine Nesting Level:
A, B, C, D, H, L	0
RAM Required:	**Assembler/Compiler Used:**
None	MAC 8
ROM Required:	**Programmer:**
45_{10}	
	Company:

```
                        ;REF. NO. D2
                        ;PROGRAM NAME BINARY SEARCH ROUTINE
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;THIS SUBROUTINE PERFORMS A BINARY SEARCH OF A TABLE.
                        ;THE E REGISTER CONTAINS THE CHARACTER BEING SOUGHT, THE D
                        ;REGISTER CONTAINS THE LENGTH OF THE TABLE (1 TO 255 CHARS)
                        ;AND THE H AND L REGISTERS POINT TO THE FIRST CHARACTER
                        ;OF THE TABLE
                        ;IF THE CHARACTER IS FOUND, THE A REGISTER CONTAINS A 1 AND
                        ;THE B REGISTER CONTAINS THE INDEX OF THAT CHARACTER IN THE
                        ;TABLE (0 TO 254).
                        ;IF THE CHARACTER IS NOT FOUND, THE REGISTER CONTAINS A 0.

                        SRCG:
0000 0E00                       MVI     C,0             ;SET LOWER INDEX LIMIT TO 0
                        LOOP:
0002 7A                 LOOP:   MOV     A,D             ;ADD LOWER AND UPPER LIMITS OF INDEX
0003 81                         ADD     C               ;AND DIVIDE BY 2
0004 1F                         RAR                     ;TO GET THE MIDDLE OF THE RANGE
0005 47                         MOV     B,A             ;SAVE INDEX OF MIDDLE IN B
0006 85                         ADD     L               ;ADD ADDRESS OF START OF TABLE
0007 D20B00                     JNC     NCAR1           ;NO CARRY TO SKIP H
000A 24                         INR     H
                        NCAR1:
000B 6F                         MOV     L,A             ;RESTORE THE L REGISTER
                        ;H AND L NOW CONTAIN THE ADDRESS OF THE MIDDLE OF THE TABLE
000C 7E                         MOV     A,M             ;LOAD THE CHARACTERS FROM THE TABLE
000D BB                         CMP     E               ;TWO CHARACTERS THE SAME?
000E DA1800                     JC      LOW             ;NO, SOUGHT CHARACTER IS GREATER
0011 CA2A00                     JZ      MATCH           ;YES, MATCH HAS BEEN FOUND
0014 50                         MOV     D,B             ;NO, SOUGHT CHARACTER IS LESS
0015 C31900                     JMP     CHECK
                        LOW:
0018 48                         MOV     C,B             ;CURRENT INDEX BECOMES LOWER INDEX
                        CHECK:
0019 7D                 CHECK:  MOV     A,L             ;RESET H AND L TO START
001A 90                         SUB     B               ;OF TABLE
001B D21F00                     JNC     NCAR2           ;NO CARRY TTO SKIP H
001E 25                         DCR     H
                        NCAR2:
001F 6F                 NCAR2:  MOV     L,A             ;RESET L
0020 7A                         MOV     A,D             ;CHECK IF LIMITS DIFFER BY 1
0021 91                         SUB     C
0022 FE01                       CPI     1
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0          MODULE   PAGE    1

  LOC  OBJ          SEQ          SOURCE STATEMENT

                      1
                      2 ;REF.  NO.  D2
                      3 ;PROGRAM NAME BINARY SEARCH ROUTINE
                      4 ;
                      5 ;
                      6 ;
                      7 ;
                      8 ;
                      9 ;
                     10 ;THIS SUBROUTINE PERFORMS A BINARY SEARCH OF A TABLE.
                     11 ;THE E REGISTER CONTAINS THE CHARACTER BEING SOUGHT, THE
                     12 ;REGISTER CONTAINS THE LENGTH OF THE TABLE (1 TO 255 CHA
                     13 ;AND THE H AND L REGISTERS POINT TO THE FIRST CHARACTER
                     14 ;OF THE TABLE
                     15 ;IF THE CHARACTER IS FOUND, THE A REGISTER CONTAINS A 1
                     16 ;THE B REGISTER CONTAINS THE INDEX OF THAT CHARACTER IN
                     17 ;TABLE (0 TO 254).
                     18 ;IF THE CHARACTER IS NOT FOUND, THE REGISTER CONTAINS A
                     19
                     20 SRCG:
  0000 0E00          21          MVI     C,0       ;SET LOWER INDEX LIMIT TO 0
                     22 LOOP:
  0002 7A            23          MOV     A,D       ;ADD LOWER AND UPPER LIMITS OF I
  0003 81            24          ADD     C         ;AND DIVIDE BY 2
  0004 1F            25          RAR               ;TO GET THE MIDDLE OF THE RANGE
  0005 47            26          MOV     B,A       ;SAVE INDEX OF MIDDLE IN B
  0006 85            27          ADD     L         ;ADD ADDRESS OF START OF TABLE
  0007 D20B00        28          JNC     NCAR1     ;NO CARRY TO SKIP H
  000A 24            29          INR     H
                     30 NCAR1:
  000B 6F            31          MOV     L,A       ;RESTORE THE L REGISTER
                     32 ;H AND L NOW CONTAIN THE ADDRESS OF THE MIDDLE OF THE TA
  000C 7E            33          MOV     A,M       ;LOAD THE CHARACTERS FROM THE TA
  000D BB            34          CMP     E         ;TWO CHARACTERS THE SAME?
  000E DA1800        35          JC      LOWER     ;NO, SOUGHT CHARACTER IS GREATER
  0011 CA2A00        36          JZ      MATCH     ;YES, MATCH HAS BEEN FOUND
  0014 50            37          MOV     D,B       ;NO,  SOUGHT CHARACTER IS LESS
  0015 C31900        38          JMP     CHECK
                     39 LOWER:
  0018 48            40          MOV     C,B       ;CURRENT INDEX BECOMES LOWER IND
                     41 CHECK:
  0019 7D            42          MOV     A,L       ;RESET H AND L TO START
  001A 90            43          SUB     B         ;OF TABLE
  001B D21F00        44          JNC     NCAR2     ;NO CARRY TTO SKIP H
  001E 25            45          DCR     H
                     46 NCAR2:
  001F 6F            47          MOV     L,A       ;RESET L
  0020 7A            48          MOV     A,D       ;CHECK IF LIMITS DIFFER BY 1
  0021 91            49          SUB     C
  0022 FE01          50          CPI     1
  0024 C20200        51          JNZ     LOOP      ;DIFFERENCE MORE THAN 1 SO REPEA
                     52 INVAL:
```

```
0024 C20200             JNZ     LOOP    ;DIFFERENCE MORE THAN 1 SO REPEAT SEARCH
                INVAL:
0027 3E01               MVI     A,1     ;RETURN WITH 0 AS UNSUCCESSFUL SEARCH
0029 C9                 RET
                MATCH:
002A 3E01               MVI     A,1     ;RETURN WITH 1 AS A SUCCESS
002C C9                 RET
0000                    END
```

```
  LOC  OBJ          SEQ          SOURCE STATEMENT

  0027 3E00           53           MVI     A,0      ;RETURN WITH 0 AS UNSUCCESSFUL S
  0029 C9             54           RET
                      55 MATCH:
  002A 3E01           56           MVI     A,1      ;RETURN WITH 1 AS A SUCCESS
  002C C600           57           ADI     0        ;AND ZERO-FLAG Z = 0
  002E C9             58           RET
                      59           END

PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
CHECK  A 0019    INVAL  A 0027    LOOP   A 0002    LOWER  A 0018    MATCH  A 002
SRCG   A 0000

ASSEMBLY COMPLETE, NO ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D3

☐ 4004 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

DECHL

Function

Macro for decrementing the 16-bit binary contents of the H and L registers.

Required Hardware

None.

Required Software

None.

Input Parameters

H and L registers.

Output Results

(HL-1) → HL

Registers Modified: H,L	**Maximum Subroutine Nesting Level:** 0
RAM Required: None	**Assembler/Compiler Used:** MAC 8
ROM Required: 7 bytes	**Programmer:** John M. Schulein Aeronutronic Ford
	Company: 3939 Fabian Way Palo Alto, Ca. 94303

```
                        ;REF. NO.  D3
                        ;PROGRAM NAME DECHL
                        ;
                        ;
                        ;
                        DECHL   MACRO
                                DCR     L       ;DECREMENT L
                                INR     L       ;INCREMENT L
                                JNZ     SKIP    ;JUMP IF L <> 0
                                DCR     H       ;DECREMENT H
                        SKIP:
                                DCR     L       ;DECREMENT L
                                ENDM
                        ;
                        ;   EXAMPLE OF USE
                        ;
0000 210001                     LXI     H,100H
                      +         DECHL
0003 2D               +         DCR     L       ;DECREMENT L
0004 2C               +         INR     L       ;INCREMENT L
  05 C20900           +         JNZ     SKIP    ;JUMP IF L <> 0
0008 25               +         DCR     H       ;DECREMENT H
                      +SKIP:
0009 2D               +         DCR     L       ;DECREMENT L

000A 2D                         DCR     L       ;DECREMENT L
000B 2C                         INR     L       ;INCREMENT L
000C C20F00                     JNZ     SKIP    ;JUMP IF L <> 0
                       SKIP:
000F 2D                         DCR     L       ;DECREMENT L
0000                            END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D4

☐ 4004 ☒ 8008 ☐ 8080 (use additional sheets if necessary)

Program Title

Morse Code

Function

The program receives message text typed on an ASR 33 teletype and sends the morse code equivalent to output port 10 bit 0. It contains a 256 character buffer so that text can be typed in faster than it is sent. Typing a "control S" will stop code output without stopping additional input to the character buffer. To continue sending a "control G" is typed. Another function is that if one makes an error while typing text it can be corrected using the ← and then typing the correct letter ie: now is the te←ime gor←←←for all good men to.

Required Hardware

5v relay around 20-30ma. pull in to operate code oscillator or transmitter.

Required Software

None

Input Parameters

tty 33 to intellec 8 I/O board UART.

Output Results

International morse code
dot=dot; dot=space between elements of a letter; dash=3 dots
dash=space between letters
space=space bar on tty=7 dots=space between words

Registers Modified: A,B,C,D,E,H,L	**Maximum Subroutine Nesting Level:** 2
RAM Required: 1k	**Assembler/Compiler Used:** Intellec 8/Mod 8 Macro Assembler, Ver 1.0
ROM Required:	**Programmer:** George B. McConnell Hunter Labs
	Company: 9529 Lee Highway Fairfax, Va. 22030

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D5

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

CMDST (CMDS2)

Function

AS STATED ON LISTING
THESE SUBROUTINES ARE USED IN CONJUNCTION WITH OUTPUTTING SINGLE BITS OF CONTROL DATA. A SINGLE BIT OF OUTPUT DATA CAN BE CHANGED WITHOUT AFFECTING PREVIOUSLY ESTABLISHED OUTPUT CONDITIONS.

Required Hardware

PARALLEL I/O INTERFACE IS USED IN CONJUNCTION WITH THIS ROUTINE, BUT IS NOT REQUIRED TO RUN IT.

Required Software

NONE

Input Parameters

AS STATED

Output Results

AS STATED

Registers Modified: A,B (A,B,C)	**Assembler/Compiler Used:** PDP-10 MACRO ASSEMBLER
RAM Required: ONE WORD (TWO WORDS)	**Programmer:** STANLEY J. JACZYNSKI
ROM Required: 14 WORDS (27 WORDS)	**Company:** EXTRION CORPORATION
Maximum Subroutine Nesting Level: UNLIMITED	**Address:** BOX 1226 BLACKBURN INDUSTRIAL PARK GLOUCESTER, MASS 01930

```
                        ;REGF. NO. D5
                        ;PROGRAM NAME CMDST (CMDS2)
                        ;
                        ;
                        ;
                        ;
                        ;
                        ;   TITLE * CMDST *
                        ;   SUBROUTINE FOR CHANGING 1 TO 8 BITS
                        ;   OF AN 8:-BIT COMMAND WORD IN RAM
                        ;   ENTER WITH 8-BIT MASK IN B
                        ;   (366 = CHANGE BITS 0 AND 3 )
                        ;   THE VALUE OF THE BITS IN A
                        ;   AND HL POINT ING TO THE COMMAND WORD
                        ;   ON RETURN, A, AND ADDRESS HL
                        ;   CONTAIN THE NEW COMMAND WORD
0000 FE00               CMDST:  CPI     0       ;IS THE BIT TO BE 0?
0002 78                         MOV     A,B
0003 CA0B00                     JZ      ZSET    ;YES,  DO AN "AND"
0006 2F                         CMA             ;COMPLIEMENTMENT MASK
0007 B6                         ORA     M       ;"OR" WITH COMMAND WWD
0008 C30C00                     JMP     MSET
000B A6                 ZSET:   ANA     M       ;"AND" WITH COMMAND WD
000C 77                 MSET:   MOV     M,A     ;RESTORE MEMORY
000D C9                         RET


                        ;   TITLE ::* CMDS2 *
                        ;   SUBROUTINE FOR CHANGING 1 TO 16 BITS
                        ;   OF A 16-BIT COMMAND WORD IN RAM
                        ;   ENTER WITH 16-BIT MASK IN BC
                        ;   (077775 ≐ BITS 1 AND 15)
                        ;   THE VALUE OF THE BIT (S) IN A
                        ;   AND HL POINTING TO THE COMMAND WORD
                        ;   LSD,M MSD IN NEXT LOC.
                        ;   ON RETURN, BC AND ADDRESS HL, HL+1
                        ;   CONTAIN THE NEW COMMAND WORD
000E FE00               CMDS2:  CPI     0       ;IS THE BIT TO BE 0
0010 79                         MOV     A,C
0011 CA1F00                     JZ      ZSET2   ;YES , WELL GO !"AND"
0014 2F                         CMA             ;COMPLIMENT MASK
0015 B6                         ORA     M       ;"OR" WITH COMMAND LSD
0016 4F                         MOV     C,A     ;STORE NEW LSD IN C
0017 78                         MOV     A,B     ;GET MASK MSD
0018 2F                         CMA             ;COMPLIMENT MASK
0019 23                         INX     H       ;INCREMENT POINTER
001A B6                         ORA     M       ;"OR" WITH COMMAND MSD
001B 47                         MOV     B,A     ;STORE NEW MSD IN B
001C C32500                     JMP     MSET2
```

```
001F A6         ZSET2:  ANA     M       ;"AND" WITH COMMAND LSD
0020 4F                 MOV     C,A     ;STORE NEW LSD IN C
0021 78                 MOV     A,B     ;GET MASK MSD
0022 23                 INX     H       ;BUMP POINTER
0023 A6                 ANA     M       ;"AND2" WITH COMMAND MSD
0024 47                 MOV     B,A     ;STORE NEW LMSD IN B
0025 70         MSET2:  MOV     M,B     ;REPLACE NEW COMMAND
0026 2B                 DCX     H       ;WORD IN RAM
0027 71                 MOV     M,C
0028 C9                 RET
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D12

☐ 4004 ☐ 4040 ☐ 8008 ☑ 8080

(use additional sheets if necessary)

Program Title

Clock Subroutine

Function

Maintains a current time of day, decimal adjusted in BCD, of hours, minutes, and seconds. Must be invoked once each second, usually by an external interrupt. Time is stored in three bytes of memory, in the 24-hour system or, optionally, in the 12-hour system.

Required Hardware

Clock Routine: No specific equipment is designated.
Test Program: Intellec 8/Mod 80 with TTY connected to the console output port.

Required Software

Clock Routine: None.
Test Program: Monitor, Version 3.0 installed in Intellec.

Input Parameters

A standard CALL instruction or a RST instruction externally inserted after a system interrupt signal. The subroutine should be invoked once per second for proper operation.

Output Results

A three-byte section of RAM is modified to reflect the current time each time the subroutine is invoked. This section of RAM may be accessed by other routines to use or display the current time as required.

Registers Modified: Accumulator	**Assembler/Compiler Used:** 8080 Macro Assembler, V3.0
RAM Required: 3 bytes	**Programmer:** M. M. Dodd
ROM Required: 52 bytes	**Company:** Telcom, Inc.
Maximum Subroutine Nesting Level: One (the subroutine itself)	**Address:** 8027 Leesburg Pike Vienna, VA 22180

```
                ;REF NO.  D12
                ;PROGRAM TITLE CLOCK SUBROUTINE
                ;
                ;
                ;
                ;
                ;   "CLOCK" IS A SUBROUTINE WHICH MAINTAINS A
                ;   THREE-BYTE STORAGE OF THE CORRECT TIME,
                ;   EXPRESSED AS TWO BCD DIGITS PER BYTE.  THE
                ;   TIME IS MAINTAINED IN THE 24-HOUR SYSTEM
                ;   OR, OPTIONALLY, IN THE 12-HOUR SYSTEM.
                ;   EACH TIME THIS SUBROUTINE IS INVOKED, THE
                ;   STORED TIME IS INCREMENTED BY ONE SECOND.
                ;
0000 3A3400     CLOCK:  LDA     TIME            ;GET SECONDS
0003 C601               ADI     1               ;INCREMENT SECONDS
0005 27                 DAA                     ;AND ADJUST IT
0006 323400             STA     TIME            ;STORE IT
0009 FE60               CPI     60H             ;CHECK FOR 60 SECS
000B C0                 RNZ                     ;NOT 60 - RETURN
  0C C6A0               ADI     (NOT 60H)+1     ;RESET SECS TO 0
                                                ;ALSO SET CARRY
000E 323400             STA     TIME            ;STORE NEW SECS
0011 3A3500             LDA     TIME+1          ;GET MINUTES
0014 CE00               ACI     0               ;INCREMENT MINUTES
0016 27                 DAA                     ;ADJUST IT
0017 323500             STA     TIME+1          ;STORE MINS
001A FE60               CPI     60H             ;CHECK FOR 60 MINS
001C C0                 RNZ                     ;NOT 60 - RETURN
001D C6A0               ADI     (NOT 60H)+1     ;RESET MINS TO 0
                                                ;ALSO SET CARRY
001F 323500             STA     TIME+1          ;STORE NEW MINS
0022 3A3600             LDA     TIME+2          ;GET HOURS
0025 CE00               ACI     0               ;INCREMENT HOURS
0027 27                 DAA                     ;ADJUST IT
0028 323600             STA     TIME+2          ;STORE HOURS
002B FE24               CPI     24H             ;CHECK FOR 24 HOURS
                        CPI     12H             ;CHECK FOR 12 HOURS - REMOVE
                                                ;SEMICOLON TO ACTVATE THIS
                                                ;INSTRUCTION AND DELETE
                                                ;ABOVE INSTRUCTION FOR 12-
                                                ;HOUR OPERATION
002D C0                 RNZ                     ;NOT MAX HOURS - RETURN
002E C6DC               ADI     (NOT 24H)+1     ;RESET HOURS TO 0
                                                ;ALSO SET CARRY
                        ADI     (NOT 12H)+1     ;RESET HOURS TO 0 IF
                                                ;USING 12-HOUR SYSTEM.  REMOVE
                                                ;SEMICOLON TO ACTIVATE.  DELETE A
                                                ;INSTRUCTION
```

```
0030 323A00            STA     TIME+2          ;STORE NEW HOURS
0033 C9                RET
0034 000000    TIME:   DB      0,0,0
0000                   END
```

intel MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D13

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

Interrupt driven clock routine

Function

Updating of clock located in RAM based on 100 ms time intervals. Pulses arriving on interrupt line. Four storage locations reserved for 100ms counter, secs counter, mins counter and hour counter. One location for interval counter, one for preset interval and one for flag indicating interval has elapsed. Updating of clock takes about 70 microsecs.

Required Hardware

External clock oscillator or divider to produce interrupt pulses 100 ms apart.
CRT for test of program with the supplied test routine

Required Software

Intellec 8 Monitor Ver 3.0 for testing of program
All software to utilize information from clock

Input Parameters

If required: Preset interval and starting time

Output Results

Incrmenting of storage locations for clock and interval counters, setting of interval flag when interval has elapsed

Registers Modified: None	**Assembler/Compiler Used:** 8080 Ver 3.0 Assembler
RAM Required: 7 bytes + 4 bytes of stack	**Programmer:** Tor M. Jansen
ROM Required: 67 bytes	**Company:** Central laboratory Norwegian telecom. Administr.
Maximum Subroutine Nesting Level: 1	**Address:** P.O.B. 83 2007 Kjeller, NORWAY

98-0348

```
                ;REF. NO. D13
                ;PROGRAM NAME INTERRUPT CLOCK DRIVEN CLOCK ROUTINE
                ;
                        ;
                        ;   TIME  750312 TJA
                        ;
                        ;   INTERRUPTED  DRIVEN CLOCK ROUTINE
                        ;
                        ;   INTERRUPT INSTRUCTION RST 8 (CF)
                        ;
                        ;   INSTRUCTION AT 0008-JMP TIME
                        ;
                        ;
1000                    ORG     1000H
1000 00         MSC:    DB      0
1001 00         SECC:   DB      0
1002 00         MINC:   DB      0
1003 00         HRC:    DB      0
1004 00         INTC:   DB      0
1005 0A         PINT:   DB      10
 1006 00        FINT:   DB      0
                        ;
                        ;MACRO FOR ADDRESSING, INCREMENTING


                        ;AND COMPARING THE DIFFERENT
                        ;COUNTERS IN ROUTINE
                        ;NAME IS THE NAME OF COUNTER
                        ;COUNT IS MAX VALUE OF COUNTER
                        ;
                COUNT   MACRO   NAME ,VALUE
                        LXI     H,NAME          ;SET COUNTER ADDRESS
                        MVI     A,VALUE         ;SET CMAX COUNT
                        CALL    INCOM
                        ENDM
                        ;SUBROUTINE FRO INCREMENTING AND
                        ;DIFFERENT COUNTERS AND COMPARE IT
                        ;TO VALUE.  IF NOT EQUAL SKIP TO
                        ;END OF PROGRAM
                        ;
1007 34         INCOM:  INR     M               ;INCR COUNTER NAME
1008 BE                 CMP     M
1009 C26610             JNZ     FIN1
100C 3600               MVI     M,0
100E C9                 RET
                        ;
                        ;MAIN CLOCK ROUTINE
                        ;
100F F5         TIME:   PUSH    PSW
```

```
1010 E5                 PUSH    H               ;SAVE REGISTERS
                +       COUNT   MSC,10          ;UPDATE MSC COUNTER
1011 210010     +       LXI     H,01000H                ;SET COUNTER ADDRESS
1014 3E0A       +       MVI     A,0000AH                ;SET CMAX COUNT
1016 CD0710     +       CALL    INCOM

1019 210010             LXI     H,MSC           ;SET COUNTER ADDRESS
101C 3E0A               MVI     A,0000AH        ;SET MAX COUNT
101E CD0710             CALL    INCOM

                +       COUNT   SECC,60         ;UPSDATE SEC-COUNTER
1021 210110     +       LXI     H,01001H                ;SET COUNTER ADDRESS
1024 3E3C       +       MVI     A,0003CH                ;SET CMAX COUNT
1026 CD0710     +       CALL    INCOM

1029 210110             LXI     H,SECC          ;SET COUNTER ADDRESS
102C 3E3C               MVI     A,0003CH        ;SET MAX COUNT
102E CD0710             CALL    INCOM

1031 210410             LXI     H,INTC
1034 34                 INR     M               ;INCR INTERVAL COUNTER
  `35 3A0510            LDA     PINT            ;GET PRESENT INTERVAL
  038 BE                CMP     M
1039 C24310             JNZ     LAB1
103C 3600               MVI     M,0             ;RESET INTERVAL COUNTER
103E 3E01               MVI     A,1
1040 320610             STA     FINT            ;SET INTERVAL FLAG
               +LAB1:   COUNT   MINC,60 ;UPDATE MINUTE COUNTER
1043 210210     +       LXI     H,01002H                ;SET COUNTER ADDRESS
1046 3E3C       +       MVI     A,0003CH                ;SET CMAX COUNT
1048 CD0710     +       CALL    INCOM

104B 210210             LXI     H,MINC          ;SET COUNTER ADDRESS
104E 3E3C               MVI     A,0003CH        ;SET MAX COUNT
1050 CD0710             CALL    INCOM

                +       COUNT   HRC,24          ;UPSDATE HOUR COUNTER
1053 210310     +       LXI     H,01003H                ;SET COUNTER ADDRESS
1056 3E18       +       MVI     A,00018H                ;SET CMAX COUNT
1058 CD0710     +       CALL    INCOM

105B 210310             LXI     H,HRC           ;SET COUNTER ADDRESS
105E 3E18               MVI     A,00018H        ;SET MAX COUNT
1060 CD0710             CALL    INCOM

1063 C36710             JMP     FIN
1066 E1         FIN1:   POP     H               ;DUMMY POP
  067 E1        FIN:    POP     H
1068 F1                 POP     PSW             ;RESTORE REGISTERS
1069 FB                 EI
```

```
106A C9                 RET
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. D14

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Calendar Subroutine

Function

Uses three bytes of RAM to store the current date arranged as two BCD digits per byte. The date is adjusted for months with 28, 29, 30, or 31 days and February is adjusted for leap years 1976, 1980, and 1984.

Required Hardware

Calendar Subroutine - None
Test Program - Intellec 8/Mod 80 with TTY connected to console output port.

Required Software

Calendar Subroutine - None
Test Program - Monitor V3.0

Input Parameters

The subroutine should be called once per day, probably by using an external interrupt. Each time it is called, it will add one day to the stored date and make any necessary adjustments.

Output Results

The three bytes of data will be modified each time the subroutine is called. The date may be examined by other routines in the main program.

Registers Modified: None (all saved)	**Assembler/Compiler Used:** 8080 Macro Assembler V3.0
RAM Required: 9 bytes (including stack)	**Programmer:** Donald E. Shorter
ROM Required: 109 bytes	**Company:** Telcom, Inc.
Maximum Subroutine Nesting Level: 1 (the subroutine itself)	**Address:** 8027 Leesburg Pike Vienna, Va. 22180

```
                ;REF. NO. D14
                ;PROGRAM NAME CALENDAR SUBROUTINE
                ;
                ;
                ;
                ;   'CAL' IS A CALENDAR SUBROUTINE WHICH
                ;   MAINTAINS THE CURRENT DATE WHEN CALLED
                ;   ONCE A DAY, THE IS ALWAYS ADJUSTED FOR
                ;   MONTHS WITH 28, 29, 30, OR 31 DAYS
                ;   FEBRUARY IS ADJUSTED FOR LEAP YEARS 1976, 1980
                ;   AND 1984.  ADDITIONAL LEAP YEARS MAY BE ADDED
                ;   IF DESIRED.
                ;
0000 E5         CAL:    PUSH    H
0001 F5                 PUSH    PSW         ; SAVE HL AND PSW
0002 216E00             LXI     H,DAY
0005 7E                 MOV     A,M
0006 FE31               CPI     31H
0008 DA2400             JC      FIX
000B 3601       CAL1:   MVI     M,1         ; RESET DAY TO 1
000D 2B                 DCX     H           ; TO MONTH
000E 7E                 MOV     A,M         ;
000F C601               ADI     1           ; INCREMENT MONTH
0011 27                 DAA                 ; ADJUST IT
0012 77                 MOV     M,A         ; STORE IT
0013 FE13               CPI     13H         ; CHECK FOR JAN
0015 DA2100             JC      CAL3        ; RETURN IF NOT JAN
0018 3601               MVI     M,1         ; RESET MONTH TO JAN
001A 23                 INX     H
001B 23                 INX     H           ; TO YEAR
001C 7E         CAL2:   MOV     A,M
001D C601               ADI     1
001F 27                 DAA
0020 77                 MOV     M,A
0021 F1         CAL3:   POP     PSW
0022 E1                 POP     H
0023 C9                 RET
0024 3A6D00     FIX:    LDA     MONTH
0027 FE04               CPI     4           ; APRIL
0029 CA4300             JZ      FIX1
002C FE06               CPI     6           ; JUNE
002E CA4300             JZ      FIX1
0031 FE09               CPI     9           ; SEPTEMBER
0033 CA4300             JZ      FIX1
0036 FE11               CPI     11H         ; NOVEMBER
0038 CA4300             JZ      FIX1
  3B FE02               CPI     2           ; FEBRUARY
003D CA4C00             JZ      FIX2
0040 C31C00             JMP     CAL2        ; NO FIX NECESSARY
```

```
0043 7E        FIX1:   MOV     A,M
0044 FE30              CPI     30H     ; CHECK FOR MAX DAYS
0046 DA1D00            JC      CAL2+1  ; NO FIX <30 DAYS
0049 C30B00            JMP     CAL1    ; GO RESET DAYS
 '4C 3A6F00    FIX2:   LDA     YEAR    ; GET CURRENT YEAR
-.04F FE76             CPI     76H
0051 CA6700            JZ      LEAP
0054 FE80              CPI     80H
0056 CA6700            JZ      LEAP
0059 FE84              CPI     84H
005B CA6700            JZ      LEAP
005E 7E                MOV     A,M     ; GET DAY
005F FE28              CPI     28H     ; CHECK FOR MAX DAYS
0061 DA1D00    FIX3:   JC      CAL2+1  ; NO FIX <MAX DAYS
0064 C30B00            JMP     CAL1
0067 7E        LEAP:   MOV     A,M     ; GET DAYS
0068 FE29              CPI     29H     ; CHECK FOR MAX DAYS
006A C36100            JMP     FIX3
006D 01        MONTH:  DB      1
006E 01        DAY:    DB      1
006F 75        YEAR:   DB      75H
 '00                   END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D15

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

PASS

Function

Program PASS transfers addresses of parameters between a calling program and subroutine. See attached sheet for details.

Required Hardware

N/A

Required Software

N/A

Input Parameters

Parameter addresses defined by calling program.

Output Results

The above parameter addresses are moved to an area reserved by the subroutine.

Registers Modified: All	**Assembler/Compiler Used:** 8080 Macro Assembler Ver 3.0
RAM Required: As required by calling program and subroutine	**Programmer:** Richard Young
ROM Required: 24 bytes for PASS	**Company:** Naval Air Rework Facility Code 323
Maximum Subroutine Nesting Level:	**Address:** U.S. Naval Air Station Alameda, California 94501

FUNCTION: Program PASS transfers addresses of parameters between a calling program and subroutine. These addresses, defined by the calling program, are moved into an area reserved by the subroutine. The calling program is assumed to have the following coding sequence:

```
          .
          .
          .
      CALL SUBR
       DW P1              Address of P1
       DW P2              Address of P2
          .
          .
          .
       DW PN              Address of Pn
(Next Instruction)
```

The general form of the subroutine must be:

```
PASS EQU XXXXH          Starting address of PASS.
IP1  DW  0
IP2  DW  0
       .
       .                 List Parameters in Label Field
       .
IPN  DW  0
SUBR CALL PASS
     DW IP1
(First instruction of Subroutine)
          .
          .
          .
        RET
```

PASS also sets the correct return address in the stack.

```
                ;REF. NO. D15
                ;PROGRAM NAME PASS
                ;
                ;
                ;
                ;
                ;    PROGRAM "PASS" TRANSFERS ADDRESSES OF PARAMETERS
                ;    BETWEEN A SUBROUTINE AND IT'S CALLING PROGRAM.
                ;
                ;
1500                    ORG     1500H
1500 E1         PASS:   POP     H       ;GET ADDRESS PUSHED BY SUBROUTINE
1501 5E                 MOV     E,M     ;SET DE TO POINT TO START
1502 23                 INX     H       ;    OF SUBROUTINE'S
1503 56                 MOV     D,M     ;    PARAMENTER LIST.
1504 01FCFF             LXI     B,-4    ;LOAD -4 INTO BC AND ADD TO JHL TO
1507 09                 DAD     B       ;    OBTAIN SUBROUTINE ENTRY ADDRESS.
1508 7D                 MOV     A,L     ;STORE LOW 8-BITS IN A.
1509 E3                 XTHL            ;GET ADDRESS PUSHED BY CALLING
                                        ;    PROGRAM-USE AS POINTER TO
                                        ;    START OF PARAMETER LIST.
150A 4E         LOOP:   MOV     C,M     ;LOAD PARAMETER
150B 23                 INX     H       ;    ADDRESS
150C 46                 MOV     B,M     ;    INTO BC.
150D 23                 INX     H       ;INCREMENT POINTER TO NEXT PARAMETER.
150E EB                 XCHG            ;EXCHANGE DE WITH HL.
150F 71                 MOV     M,C     ;STORE PARAMETER ADDRESS
1510 23                 INX     H       ;    IN SUBROUTINE'S
1511 70                 MOV     M,B     ;    PARAMETER LIST.
1512 23                 INX     H       ;INCREMENT POINTER TO NEXT PARAMETER.
1513 EB                 XCHG            ;EXCHANGE DE AND HL.
1514 BB                 CMP     E       ;COMPARE E WITH A.
1515 C20A15             JNZ     LOOP    ;RETURN TO LOOP IF THERE IS ANOTHER
1518 E3                 XTHL            ;    PARAMETER; OTHERWISE SET STACK FOR
                                        ;    PROPER SUBROUTINE RETURN TO
                                        ;    CALLING PROGRAM.
1519 110500             LXI     D,5     ;SET DE=5 AND ADD TO HL TO
151C 19                 DAD     D       ;    OBTAIN PROPER RETURN TO SUBROUTINE.
151D E9                 PCHL            ;RETURN TO SUBROUTINE.
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D17

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Data Array Move

Function

A contiguous array of data may be relocated in memory, regardless of the magnitude and direction of the move. The source and destination array locations may overlap. The max. array size is 2^{16} bytes.

Required Hardware

Required Software

Any program sequence which stores input parameters into RAM in the format indicated below.

Input Parameters

Source array starting address, 2 bytes:
<SSA> = Lo order source array starting address
<SSA + 1> = Hi order source array starting address

Source array ending address, 2 bytes:
<SEA> = Lo order source array ending address
<SEA + 1> = Hi order source array ending address

NOTE: Source array ending address must be greater than source array starting address.

Output Results

Destination array starting address, 2 bytes:
<DSA> = Lo order destination array starting address
<DSA + 1> = Hi order destination array starting address

Array of data starting at source array starting address and ending at source array ending address has been stored starting at destination array starting address.

Registers Modified:	Assembler/Compiler Used:
A, F, B, C, D, E, H, L	Intellec 8/Mod 80 Ver. 2.0
RAM Required: 6 bytes	**Programmer:** Richard Dérickson
ROM Required: 58 bytes	**Company:** Logical Services Inc.
Maximum Subroutine Nexting Level: 1	**Address:** 1901 Old Middlefield Rd., Suite 17 Mountain View, CA 94043

```
                        ;REF. NO. D17
                        ;PROGRAM TITLE DATA ARRAY MOVE
                        ;
                        ;
                        ;
                        ;DATA ARRAY MOVE SUBROUTINE
                        ;
0100                    ORG 0100H
0100            SSA:    DS 2            ;<SSA>=SOURCE ARRAY START ADR
0102            SEA:    DS 2            ;<SEA>=SOURCE ARRAY END ADR
0104            DSA:    DS 2            ;<DSA>=DESTINATION ARRAY START ADR
0106 2A0001     DAM:    LHLD SSA
0109 EB                 XCHG            ;DE=SOURCE START
010A 2A0201             LHLD SEA        ;HL=SOURCE END
010D 7D                 MOV A,L         ;DETERMINE ARRAY SIZE-1
010E 93                 SUB E
010F 4F                 MOV C,A
0110 7C                 MOV A,H
0111 9A                 SBB D
0112 47                 MOV B,A         ;BC=ARRAY SIZE-1
 !13 2A0401             LHLD DSA        ;HL=DESTINATION START
0116 7D                 MOV A,L         ;DETERMINE MOVE DIRECTION
0117 93                 SUB E
0118 7C                 MOV A,H
0119 9A                 SBB D           ;IF CARRY, SSA>DSA (REVERSE DIR)
011A D22801             JNC FWD         ;SET UP FOR FORWARD MOVE
011D CD3701     REVL:   CALL TEST       ;DECREMENT ARRAY SIZE
0120 C8                 RZ              ;RETURN WHEN ARRAY SIZE=0
0121 1A                 LDAX D          ;A=SOURCE DATA
0122 77                 MOV M,A         ;DATA STORED AT DESTINATION
0123 13                 INX D           ;SOURCE ADR INCREMENTED
0124 23                 INX H           ;DESTINATION ADR INCREMENTED
0125 C31D01             JMP REVL        ;REITERATE REVERSE LOOP
0128 09         FWD:    DAD B           ;HL=DESTINATION ARRAY TOP ADR
0129 EB                 XCHG            ;HL=SOURCE BOTTOM, DE=DEST TOP
012A 09                 DAD B           ;HL=SOURCE TOP
012B EB                 XCHG            ;HL=DEST TOP, DE=SOURCE TOP
012C CD3701     FWDL:   CALL TEST       ;DECREMENT ARRAY SIZE
012F C8                 RZ              ;RETURN WHEN ARRAY SIZE=0
0130 1A                 LDAX D          ;A=SOURCE DATA
0131 77                 MOV M,A         ;DATA STORED AT DESTINATION
0132 1B                 DCX D           ;SOURCE ADR DECREMENTED
0133 2B                 DCX H           ;DESTINATION ADR DECREMENTED
0134 C32C01             JMP FWDL        ;REITERATE FORWARD LOOP
0137 0B         TEST:   DCX B           ;BC=ARRAY SIZE-2
^\38 3EFE               MVI A,0FEH      ;A=-2
 ⊥3A B9                 CMP C
013B C0                 RNZ             ;ARRAY SIZE NOT 0, CONTINUE LOOP
013C 3EFF               MVI A,0FFH
```

```
013E B8                 CMP B          ;Z FLAG INDICATES IF ARRAY SIZE=0
013F C9                 RET
                        ;
                        ;DATA ARRAY MOVE TEST PROGRAM
                        ;
                    ;WRITE ASCENDING DATA FROM LOCATION 0200H THRU 02FFH
0140 210002         INIT:   LXI H,0200H
0143 AF                     XRA A
0144 77             WL:     MOV M,A
0145 23                     INX H
0146 3C                     INR A
0147 C24401                 JNZ WL
014A 00                     NOP             ;ARRAY WRITTEN, MONITOR BREAK HERE
                    ;TEST FORWARD MOVE WITH NO OVERLAP OF ARRAYS
014B 210002         T1:     LXI H,0200H ;SSA ASSIGNED 0200H
014E 110F02                 LXI D,020FH ;SEA ASSIGNED 020FH
0151 01E802                 LXI B,02E8H ;DSA ASSIGNED 02E8H
0154 C37801                 JMP EXEC
                    ;TEST FORWARD MOVE WITH OVERLAP OF ARRAYS
0157 211102         T2:     LXI H,0211H ;SSA ASSIGNED 0211H
015A 111802                 LXI D,0218H ;SEA ASSIGNED 0218H
 \5D 011202                 LXI B,0212H ;DSA ASSIGNED 0212H
0160 C37801                 JMP EXEC
                    ;TEST REVERSE MOVE WITH NO OVERLAP OF ARRAYS
0163 21C702         T3:     LXI H,02C7H ;SSA ASSIGNED 02C7H
0166 11E602                 LXI D,02E6H ;SEA ASSIGNED 02E6H
0169 012702                 LXI B,0227H ;DSA ASSIGNED 0227H
016C C37801                 JMP EXEC
                    ;TEST REVERSE MOVE WITH OVERLAP OF ARRAYS
016F 215F02         T4:     LXI H,025FH ;SSA ASSIGNED 025FH
0172 11C402                 LXI D,02C4H ;SEA ASSIGNED 02C4H
0175 014F02                 LXI B,024FH ;DSA ASSIGNED 024FH
0178 220001         EXEC:   SHLD SSA
017B EB                     XCHG
017C 220201                 SHLD SEA
017F 210401                 LXI H,DSA
0182 71                     MOV M,C
0183 23                     INX H
0184 70                     MOV M,B     ;SSA,SEA,DSA, LOADED
                    ;CALL DATA ARRAY MOVE SUBROUTINE
0185 CD0601                 CALL DAM
0188 C38801         LOOP:   JMP LOOP    ;ARRAY MOVED, MONITOR BREAK HERE
0000                        END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. D18

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

SHELLSORT

Function

SORTS ARRAYS IN PLACE USING SHELL'S METHOD (DIMINISHING INCREMENT)

Required Hardware

MCS-80

Required Software

DRIVER PROGRAM

Input Parameters

(ADDRESS OF ARRAY, LENGTH)

Output Results

ARRAY ELEMENTS ARE SORTED WITH SMALLEST ELEMENT FIRST

Registers Modified:	**Assembler/Compiler Used:** PL/M 8008/8080
RAM Required: 12 BYTES	**Programmer:** CHARLES A. PALERMO
ROM Required: approx. 1E0H BYTES	**Company:** GENERAL DYNAMICS
Maximum Subroutine Nesting Level: 6	**Address:** MZ 2119, Box 748 FT. WORTH, TEXAS 76101

817/732-4811

```
00001  1
00002  1   /*REF. NO. D18 */
00003  1   /*PROGRAM TITLE SSORT */
00004  1
00005  1
00006  1   /*DRIVER PROGRAM USED TO VERIFY PROCEDURE. */
00007  1   DECLARE B(10) BYTE INITIAL (10 , 9, 8, 7, 6, 5, 4,
00008  1   3, 2, 1);
00009  1   DECLARE I BYTE, N BYTE INITIAL (10);
00010  1
00011  1
00012  1   SHELLSORT: PROCEDURE (ARRAYPTR,N);
00013  2
00014  2   /* THIS PROCEDURE IS USED TO SORT BYTE ARRAYS OF LENGTH N.
00015  2    RE-DECLARING A AND TEMP AS ADDRESS. ARRAYPTR IS THE
00016  2    ADDRESS OF THE FIRST ELEMENT OF THE ARRAY TO BE SORTED.
00017  2    IT IS RESONABLY EFFICIENT FOR N<=1000 AND DOESN'T DEGRADE
00018  2    IF THE ARRAY IS IN ORDER(POSSIBLE PROBLEM WITH QUICKSORT)
00019  2    SEE KNUTH VOL.III  SEARCHING AND SORTING FOR DETAILS. */
00020  2    DECLARE ARRAYPTR ADDRESS, A BASED ARRAYPTR BYTE,
00021  2    (N,I) ADDRESS, (SWITCHED,J,K,TEMP) BYTE, INCR ADDRESS;
00022  2    IF N <=1 THEN RETURN;
00023  2    INCR=N/2 ;
00024  2    DO WHILE INCR>0;
00025  2     DO I=0 TO INCR;
00026  3     SWITCHED=1;
00027  4     DO WHILE SWITCHED;
00028  4      SWITCHED=0;K=I;J=INCR+I;
00029  5      DO WHILE J<N;
00030  5      IF A(K)>A(J) THEN
00031  6      DO;
00032  6       TEMP=A(K); A(K)=A(J); A(J)=TEMP; SWITCHED=1;
00033  7      END;
00034  6     K=J;
00035  6     J=J+INCR;
00036  6      END;
00037  5     END;
00038  4    END;
00039  3   INCR=INCR/2;
00040  3   END;
00041  2   RETURN;
00042  2   END SHELLSORT;
00043  1
00044  1   /* THE FOLLOWING IS THE REST OF THE TEST DRIVER PROGRAM */
00045  1   CALL SHELLSORT (.B, 10);
00046  1   DO I=0 TO N-1;
00047  1   OUTPUT(1) = B(1);
00048  2   END;
00049  1   EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D19

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

Text Storage Program

Function

Allows text to be stored in memory using a letter of the alphabet as a pointer. After the message is stored, it can be retrieved by depressing a single key on the TTY. Up to 32 messages may be stored and retrieved independently.

Required Hardware

8080, memory and TTY I/O port

Required Software

None, other than this program

Input Parameters

See program listing heading

Output Results

Output is previously stored message

Registers Modified:	Assembler/Compiler Used:
All	Cross 11
RAM Required:	**Programmer:**
100H & buffer	Paul F. Fitts
ROM Required:	**Company:**
100H max	Innovatek Microsystems Inc.
Maximum Subroutine Nesting Level:	**Address:** Smithfield Road Millerton, New York 12546
1	

```
                ;REF. NO. D19
                ;PROGRAM NAME TEXT STORAGE PROGRAM
                ;
                ;
                ;
                ;

                ;LOAD CHAR TO MEM BUFFER SAVING POINTER
                ;READ CHAR FROM MEM BUFFER USING POINTER
                ;POINTER IS LOW 5 BITS FROM TTY KBD
                    ;GIVING 32 POSSIBLE POINTERS
                ;
                    ;INNOVATEK MICROSYSTEMS INC
                    ;SMITHFIELD RD
                    ;MILLERTON, N.Y. 12546
                    ;TEL 914-373-9122
                ;
                    ;PROGRAMMER: PAUL F. FITTS
                    ;DATE: JUNE 26, 1975
                ;CONTROL A CAUSE ENTRY INTO LOAD MODE
                ;CONTROL B ALLOWS ENTRY OF TEXT USING
                    ;IMMEDIATE PREVIOUS CHARACTER AS POINTER,
                    ;ECHOING CHARACTER FOLLOWED BY ^
                ;CONTROL C SIGNALS END OF MESSAGE
                ;CONTROL D CAUSES EXIT FROM TEXT ENTRY MODE
                ;GUESTS MAY BE GREETED BY MESSAGE PREVIOUSLY
                    ;STORED AND RETRIEVED BY TYPING FIRST
                    ;LETTER OF THEIR NAME
                ;PROGRAM REQUIREDS 160 BYTES PLUS 64 BYTES
                    ;FOR TABLE. MESSAGE BUFFER MAY BE ANY SIZE.
                ;TTY READER IS ENABLE BY BIT 0 OF OUT PORT 1
                ;TTY KBD READY INDICATED BY BIT 0 OF IN PORT 1 BEING 0
                ;TTY PRTR READY INDICATED BY BIT 2 OF IN PORT 1 BEING 0
                ;TTY PRINTER BUFFER IS OUT PORT 0
                ;TTY KBD BUFFER IS IN PORT 0
                ;THIS IS COMPATIBLE WITH INTELLEC 8
                ;
                ;CONSTANT DEFINITIONS
0081            SOH     EQU     81H     ;CNTRL A WITH PARITY
0003            ETX     EQU     03H     ;CNTRL C
0084            EOT     EQU     84H     ;CNTRL D
0082            STX     EQU     82H     ;CNTRL B
0001            REN     EQU     01H     ;READER ENABLE
0001            STRD    EQU     01H     ;STATUS OUT
0001            STRG    EQU     01H     ;STATUS REGION
0001            CHRDY   EQU     01H     ;CHAR RDY BIT
   00           KBDB    EQU     0       ;KBD IN PORT
0004            PTRDY   EQU     04H     ;PTR RDY BIT
0000            PTRB    EQU     0       ;PTR OUT PORT
```

```
00FF            RUBOUT  EQU     0FFH    ;RUBOUT
00FC            NETX    EQU     0FCH    ;NOT ETX
                ;
0100                    ORG     100H
0100 31FE1F     INIT:   LXI     SP,STKA ;INIT STACK POINTER
0103 210002             LXI     H,TBLA  ;FILL TBLA WITH START
0106 114202             LXI     D,BFRSP ;ADRS OF BUFFER SPACE
0109 0E20               MVI     C,32
010B 73         INITLP: MOV     M,E
010C 23                 INX     H
010D 72                 MOV     M,D
010E 23                 INX     H
010F 0D                 DCR     C
0110 C20B01             JNZ     INITLP
0113 CD8401     GETC:   CALL    GETCHR  ;CHAR FROM ITTY KBD
0116 FE81               CPI     SOH     ;CNTRL A
0118 CA3601             JZ      LOADC
011B E61F               ANI     1FH
011D 07                 RLC             ;TWO BUYTES / ADRS
011E 4F                 MOV     C,A
011F 0600               MVI     B,0
0121 210002             LXI     H,TBLA  ;TABLE START ADDRESS
0124 09                 DAD     B       ;ADD INDEX, NOW H&L POINT
                                        ;TO TABLE LOCATION WHERE IS
                                        ;STORED START ADRS OF MESSAGE
0125 5E                 MOV     E,M     ;MOVE START ADRS TO D&E
0126 23                 INX     H
0127 56                 MOV     D,M
0128 EB                 XCHG            ;CHAR AADRS IN H&L
0129 46         RTRLP:  MOV     B,M     ;GET CHAR FROM BUFFER & PRINT
012A CD9601             CALL    SNDCHR
012D FEFC               CPI     NETX    ;CHAR INVERTED
012F 23                 INX     H
0130 C22901             JNZ     RTRLP
0133 C31301             JMP     GETC
0136 47         LOADC:  MOV     B,A
0137 CD8401             CALL    GETCHR
013A FE84               CPI     EOT
013C FE84               CPI     EOT     ;CNTRL D
013E CA1301             JZ      GETC    ;LEAVE LOAD CHAR MODE
0141 FE82               CPI     STX     ;CNTRL B, ARSTART MESSAGE
0143 C23601             JNZ     LOADC
0146 CD9601             CALL    SNDCHR  ;ECHO CHAR
0149 2A4002             LHLD    BFPTR   ;START ADRS OF NEXT MESSAGE
014C EB                 XCHG
014D 78                 MOV     A,B
014E E61F               ANI     1FH
0150 07                 RLC
0151 4F                 MOV     C,A
0152 0600               MVI     B,0
```

```
0154 210002          LXI     H,TBLA
0157 09              DAD     B          ;H&L POINT TO TBL WHERE
0158 73              MOV     M,E        ;STRT ADRS IS TO BE STORED
0159 23              INX     H
015A 72              MOV     M,D
015B EB                XCHG             ;  ;BUFFER ADRS IN H&L
015C 065E            MVI     B,'^'
015E CD9601          CALL    SNDCHR
0161 CD8401  LDCLP:  CALL    GETCHR
0164 FEFF            CPI     RUBOUT
0166 CA7B01          JZ      RBUT
0169 47              MOV     B,A
016A CD9601          CALL    SNDCHR     ;ECHO CHAR
016D 70              MOV     M,B        ;STORE IN BUFFER AREA
016E 78              MOV     A,B
016F FE03            CPI     ETX        ;END OF MESSAGE?
0171 23              INX     H
0172 C26100          JNZ     LDCLP
0175 224002          SHLD    BFPTR      ;ESAVE ADRS OF NEXT MESSAGE
0178 C33601          JMP     LOADC
017B 2B      RBUT:   DCX     H          ;ERROR CORRECTION ROUTINE
017C 065C            MVI     B,'\'
017E CD9601          CALL    SNDCHR
0181 C36101          JMP     LDCLP
0184 3E01    GETCHR: MVI     A,REN      ;ENABLE READER
0186 D301            OUT     STRD
0188 AF              XRA     A
0189 D301            OUT     STRD
018B DB01    GTCHLP: IN      STRG       ;CHAR READY?
018D E601            ANI     CHRDY
018F C28B01          JNZ     GTCHLP
0192 DB00            IN      KBDB       ;INPUT CHAR AND COMPLEMENT
0194 2F              CMA
0195 C9              RET
0196 DB01    SNDCHR: IN      STRG       ;PTR BUFFER AVAIL?
0198 E604            ANI     PTRDY
019A C29601          JNZ     SNDCHR
019D 78              MOV     A,B
019E 2F              CMA                ;COMPLEMENT AND OUTPUT
019F D300            OUT     PTRB
01A1 C9              RET
0200                 ORG     200H
0200         TBLA:   DS      64
0240 4202    BFPTR:  DW      BFRSP
0242         BFRSP:  DS      1000H
1FFE                 ORG     1FFEH
1FFE 00      STKA:   DB      0H
                     END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D20

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

Time Sharing Communications

Function

To communicate with medium to large scale computer system as an external time-share user.

Required Hardware

Intellec 8/mod 80, 4k RAM, 2 I/O ports, Acoustic Coupler(Modem)
(Floppy disk for storage of Hex programs - optional)

Required Software

PLM compiler on host machine
Monitor on 8080
Disk Driver on 8080 - Optional

Input Parameters

NA

Output Results

NA

Registers Modified:	Assembler/Compiler Used:
A,B,C,D,E,H,L,SP	PLM
RAM Required:	**Programmer:**
4K	Jim Jowell
ROM Required:	**Company:,**
NA	Univ. of Texas at Houston(UTHERCC)
Maximum Subroutine Nesting Level:	**Address:** 6519 Fannin Houston, Texas 77025
NA	

```
00001  1   800H: /* ASCII TIME SHARING COMMUNICATIONS  */
00002  1     /*      INTELLEC 8/MOD 80          */
00003  1     /*THIS PROGRAM WAS WRITTEN TO BE ABLE TO COMMUNICATE  */
00004  1     /*WITH A CDC TIME SHARING SYSTEM AS A EXTERNAL TIME SHARE*/
00005  1     /*USER.  A BY-PRODUCT IS RECEIVING THE HEX OUTPUT FROM THE */
00006  1     /*PLM COMPILER.                                            */
00007  1     /*AS THE PROGRAM STANDS NOW IT WILL PRINT EITHER ON THE    */
00008  1     /*CONSOLE OR THE LIST DEVICE DEPENDING ON THE CONTENTS OF  */
00009  1     /*MEMORY LOCATION 3. (00 FOR CONSOLE, 80H FOR LIST)        */
00010  1     /*IT IS PRESENTLY SAVING THE HEX CODE ON A FLOPPY DISK.    */
00011  1     /*AS YOU CAN SEE IT IS NOT SAVING ANY OF THE OTHER INFO    */
00012  1     /*IN MEMORY, JUST THE HEX FILE.      */
00013  1       DECLARE LIT LITERALLY #LITERALLY#,DCL LIT #DECLARE#;
00014  1       DCL DA LIT #01H#, TBRE LIT #04H#;
00015  1     /*  DA = DATA AVAILABLE,  TBRE = TRANSMITTER BUFFER EMPTY */
00016  1       DCL PRO LIT #PROCEDURE#, DAT BYTE;
00017  1       DCL SECTR BYTE;
00018  1       DCL BUFF ADDRESS,BUF BASED BUFF BYTE;
00019  1       DCL (COUNT, SWITCH) BYTE;
00020  1       DCL STATUS BYTE, (I,WORK,J,K,H) BYTE;
00021  1       DCL CR LIT #0DH#,LF LIT #0AH#,DATX BYTE;
00022  1     DCL HDG DATA (CR,LF,#ENTER 1 TO 7 FOR DISK, S TO STOP#,CR,LF);
00023  1
00024  1     /*FLOPPY PARAMETERS:  1 BYTE - UNIT, DRIVE     */
00025  1     /*                    1 ADDRESS - SECTOR NUMBER */
00026  1     /*                    1 ADDRESS - BUFFER LOCATION*/
00027  1     /*THE SYSTEM ALWAYS TRANSFERS 128 BYTES TO DISK. */
00028  1
00029  1       DCL DRIV BYTE, (S,B) ADDRESS;
00030  1     /*STARTING FILE LOCATIONS ON THE FLOPPY DISK   */
00031  1       DCL TRACK DATA (044H,0EH,017H,020H,029H,032H,03BH,044H);
00032  1       /*                0    1    2    3    4    5    6    7  */
00033  1
00034  1   WRTDSK: PRO (DATT);
00035  2       DCL WR LIT #37A3H#;
00036  2       DCL DATT ADDRESS;
00037  2       GO TO WR;
00038  2   END WRTDSK;
00039  1
00040  1   CON$PRNT: PRO (CHARS);
00041  2       DCL CHARS BYTE,PRNT LIT #3050H#;
00042  2   PIT: GO TO PRNT;
00043  2   END CON$PRNT;
00044  1
00045  1   LIST$PRNT: PRO (DTX);
00046  2       DCL DTX BYTE,LO LIT #380FH#;
00047  2       GO TO LO;
00048  2   END LIST$PRNT;
00049  1
00050  1   CON$STAT: PRO (STS) BYTE;
00051  2       DCL STS BYTE;
00052  2       K = INPUT (1);
00053  2       RETURN K;
00054  2   END CON$STAT;
00055  1
00056  1   CRT$STAT: PRO (CRSTS) BYTE;
00057  2       DCL CRSTS BYTE;
00058  2       K = INPUT (5);
00059  2       RETURN K;
00060  2   END CRT$STAT;
00061  1
```

```
00062  1    CRT$OUT: PRO (CHAT);
00063  2        DCL CHAT BYTE,DUMM BYTE;
00064  2        DUMM = NOT CHAT;
00065  2        OUTPUT (.) = DUMM;
00066  2    END CRT$OUT;
00067  1
  068  1    CRT$IN: PRO (INP) BYTE;
00069  2        DCL INP BYTE, K BYTE;
00070  2        INP = INPUT (4);
00071  2        K = NOT INP;
00072  2        RETURN K;
00073  2    END CRT$IN;
00074  1
00075  1
00076  1        DCL MEMORY (0) BYTE;
00077  1        DCL CHAR BYTE;
00078  1    HEAD:
00079  1      /*PRINT HEADING DATA TO CONSOLE    */
00080  1        DO I = 0 TO LAST (HDG);
00081  1        DAT = HDG (I);
00082  2        CALL CON$PRNT (DAT);
00083  2        END;
00084  1    LOOP:
00085  1      /* LOOP TO GET DISK FILE NUMBER FOR HEX OUTPUT  */
00086  1      /* OR S TO RETURN TO THE INTEL MONITOR      */
00087  1        STATUS = CON$STAT (K);
00088  1        WORK = STATUS AND 0A;
00089  1        IF WORK <> 0 THEN GO TO LOOP;
00090  1        ELSE
00091  1        DAT = INPUT (0) AND 07FH;
00092  1        OUTPUT (0) = DAT;
  093  1        DAT = NOT DAT AND 07FH;
00094  1        IF DAT = #S# THEN GO TO 3800H;
00095  1        IF DAT < 031H THEN GO TO HEAD; /* FILE ONE  */
00096  1        IF DAT > 037H THEN GO TO HEAD; /* FILE SEVEN  */
00097  1      /*CALCULATIONS NECESSARY TO CREATE DISK ADDRESSES */
00098  1      /*AND BUFFER LOCATION      */
00099  1        DAT = DAT AND 0FH;
00100  1        TRACK = TRACK (DAT);
00101  1        DATX = DAT + 7;
00102  1      /*SYSTEM ON UNIT 0, DATA ON UNIT 1    */
00103  1        DRIV = 1;
00104  1      /*26 SECTORS PER TRACK      */
00105  1        S = TRACK * 26;
00106  1        I = 0;
00107  1        COUNT = 0;
00108  1        SECTR = 0;
00109  1        H = 0;
00110  1        SWITCH = 15;
00111  1        BUFF = .MEMORY;
00112  1    MAIN:
00113  1      /*STATUS LOOP FOR CONSOLE AND INTELLEC CRT PORT  */
00114  1        STATUS = CON$STAT (K);
00115  1        WORK = STATUS AND 0A;
00116  1        IF WORK = 0 THEN GO TO CONSOLE$DATA;
00117  1        STATUS = CRT$STAT (K);
00118  1        WORK = STATUS AND 0A;
  119  1        IF WORK = 0 THEN GO TO CRT$DATA;
00120  1        ELSE GO TO MAIN;
00121  1
00122  1    CONSOLE$DATA:
00123  1        DAT = INPUT (0) AND 07FH;
00124  1    GET$STATUS:
00125  1        STATUS = CRT$STAT (K);
```

```
00126  1        WORK = STATUS AND TBRE;
00127  1        IF WORK <> 0 THEN GO TO GET$STATUS;
00128  1        ELSE
00129  1        OUTPUT (4) = DAT;
00130  1        GO TO MAIN;
00131  1
00132  1    CRT$DATA:
00133  1        DAT = CRT$IN (K) AND 07FH;
00134  1
00135  1    CON$STATUS:
00136  1        STATUS = CON$STAT (K);
00137  1        WORK = STATUS AND TBRE;
00138  1        IF WORK <> 0 THEN GO TO CON$STATUS;
00139  1      /*LIST WILL DEFAULT TO THE CONSOLE      */
00140  1        CALL LIST$PRNT (DAT);
00141  1      /*A STRING OF 40 ASTERISKS PRECEDE THE HEX FILE  */
00142  1        IF DAT = #*# THEN COUNT = COUNT+1;
00143  1        ELSE IF COUNT <> 0 THEN COUNT = COUNT - 1;
00144  1        IF SWITCH = 0 THEN GO TO SAVE;
00145  1      /*SWITCH WILL BE SET WHEN 30 ASTERISKS IN A ROW HAVE */
00146  1      /*BEEN RECEIVED, ALONG WITH THE COLON WHICH PRECEDES */
00147  1      /*EACH DATA LINE    */
00148  1        IF COUNT < 30 THEN GO TO MAIN;
00149  1        IF DAT <> #:# THEN GO TO MAIN;
00150  1        ELSE SWITCH = 0;
00151  1
00152  1    SAVE:
00153  1      /*A STRING OF 40 ASTERISKS FOLLOW THE HEX FILE      */
00154  1      /*H COUNTS THE ASTERISKS THAT ARE TOGETHER.  I AM   */
00155  1      /*ASSUMING THAT THE DATA (HEX INSTRUCTIONS) WILL    */
00156  1      /*NOT HAVE 3 ASTERISK VALUES IN A ROW.              */
00157  1        BUF = DAT;
00158  1        IF DAT = #*# THEN  H = H + 1;
00159  1        IF H = 3 THEN GO TO DSKWRT;
00160  1        BUFF = BUFF + 1;
00161  1        GO TO MAIN;
00162  1    DSKWRT:
00163  1        BUFF = .MEMORY;
00164  1        H = 0;
00165  1        K = 0;
00166  1    ORD:
00167  1      /*THIS SEARCH IS FOR THE STRING OF 10 ZEROS THAT */
00168  1      /*ARE ON THE LST DATA LINE OF THE HEX FILE.      */
00169  1      /* (DO NOT USE AUTO START FOR THIS FUNCTION.)    */
00170  1      /*DSK WRITE IS A 128 BYTE CHARACTER BLOCK.       */
00171  1        DO I = 0 TO 127;
00172  2        IF BUF(I) = #0# THEN H = H + 1;
00173  2        ELSE I H <> 0 THEN H = H - 1;
00174  2        IF BUF(I) = CR THEN IF H > 7 THEN K = 1;
00175  2        END;
00176  1      /*B IS BUFFER LOCATION FOR DISK ROUTINE      */
00177  1        B = BUFF;
00178  1        CALL WRTDSK (.DRIV);
00179  1      /*SECTR IS SECTOR NUMBER FOR DISK ROUTINE  */
00180  1        SECTR = SECTR + 1;
00181  1        IF K <> 0 THEN GO TO STZ;
00182  1        ELSE BUFF = BUFF + 128;
00183  1        GO TO ORD;
00184  1    STZ:
00185  1        SECTR = SECTR + 1;
00186  1      /*SECTOR COUNT REPRESENTS SECTOR + 1 TO SYSTEM */
00187  1      /*DATX IS SECTOR NUMBER FOR SYSTEM DISK FILE CONTROL */
00188  1        S = DATX;
00189  1        B = .SECTR;
```

```
00190  1        CALL WRTOSK (.DRIV);
00191  1      /*ALLOWS FOR MORE THAN ONE FILE TRANSMISSION   */
00192  1        GO TO 800H;
00193  1
00194  1    EOF
^Q PROGRAM ERRORS

NO PROGRAM ERRORS

8080 PLM2 VERS 2.0
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D21

☐ 4004 ☐ 8008 ☒ 8080 ☐ 4040

(use additional sheets if necessary)

Program Title

A Generalized Stepper Motor Driver Program

Function

Operations performed by the program are: using entry variables of number of steps, clockwise or counterclockwise direction and speed of steps - several programs are illustrated for moving a stepper motor in either direction then stopping, moving N steps forward then return N steps, moving motor continuously in either direction until interrupted by a TTY KYBD entry, also programs using an led-photodetector sensor for absolute motor position.

Required Hardware

This program was run on an Intellec, 8 MOD80. TTY on port 0 & 1; motor output on port 3, bits 0-3; sensor input on port 1, bit 7.

Required Software

Existing firmware on Intellec 8, MOD80 - particularly subroutines "CO","CRLF", and "CSTS"

Input Parameters

Initialize: "MOT1" = 0 after pwr on - this is a RAM location used to keep track of motor position. More stepper motors would require additional locations
"STEPS" = number of steps required in double BCD digits - 00 to 99.
"FREQ" = number of 0.5 ms. delays between steps in double BCD digits - 00 to 99.

Output Results

A 2-bit binary counter is incremented or decremented and stored to keep track of motor position. Prior to outputing a new motor position (a step) to a motor, this binary value is transformed (thru a table) to a 4-bit value that is applied (thru transistor drivers) to a "SLO-SYN" type stepper motor.

Registers Modified: All	**Maximum Subroutine Nesting Level:** 3
RAM Required: Minimum one to three Maximum = progr. size	**Assembler/Compiler Used:** 8080 MACRO assembler, ver. 2.0
ROM Required: About 90 loc. for a useful progr.	**Programmer:** Floyd L. Nordin
	Company: Nordin Enterprises Box 1277 - Cupertino, Calif.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D22

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080

(use additional sheets if necessary)

Program Title

IBM Selectric output program

Function

Allows IBM Selectric model 731 to be used as output device.

Required Hardware

MCS-8 system, IBM model 731,appropriate drivers

Required Software

MCS-8 monitor

Input Parameters

Entered with character in register B.

Output Results

Character printed by Selectric.
Input character returned in registers A and B.

Registers Modified: A,B,H,&L	**Assembler/Compiler Used:** 8008 MacroAssembler Ver 2.0
RAM Required: none	**Programmer:** J.Harrison/W.Haskett
ROM Required: 256 Bytes	**Company:** Northeast Electronics
Maximum Subroutine Nesting Level: 2	**Address:** Airport Road,Concord,NH 03301

```
                ;REF. NO. D22
                ;PROGRAM NAME IBM SELECTRIC OUTPUT PROGRAM
                ;
                ;

                        ;SELECTRIC OUTPUT CONVERSION

3100                    ORG     3100H
3100 3E80       START:  MVI     A,80H
3102 B0                 ORA     B       ;SET BIT 8
3103 6F                 MOV     L,A     ;USE FOR TABLE ADDR.
3104 2631               MVI     H,START/100H
3106 0620       WAIT:   MVI     B,20H   ;ROUTINE TO WAIT FOR
3108 DB02       WAIT2:  IN      2       ;SELECTRIC TO BE IDLE
310A E630               ANI     30H
310C C20831             JNZ     WAIT2
310F 05                 DCR     B
3110 C20831             JNZ     WAIT2
3113 DB02               IN      2
3115 E640               ANI     40H     ;CHECK FOR RIGHT MARGIN
 :17 C26431             JNZ     ENDOL
311A 7D                 MOV     A,L
311B FEA0               CPI     0A0H
311D DA5931             JC      CONTRL  ;CHECK IF CONTROL FUNC.
3120 CA4031             JZ      SPACE   ;CHECK IF SPACE
3123 DB02               IN      2
3125 AE                 XRA     M       ;GET SELECTRIC CODE
3126 E680               ANI     80H     ;TEST FOR SHIFT CHENGE
3128 C24531             JNZ     CHANGE
312B 7E                 MOV     A,M     ;GET SELECTRIC CODE
312C B7                 ORA     A       ;TEST FOR VALID CHAR
312D CA3B31             JZ      RESTOR
3130 E67F               ANI     7FH


3132 D30A       PRINT:  OUT     10      ;PRINT CHARACTER
3134 CD7331             CALL    TIME    ;WAIT FOR RESPONSE
3137 3E00               MVI     A,0
3139 D30A               OUT     10      ;RESET OUTPUT
313B 3E7F       RESTOR: MVI     A,7FH   ;MOVE ASCII CODE BACK
313D A5                 ANA     L       ;INTO REGISTER B
313E 47                 MOV     B,A
313F C9                 RET
3140 3E80       SPACE:  MVI     A,80H
3142 C33231             JMP     PRINT   ;PRINT SPACE
 ?45 DB02       CHANGE: IN      2
 ,47 E680               ANI     80H
3149 C25131             JNZ     SHIFT   ;CHANGE SHIFT
314C 3E01               MVI     A,01H
```

```
314E C35331             JMP     CSEND
3151 3E02       SHIFT:  MVI     A,02H
3153 CD6931     CSEND:  CALL    SEND    ;SEND A CONTROL FUNC,
3156 C30631             JMP     WAIT
3159 7E         CONTRL: MOV     A,M     ;GET A CONTROL CODE
315A B7                 ORA     A       ;TEST IF VALID
315B CA3B31             JZ      RESTOR
315E CD6931             CALL    SEND
3161 C33B31             JMP     RESTOR
3164 3E04       ENDOL:  MVI     A,04H   ;START NEW LINE
3166 C35331             JMP     CSEND
3169 D30B       SEND:   OUT     11      ;SEND CONTROL
316B CD7331             CALL    TIME    ;WAIT FOR RESPONSE
316E 3E00               MVI     A,0
3170 D30B               OUT     11      ;RESET OUTPUT
3172 C9                 RET
3173 0620       TIME:   MVI     B,20H   ;ROUTINE TO WAIT
3175 DB02       TIME2:  IN      2       ;FOR RESPONSE
3177 E630               ANI     30H
3179 CA7531             JZ      TIME2
317C 05                 DCR     B
 \7D C8                 RZ
⊿17E C37531             JMP     TIME2
                        ;CODE CONVESION TABLE
3180                    ORG     3180H

3180 0000               DW      00000H
3182 0000               DW      00000H
3184 0000               DW      00000H
3186 0000               DW      00000H
3188 2020               DW      02020H
318A 0408               DW      00804H
318C 0000               DW      00000H
318E 0102               DW      00201H
3190 0040               DW      04000H
3192 8000               DW      00080H
3194 0000               DW      00000H
3196 0000               DW      00000H
3198 0000               DW      00000H
319A 0000               DW      00000H
319C 0000               DW      00000H
319E 0000               DW      00000H
31A0 00FF               DW      0FF00H
31A2 95BE               DW      0BE95H
31A4 F9F5               DW      0F5F9H
31A6 BD15               DW      015BDH
 ^\A8 F0B1              DW      0B1F0H
  ⊿AA FCC6              DW      0C6FCH
31AC 4CC0               DW      0C04CH
31AE 1649               DW      04916H
```

```
31B0 317F               DW      07F31H
31B2 763E               DW      03E76H
31B4 7975               DW      07579H
31B6 343D               DW      03D34H
31B8 7C70               DW      0707CH
31BA 8D0D               DW      00D8DH
31BC 0046               DW      04600H
31BE 00C9               DW      0C900H
31C0 F69C               DW      09CF6H
31C2 A0AC               DW      0ACA0H
31C4 EDA5               DW      0A5EDH
31C6 8E8F               DW      08F8EH
31C8 E1D4               DW      0D4E1H
31CA 87E4               DW      0E487H
31CC A09F               DW      09FA0H
31CE A699               DW      099A6H
31D0 C584               DW      084C5H
31D2 DDD1               DW      0D1DDH
31D4 E7EE               DW      0EEE7H
31D6 DE90               DW      090DEH
31D8 AF81               DW      081AFH
 DA B700                DW      000B7H
31DC 0000               DW      00000H
31DE 0000               DW      00000H
31E0 001C               DW      01C00H
31E2 202C               DW      02C20H
31E4 6D25               DW      0256DH
31E6 0E0F               DW      00F0EH
31E8 6154               DW      05461H
31EA 0764               DW      06407H
31EC 291F               DW      01F29H
31EE 2619               DW      01926H
31F0 4504               DW      00445H
31F2 5D51               DW      0515DH
31F4 676E               DW      06E67H
31F6 5E10               DW      0105EH
31F8 2F01               DW      0012FH
31FA 3700               DW      00037H
31FC 0000               DW      00000H
31FE 0000               DW      00000H
0000                    END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. D23

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

Binary Search

Function

Program searches a table of up to 128 entries. Each entry is composed of a 1 byte argument (Search Key) and an associated result. The result field may be up to 255 bytes for each argument. Result fields must all be the same length. Table format on attached sheet.

Required Hardware

8080 with adequate memory for program and table.

Required Software

None

Input Parameters

Search parameter stored in location K.
Prestructed entry table (Note: assembly listing shown for 128 entries and a result field of 3 bytes).

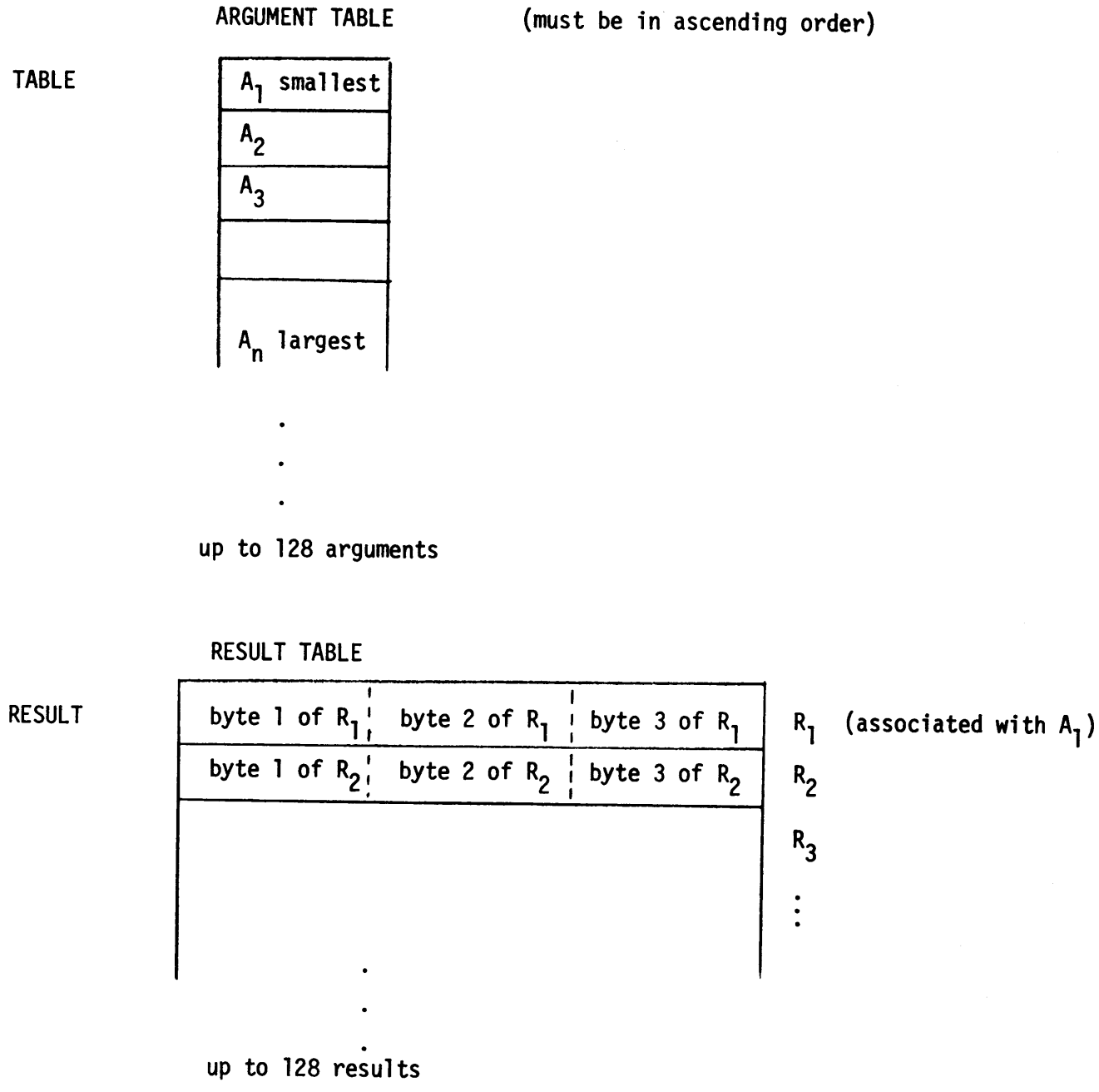
Output Results

Address of 1st byte of result field in HL if a match is found. Also, carry bit ←Ø if success and carry bit ←1 if argument not found in table.

Registers Modified: All except sp	**Assembler/Compiler Used:** 8080 Macro Version 2.0
RAM Required: 79 bytes + table	**Programmer:** H. Corbin
ROM Required:	**Company:**
Maximum Subroutine Nesting Level: 0	**Address:** 11704 Ibsen Dr. Rockville, M.D. 20852

BINARY SEARCH

ENTRY TABLE FORMAT



R length variable and specified by RSULG (format above shown for RSULG = 3) All R's must be the same length.

```
                        ;REF. NO. D23
                        ;PROGRAM TITLE "BINARY SEARCH"
                        ;
                        ;
                        ;
                        ;
0064                    ORG     100
0064 0600               MVI     B,0             ;INITIALIZE LOWER PTR TO 0
0066 0E80               MVI     C,NOENT         ;INITIALIZE UPPER PTR TO N
0068 78         MID:    MOV     A,B             ;FIND MID PTR
0069 81                 ADD     C               ;SUM UPPER & LOWER PTRS
006A 0F                 RRC                     ;DIVIDE BY 2
006B C67F               ADI     07FH            ;CLEAR SIGN BIT
006D 5F                 MOV     E,A             ;SAVE MID PTR
006E 79                 MOV     A,C             ;SET UP FOR FAILURE TEST
006F B8                 CMP     B
0070 DAAC00             JC      NONE            ;LOWER PTR .GT. UPPER PTR
0073 21B000             LXI     H,TABAD         ;GET BASE OF TABLE
0076 1600               MVI     D,0             ;CLEAR MSB
0078 19                 DAD     D               ;COMPUTE MID PTR ADDRESS
0079 3AB200             LDA     K               ;GET SEARCH PARAMETER VALUE
007C BE                 CMP     M               ;COMPARE K-K(MID)
007D CA8D00             JZ      MATCH           ;ARGUMENT FOUND
0080 DA8800             JC      LESS
0083 1C                 INR     E               ;K.GT.K(MID), INC MID PTR
0084 43                 MOV     B,E             ;SET NEW LOWER PTR
0085 C36800             JMP     MID
0088 1D         LESS:   DCR     E               ;K.LT.K(MID), DEC MID PTR
0089 4B                 MOV     C,E             ;SET NEW UPPER PTR
008A C36800             JMP     MID
008D 0600       MATCH:  MVI     B,0             ;COMPUTE RESULT ADDRESS
008F 1609               MVI     D,9             ;BIT COUNTER
0091 4B                 MOV     C,RSULG         ;GET RESULT LENGTH
0092 79         MUL1:   MOV     A,C             ;MULTIPLIER
0093 1F                 RAR
0094 4F                 MOV     C,A             ;LOW-ORDER BYTE OF RESULT
0095 15                 DCR     D
0096 CAA300             JZ      ADDR            ;FINI
0099 78                 MOV     A,B
009A D29E00             JNC     MUL2
009D 83                 ADD     E               ;MULTIPLICAND ADDED IF BIT :1
009E 1F         MUL2:   RAR                     ;CARRY:0 SHIFT MSB
009F 47                 MOV     B,A
00A0 C39200             JMP     MUL1
00A3 21AE00     ADDR:   LXI     H,RSUAD         ;BASE ADDR OF RESULT
00A6 09                 DAD     B               ;COMPUTE MATCH RESULT ADDR
00A7 37                 STC
00A8 3F                 CMC                     ;SUCCESS, C:0
00A9 C3AD00             JMP     DONE
```

```
00AC 37        NONE:   STC                     ;FAILURE, C:1
00AD 76        DONE:   HLT
00AE 3301      RSUAD:  DW      RSULT           ;RESULT BASE ADDRESS
00B0 B300      TABAD:  DW      TABLE           ;ARGUMENT BASE ADDRESS
0080           NOENT   EQU     128             ;NO TABLE ENTRIES
0003           RSULG   EQU     3               ;LENGTH OF RESULT ENTRY
00B2 00        K:      DB      0               ;SEARCH PARAMETER
00B3           TABLE:  DS      128             ;ARGUMENTS
0133           RSULT:  DS      384             ;RESULTS
0000                   END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. D24

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

TIMIT

Function

Interrupt driven, real time clock routine. Provides time and calendar information.

Required Hardware

100 Hz time base oscillator to drive interrupt circuitry of CPU. Vectored interrupt circuitry recommended.

Required Software

Routine to initialize TIMIT's counters (symbolic location TIMER) with current time, day and year.

Input Parameters

Time: hours, miniutes, seconds, hundreths of seconds.
Calendar: day of the year and year.

Output Results

Maintains time in 24 hour format to hundreths of a second; also, day of the year, and year. All values are stored in BCD as 8 bytes starting at location TIMER.

(See additional sheets for detailed description)

Registers Modified:	Assembler/Compiler Used:
A,B,C,D,E	MAC80, version 2.2
RAM Required:	**Programmer:**
105 bytes	Steve Becquer
ROM Required:	**Company:**
	U.C. Berkeley (student)
Maximum Subroutine Nesting Level:	**Address:** 365 Clipper St. San Francisco, Ca. 94114
4	

TIMIT * INTERRUPT DRIVEN, REAL TIME CLOCK

'TIMIT' UTILIZES INTERRUPT CIRCUITRY TO MAINTAIN ACCURATE COUNTING OF TIME OF DAY (TO .01 SEC), DAY OF THE YEAR, AND YEAR, FOR 8080 BASED SYSTEMS. THIS ROUTINE CAN BE USED FOR INTERVAL COUNTING, ELAPSED TIME MEASUREMENT, SAMPLING AT A FIXED RATE (FOR EXAMPLE, EVERY .1 SEC, EVERY HOUR, ETC), AS WELL AS FOR PROVIDING TIME AND DATE INFORMATION (FOR HEADINGS ON LISTINGS, REPORTS, ETC).

A .01 SEC TIME BASE FOR 'TIMIT' MUST BE PROVIDED BY THE USER THROUGH HARDWARE. THIS IS ESTABLISHED BY DIVIDING SYSTEM CLOCK 02 TO 100 HZ, OR WITH AN EXTERNAL OSCILLATOR, TO DRIVE THE INTERRUPT REQUEST INPUT OF THE CPU (IMM 8-83, ETC). WITH EACH CYCLE OF THE TIME BASE OSCILLATOR, THE 8080 RECOGNIZES THE INTERRUPT REQUEST, ISSUES AN INTERRUPT ACKNOWLEDGE SIGNAL, AND ENTERS AN INTERRUPT MACHINE CYCLE. USER PROVIDED LOGIC THEN GATES A 'RST' INSTRUCTION ONTO THE INPUT BUS, CAUSING A SUBROUTINE JUMP INTO 'TIMIT' (SEE INTEL INTELLECT 8/MOD 80 HARDWARE REFERENCE MANUAL FOR DETAILED DESCRIPTION OF INTERRUPT OPERATION). WHEN THUS CALLED, 'TIMIT' WILL INCREMENT ITS TIME COUNTERS BY 1 (.01 SEC), AND RETURN EXECUTION TO WHATEVER ROUTINE WAS EXECUTING AT THE TIME OF THE INTERRUPT. THIS SEQUENCE OF EVENTS WILL THEN REPEAT EVERY .01 SECONDS. FOR MAXIMUM SYSTEM FLEXIBILITY, VECTORED INTERRUPT CIRCUITRY IS RECOMMENDED.

THE USER MUST INITIALIZE THE VARIOUS COUNTERS WITH THE CURRENT TIME, DAY, ETC AT INITIAL STARTUP OF THE SYSTEM. ONCE INITIALIZED, COUNTING IS MAINTAINED BY THE TIME BASE OSCILLATOR, AS LONG AS THE PROCESSOR CAN RESPOND TO ANY INTERRUPT REQUESTS. THE USER MUST BE WARY THAT DURING CERTAIN STATES (HOLD, WAIT, ETC), INTERRUPT REQUESTS ARE NOT SERVICED AND IF THESE LAST FOR A PERIOD OF MORE THAN .01 SEC, 'TIMIT' WILL LOOSE COUNTS.

SYMBOLIC LOCATION 'TIMER' (SEE PROGRAM LISTING) IS THE COMMUNICATION AREA BETWEEN 'TIMIT' AND THE USER. 'TIMER' IS INITIALIZED BY THE USER, AND UPDATED BY 'TIMIT', FOR READ OUT BY THE USER. ALL TIME COUNTS ARE STORED IN BCD, THEREFORE, THE USER MUST ASSURE THAT 'TIMER' IS LOADED WITH VALID BCD VALUES WHICH DO NOT SURPASS THE RANGE OF THE COUNTERS, FOR PROPER OPERATION. THE 8 BYTE FIELD, 'TIMER', IS PARTITIONED AS FOLLOWS:

BYTE	CONTENTS	RANGE
0	HUNDRETHS OF A SECOND	0-99
1	SECONDS	0-59
2	MINUTES	0-59
3	HOURS	0-23
4	LOW ORDER TWO DIGITS OF DAY	1-99 OR 0-66 *
5	HIGH ORDER DIGIT OF DAY	0-3
6	LOW ORDER TWO DIGITS OF YEAR	0-99
7	HIGH ORDER DIGITS OF YEAR	0-99 **

* RANGE DEPENDS ON VALUE OF BYTE 5 AND WHETHER YEAR IS LEAP
** NOT USED BY 'TIMIT'

TIMIT * INTERRUPT DRIVEN, REAL TIME CLOCK

'TIMIT' STORES AND RESTORES STATUS, ACCUMULATOR, AND ALL REGISTERS TO BE MODIFIED, TO ALLOW CONTINUITY OF THE INTERRUPTED PROGRAM. THE USER MUST PROVIDE 4 LEVELS OF STACK STORAGE BY INITIALIZING THE STACK POINTER (LXI SP) BEFORE STARTUP OF 'TIMIT'.

```
                ;REF. NO. D24
                ;PROGRAM TITLE TIMIT
                ;
                ;
                ;
                ;
0008                    ORG     8       ;ENTER VIA AND INTERRUPT
                                        ; AND A RST INSTRUCTION.
0008 C39001             JMP     TIMIT   ;JUMP INTO TIMIT FROM
                                        ;APPROPRIATE RST VECTOR.
0190                    ORG     400     ;
                TIMIT:                  ;AN INTERRUPT DRIVEN,
                                        ;REAL TIME CLOCK.
                                        ;BY STEVE BECQUER,
                                        ;VERS 1.0 76008.
0190 F5                 PUSH    PSW     ;SAVE STATUS AND REGISTERS
0191 C5                 PUSH    B       ;OF INTERRUPTED ROUTINE.
0192 D5                 PUSH    D       ;
0193 11F201             LXI     D,TIMER ;GET ADDRESS OF TIME COUNTERS.
0196 0600               MVI     B,0     ;SET MODULO TO 100.
  98 0E00               MVI     C,0     ;SET STARTING COUNT TO 0.
019A CDE101             CALL    DIVIT   ;UPDATE 100THS OF SEC COUNTER.
019D 0660               MVI     B,60H   ;SET MODULO TO 60.
019F CDE101             CALL    DIVIT   ;UPDATE SECONDS COUNTER.
01A2 CDE101             CALL    DIVIT   ;UPDATE MINUTES COUNTER.
01A5 0624               MVI     B,24H   ;SET MODULO TO 24.
01A7 CDE101             CALL    DIVIT   ;UPDATE HOURS COUNTER.
01AA 13                 INX     D       ;POINT TO HIGH ORDER DIGIT
                                        ;OF DAY COUNT.
01AB 1A                 LDAX    D       ;LOAD COUNT.
01AC FE03               CPI     3       ;CHECK IF DAY COUNT IS OVER 300.
01AE 0600               MVI     B,0     ;SET MODULO TO 100.
01B0 C2CC01             JNZ     JULDY   ;IF LESS THAN 300,
                                        ;RESET AT COUNT 100.
01B3 13                 INX     D       ;POINT TO LOW-ORDER DIGITS
                                        ;OF YEAR COUNT.
01B4 1A                 LDAX    D       ;CHECK IF YEAR IS LEAP.
                                        ;YEAR IS LEAP IF DIVISIBLE BY 4.
01B5 0666               MVI     B,66H   ;SET MODULO TO 66,
                                        ;IF YEAR IS NOT LEAP.
01B7 0F                 RRC             ;DIVIDE BY 2.
01B8 DAC901             JC      NLEAP   ;CARRY INDICATES NOT DIVISIBLE.
01BB F6F0               ORI     0F0H    ;IF LEAST SIGNIFICANT 4 BITS,
01BD FEF9               CPI     0F9H    ;ARE GREATER THAN 9,
01BF DAC401             JC      DIV4    ;
01C2 D603               SUI     3       ;DECIMAL ADJUST.
  C4 0F         DIV4:   RRC             ;DIVIDE BY 2, AGAIN.
01C5 DAC901             JC      NLEAP   ;IF CARRY, NOT DIVISIBLE BY 4,
                                        ;AND YEAR IS NOT LEAP.
```

```
01C8 04              INR   B       ;SET MODULO TO 67 FOR LEAP YEAR.
01C9 0E01    NLEAP:  MVI   C,1      ;SET STARTING COUNT TO 1.
01CB 1B              DCX   D       ;POINT BACK TO
01CC 1B      JULDY:  DCX   D       ;TO LOW ORDER DIGITS OF DAY.
01CD CDE101          CALL  DIVIT   ;UPDATE LOW ORDER DIGITS OF DAY.
01D0 0604            MVI   B,4     ;SET MODULO TO 4.
01D2 0E00            MVI   C,0     ;SET STARTING COUNT TO 0.
01D4 CDE101          CALL  DIVIT   ;UPDATE HIGH ORDER DIGIT OF DAY.
01D7 0600            MVI   B,0     ;SET MODULO TO 100.
01D9 CDE101          CALL  DIVIT   ;UPDATE YEAR COUNT.
01DC D1      EXIT:   POP   D       ;RESTORE REGISTERS.
01DD C1              POP   B       ;
01DE F1              POP   PSW     ;RESTORE STATUS.
01DF FB              EI            ;ENABLE FURTHER INTERRUPTS.
01E0 C9              RET           ;RETURN TO INTERRUPTED ROUTINE.
             DIVIT:                ;VARIABLE MODULO COUNTER.
01E1 1A              LDAX  D       ;LOAD COUNTER TO BE INCREMENTED.
01E2 37              STC           ;
01E3 3F              CMC           ;CLEAR CARRY.
01E4 3C              INR   A       ;INCREMENT COUNT.
01E5 27              DAA           ;DECIMAL ADJUST.
  E6 B8              CMP   B       ;TEST IF RESET COUNT
                                   ;HAS BEEN REACHED.
01E7 C2EB01          JNZ   SAVE    ;IF NOT REACHED,
                                   ;SAVE COUNT AND EXIT.
01EA 79              MOV   A,C     ;RESET TO STARTING COUNT.
01EB 12      SAVE:   STAX  D       ;RETURN COUNT TO MEMORY.
01EC 13              INX   D       ;POINT TO NEXT COUNTER.
01ED C8              RZ            ;IF RESET OCCURED,
                                   ;GO UPDATE NEXT COUNTER.
01EE D1              POP   D       ;IF NOT, ISSUE A DUMMY RETURN.
01EF C3DC01          JMP   EXIT    ;EXIT DIVIT.
             TIMER:                ;HERE TIME IS STORED, IN BCD,
                                   ;AS FOLLOWS:
                                   ;  BYTE     CONTAINS
01F2 00              DB    0       ;    0    HUNDRETHS OF A SECOND
01F3 00              DB    0       ;    1    SECONDS
01F4 00              DB    0       ;    2    MINUTES
01F5 00              DB    0       ;    3    HOURS
01F6 00              DB    0       ;    4    LOW ORDER TWO DIGITS
                                   ;             OF DAY
01F7 00              DB    0       ;    5    HIGH ORDER DIGIT
                                   ;             OF DAY
01F8 00              DB    0       ;    6    LOW ORDER TWO DIGITS
                                   ;             OF YEAR
01F9 00              DB    0       ;    7    HIGH ORDER TWO DIGITS
                                   ;             OF YEAR
  00                 END           ;
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. D25

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

Absorbance Calculation

Function

Calculate absorbances from % transmission data.
Transmission data (I) is in fixed point format (16 bits maximum)
Absorbance is fixed point times 10^4

Required Hardware

None

Required Software

Multiply 16 X 16 to give 16
Divide 16 X 16 to give 16 fraction (decimal at left)
Shift right

Input Parameters

PRZER contains I_o (reference intensity)
PRDAT contains I (sample intensity)
(where I/I_o = % transmission)

Output Results

Registers D & E holds absorbance $X10^4$
in fixed point format

Registers Modified:	Assembler/Compiler Used:
all	Intellec 8 MAC80
RAM Required:	**Programmer:**
8	W. B. Telfair
ROM Required:	**Company:**
192	Wilks Scientific Corp.
Maximum Subroutine Nesting Level:	**Address:** 140 Water Street Box 449, S. Norwalk, Conn.06856
3	

```
                ;REF. NO. D25
                ;PROGRAM TITLE ABSORBANCE CALCULATION
                ;
                ;
                ;
0100             ORG 100H
0100            DEN: DS 2
0102            NUM: DS 3
0105            PRZER: DS 2
0107            PRDAT: DS 2
0109            PRONE: DS 1
010A            PRRES: DS 2
010C            NMB: DS 1
010D            ZERO: DS 2 ;I0
010F            DATA: DS 40 ;I1,I2,I3,ETC.
0137            ABSBC: DS 40 ;A1,A2,A3,ETC.

015F 2A0501      ABSOR: LHLD PRZER ;ABSORBANCE CALCULATION:  A = LOG(I0/I) =
0162 EB         XCHG ;                = 2*LOG(E)*( X+ X**3/3+ X**5/5+... )
0163 2A0701     LHLD PRDAT ;            X = (I0-I)/(I0+I)
 l66 0600       MVI B,0
@168 7C         LSHFT: MOV A,H ;MAKE I0>I>I0/2
0169 17         RAL
016A 92         SUB D
016B F27901     JP LOGIT
016E AF         XRA A
016F 7D         MOV A,L
0170 17         RAL
0171 6F         MOV L,A
0172 7C         MOV A,H
0173 17         RAL
0174 67         MOV H,A
0175 04         INR B
0176 C36801     JMP LSHFT
0179 C5         LOGIT: PUSH B
017A 7B         MOV A,E
017B 95         SUB L
017C 4F         MOV C,A
017D 7A         MOV A,D
017E 9C         SBB H
017F 19         DAD D
0180 220001     SHLD DEN
0183 67         MOV H,A
0184 69         MOV L,C
0185 220201     SHLD NUM
^188 CD3302     CALL DIV ; X
 ⊥8B CD6C02     CALL DAVE
018E EB         XCHG
018F 220A01     SHLD PRRES
```

```
0192 220701    SHLD PRDAT
0195 220201    SHLD NUM
0198 220001    SHLD DEN
019B CD7D02    CALL MULT ; X**2
019E CDBA02    CALL MAVE
01A1 EB        XCHG
01A2 220501    SHLD PRZER
01A5 3E10      MVI A,10H
01A7 320901    STA PRONE
01AA 2A0701    ITERA: LHLD PRDAT
01AD 220201    SHLD NUM
01B0 2A0501    LHLD PRZER
01B3 220001    SHLD DEN
01B6 CD7D02    CALL MULT ; X**(2N+1)
01B9 CDBA02    CALL MAVE
01BC EB        XCHG
01BD 3E00      MVI A,0
01BF BC        CMP H
01C0 C2C701    JNZ FAN
01C3 BD        CMP L
01C4 CAF701    JZ DON ; DONE ITERATION IF ZERO
 1C7 220701    FAN: SHLD PRDAT
01CA 220201    SHLD NUM
01CD 3A0901    LDA PRONE
01D0 C620      ADI 20H
01D2 320901    STA PRONE
01D5 67        MOV H,A
01D6 2E00      MVI L,0
01D8 220001    SHLD DEN
01DB CD3302    CALL DIV ; X**(2N+1)/(2N+1)
01DE CD6C02    CALL DAVE
01E1 0604      MVI B,4
01E3 CD2002    CALL SHFTR ; READJUST DECIMAL
01E6 2A0A01    LHLD PRRES
01E9 19        DAD D
01EA 220A01    SHLD PRRES
01ED 3E00      MVI A,0
01EF BA        CMP D
01F0 C2AA01    JNZ ITERA
01F3 BB        CMP E
01F4 C2AA01    JNZ ITERA ; REPEAT IF X**(2N+1)/(2N+1) # 0
01F7 2A0A01    DON: LHLD PRRES
01FA EB        XCHG
01FB 0602      MVI B,2
01FD CD2002    CALL SHFTR ; READJUST DECIMAL
0200 EB        XCHG
?201 112E16    LXI D,162EH ; LN 2
 204 C1        POP B
0205 AF        XRA A
0206 B8        CMP B
```

```
0207 CA0F02   ADDLP: JZ RESUL
020A 19       DAD D
020B 05       DCR B
020C C30702   JMP ADDLP
020F 220201   RESUL: SHLD NUM
0212 21B887   LXI H,87B8H ; 2*LOG E
0215 220001   SHLD DEN
0218 CD7D02   CALL MULT
021B CDBA02   CALL MAVE
021E 1B       DCX D
021F C9       RET

0220 AF        SHFTR: XRA A
0221 7A       MOV A,D
0222 1F       RAR
0223 57       MOV D,A
0224 7B       MOV A,E
0225 1F       RAR
0226 5F       MOV E,A
0227 05       DCR B
0228 C22002   JNZ SHFTR
 ?2B CE00     ACI 0
022D 5F       MOV E,A
022E 3E00     MVI A,0
0230 8A       ADC D
0231 57       MOV D,A
0232 C9       RET

0233 1610      DIV: MVI D,16 ; 16/16=>FRACTION 1.15
0235 2A0201   LHLD NUM
0238 44       MOV B,H
0239 4D       MOV C,L
023A 2A0001   LHLD DEN
023D D5       PUSH D
023E 110000   LXI D,0
0241 7C       MOV A,H
0242 2F       CMA
0243 67       MOV H,A
0244 7D       MOV A,L
0245 2F       CMA
0246 6F       MOV L,A
0247 E5       DODIV: PUSH H
0248 23       INX H
0249 09       DAD B
024A D25602   JNC NOCAR
^?4E 4D       MOV B,H
 24F EB       MOV C,L
0250 29       XCHG
0251 23       DAD H
              INX H
```

```
0252 EB        XCHG
0253 C35902    JMP GO
0256 EB        NOCAR: XCHG
0257 29        DAD H
0258 EB        XCHG
0259 E1        GO: POP H
025A 7C        MOV A,H
025B 37        STC
025C 1F        RAR
025D 67        MOV H,A
025E 7D        MOV A,L
025F 1F        RAR
0260 6F        MOV L,A
0261 E3        XTHL
0262 25        DCR H
0263 CA6A02    JZ OUTDIV
0266 E3        XTHL
0267 C34702    JMP DODIV
026A C1        OUTDIV: POP B
026B C9        RET

 26C 3E00       DAVE: MVI A,0 ; READJUST DECIMAL TO GIVE 0.16
026E BA        CMP D
026F C27402    JNZ DEC2
0272 BB        CMP E
0273 C8        RZ
0274 1B        DEC2: DCX D
0275 AF        XRA A
0276 7B        MOV A,E
0277 17        RAL
0278 5F        MOV E,A
0279 7A        MOV A,D
027A 17        RAL
027B 57        MOV D,A
027C C9        RET

027D 210301     MULT: LXI H,NUM+1 ; 16X16=>32
0280 46        MOV B,M
0281 3600      MVI M,0
0283 2B        DCX H
0284 4E        MOV C,M
0285 3611      MVI M,17
0287 110000    LXI D,0
028A 78        MULT1: MOV A,B
028B 1F        RAR
028C 47        MOV B,A
^28D 79        MOV A,C
 28E 1F        RAR
028F 4F        MOV C,A
0290 35        DCR M
```

```
0291 CAB402    JZ MULOU
0294 D2A102    JNC MULT2
0297 2D        DCR L
0298 2D        DCR L
0299 7B        MOV A,E
029A 86        ADD M
029B 5F        MOV E,A
029C 2C        INR L
029D 7A        MOV A,D
029E 8E        ADC M
029F 57        MOV D,A
02A0 2C        INR L
02A1 7A        MULT2:    MOV A,D
02A2 1F        RAR
02A3 57        MOV D,A
02A4 7B        MOV A,E
02A5 1F        RAR
02A6 5F        MOV E,A
02A7 23        INX H
02A8 7E        MOV A,M
02A9 1F        RAR
 2AA 77        MOV M,A
02AB 23        INX H
02AC 7E        MOV A,M
02AD 1F        RAR
02AE 77        MOV M,A
02AF 2B        DCX H
02B0 2B        DCX H
02B1 C38A02    JMP MULT1
02B4 23        MULOU: INX H
02B5 7E        MOV A,M
02B6 23        INX H
02B7 6E        MOV L,M
02B8 67        MOV H,A
02B9 C9        RET

02BA 3E80       MAVE: MVI A,80H ; ROUND 32 TO UPPER 16
02BC 84        ADD H
02BD 3E00      MVI A,0
02BF 8B        ADC E
02C0 5F        MOV E,A
02C1 3E00      MVI A,0
02C3 8A        ADC D
02C4 57        MOV D,A
02C5 C9        RET

^2C6 010000     ABCAL: LXI B,0 ; CALCULATE ABSORBANCE FOR SERIES OF DATA (I)
 2C9 C5        LPCAL: PUSH B ;                WITH SAME I0
02CA 2A0D01    LHLD ZERO
02CD 220501    SHLD PRZER
```

```
02D0 210F01    LXI H,DATA
02D3 09        DAD B
02D4 09        DAD B
02D5 5E        MOV E,M
02D6 23        INX H
02D7 56        MOV D,M
02D8 EB        XCHG
02D9 220701    SHLD PRDAT
02DC CD5F01    CALL ABSOR
02DF 213701    LXI H,ABSBC
02E2 C1        POP B
02E3 09        DAD B
02E4 09        DAD B
02E5 73        MOV M,E
02E6 23        INX H
02E7 72        MOV M,D
02E8 03        INX B
02E9 3A0C01    LDA NMB
02EC 91        SUB C
02ED C2C902    JNZ LPCAL
02F0 C9        RET
  300          END
```

Ref. #D26

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

TELEPROCESSING BUFFER ROUTINE

Function

To allow a microprocessor to buffer input/output to a USART to allow for speed differences between receiver/transmitter.

Required Hardware

Intel 8251 USART

Required Software

Intel PL/M compiler

Input Parameters

Accepts input from 8251 and places character into buffer.

Output Results

Transmits character from input buffer.

Registers Modified:	Programmer: William Speary
RAM Required:	Company: SPEARCO
ROM Required: $7C_H$	Address: 19337 Olney Mill Road
Maximum Subroutine Nesting Level: 12_H	City: Olney
Assembler/Compiler Used: PLM VERS 2.0	State: Maryland 20832

```
00001  1
00002  1   /*REF. NO. D25 */
00003  1   /*PROGRAM TITLE TELEPROCESSING BUFFER ROUTINE */
00004  1    /* THIS IS A EXAMPLE ON A BUFFER HANDLING PROGRAM. */
00005  1    /* IT CAN BE USED TO BUFFER INPUT TO AND FROM USARTS. */
00006  1    /* IS DATA COME IN AT A DIFFERENT RATE THEN THE OUTPUT */
00007  1    /* CAN ACCEPT IT, THE PROGRAM WILL BUFFER IT. */
00008  1    /* THE TIME DIFFERENCE BETWEEN THEN INPUT AND OUTPUT */
00009  1    /* CAN BE HANDLED BY VARYING THE SIZE OF 'BUFFER' AND */
00010  1    /* 'BUFLEN'. NOTE: 'BUFLEN' MUST BE ONE (1) LESS THAN */
00011  1    /* THE SIZE OF 'BUFFER' . */
00012  1
00013  1              DECLARE LIT LITERALLY 'LITERALLY', DCL LIT 'DECLARE' ;
00014  1              DCL VAD LIT '021H' ;
00015  1              DCL VAD$IN LIT '020H' ;
00016  1              DCL PROC LIT 'PROCEDURE' ;
00017  1              DCL R$RDY LIT '002H' ;
00018  1              DCL T$RDY LIT '01H' ;
00019  1              DCL BUFLEN LIT '100' ;
00020  1              DCL BUFFER(101) BYTE ;
00021  1              DCL COUNT BYTE ;
00022  1              DCL FREEPTR BYTE, OUTPTR BYTE ;
00023  1
00024  1    /* SET COUNT TO NUMBER OF ITEMS THAT BUFFER CAN HOLD */
00025  1    /* THIS NUMBER WILL BE REDUCED FOR EACH ITEM PLACED INTO */
00026  1    /* BUFFER. IF THE BUFFER SPACE IS EXHAUSED (I.E., DATA */
00027  1    /* IS COMING IN FASTER THAN CAN BE OUTPUTED) THIS PROGRAM */
00028  1    /* HALTS. */
00029  1    /* IF 'COUNT' AND 'BUFLEN' ARE THE SAME THE BUFFER IS */
00030  1    /* EMPTY AND THERE IS NOT DATA TO BE OUTPUTED. */
00031  1
00032  1              COUNT=BUFLEN ;
00033  1
00034  1    /* 'FREEPTR' POINTS TO NEXT FREE SLOT IN BUFFER.*/
00035  1    /* 'OUTPUT' POINTS TO THE NEXT CHARACTER TO BE OUTPUTED. */
00036  1
00037  1              FREEPTR,OUTPTR= 1 ;
00038  1
00039  1    GETSCHAR:   PROC ;
00040  2
00041  2    /*IF THERE IS NO MORE ROOM IN BUFFER 'HALT' */
00042  2
00043  2                  IF (COUNT := COUNT - 1) = 0 THEN
00044  2                  HALT ;
00045  2
00046  2    /* GET CHARACTER AND PUT IT IN BUFFER. */
00047  2
00048  2                  BUFFER(FREEPTR)= INPUT(VAD$IN) ;
00049  2
00050  2    /* UPDATE NEXT BUFFER EMPTY SLOT, IF END OF BUFFER */
00051  2    /* WRAP TO THE BEGINING. */
00052  2
00053  2                  IF (FREEPTR := FREEPTR +1) = BUFLEN THEN
00054  2                  FREEPTR=1 ;
00055  2              END    ;
 0056  1
00057  1
00058  1    PUTSCHAR:   PROC ;
00059  2
00060  2    /* IF THEN LEN OF BUFFER AND THE COUNT OF THE NUMBER */
```

00003000
00004000
00005000
00006000
00007000
00008000
00009000
00010000
00011000
00012000
00013000
00014000
00015000
00016000
00017000
00018000
00019000
00020000
00021000
00022000
00023000
00024000
00025000
00026000
00027000
00028000
00029000
00030000
00031000
00032000
00033000
00034000
00035000
00036000
00037000
00038000
00039000
00040000
00041000
00042000
00043000
00044000
00045000
00046000
00047000
00048000
00049000
00050000
00051000
00052000
00053000
00054000
00055000
00056000
00057000
00058000
00059000

```
00061  2     /* OF FREE SLOTS IN THE BUFFER ARE THE SAME, THEN RETURN */
00062  2     /* CAUSE THERE IS NOTHING TO OUTPUT. */
00063  2
00064  2                   IF (COUNT := COUNT+1)= BUFLEN+1 THEN
00065  2                  DO    ;
00066  2                     COUNT=BUFLEN ;
00067  3                     RETURN ;
00068  3                  END    ;
00069  2
00070  2
00071  2     /* OUPUT BUFFER CHARACTER. */
00072  2     /* IF AT END OF BUFFER, WRAP BACK TO BEGINING. */
00073  2
00074  2                OUTPUT(VADSIN)= BUFFER(OUTPTR) ;
00075  2                  IF (OUTPTR := OUTPTR+1) = BUFLEN THEN
00076  2                  OUTPTR=1 ;
00077  2              END    ;
00078  1
00079  1
00080  1     /* IF THERE IS A CHARACTER IN THE 8251 BUFFER, THEN */
00081  1     /* GO GET IT. */
00082  1
00083  1     LOOP:        IF (INPUT(VAD) AND R$RDY) <> 0 THEN
00084  1                  CALL GET$CHAR ;
00085  1
00086  1     /* IF THEN IS A CHARACTER IN THE BUFFER AND THE 8251 CAN */
00087  1     /* ACCEPT A CHARACTER THEN GO OUTPUT A CHARACTER. */
00088  1
00089  1                  IF COUNT <> BUFLEN THEN
00090  1                    IF (INPUT(VAD) AND T$RDY) <> 0 THEN
 0091  1                    CALL PUT$CHAR ;
00092  1                GO TO LOOP ;
00093  1    EOF
NO PROGRAM ERRORS
```

```
00060000
00061000
00062000
00063000
00064000
00065000
00066000
00067000
00068000
00069000
00070000
00071000
00072000
00073000
00074000
00075000
00076000
00077000
00078000
00079000
00080000
00081000
00082000
00083000
00084000
00085000
00086000
00087000
00088000
00089000
00090000
00091000
00092000
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Dec. 1976
Ref. D28

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

RUN Ø

Function

Under ISIS initiation, loads a binary module into RAM above ISIS then moves the binary module into memory locations determined by assembly (or compilation) that includes ISIS area. Unmodified area of ISIS is erased to minimize risk to diskette integrity.

Required Hardware

MDS with Diskette System

Required Software

ISIS, PL/M Compiler

Input Parameters

Name of binary file that is to be loaded or a defalt file name that is selected before RUN Ø is compiled. Binary module is first loaded with an offset of 4000H. Restarts 1 thru 7 are available to binary module.

Output Results

Binary module program starts executing from entry address it contains. Interrupt Ø can still be used for returning to MONITOR. Need to reboot system to use ISIS again.

Registers Modified:	**Assembler/Compiler Used:** PL/M
RAM Required:	**Programmer:** Victor H. Grinich
ROM Required:	**Company:** Stanford University
Maximum Subroutine Nesting Level:	**Address:** 26346 Esperanza Dr. Los Altos Hills, CA. 94022

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Dec. 1976

Ref. D29

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

STAGE 2

Function

STAGE 2 is a language independent macro processor developed by Prof. William M. Waite of the University of Colorado. It is primarily intended to be used to implement machine independent software by means of abstract machine modeling, but it is suitable for use as a prepass for any language translator for purposes of providing macro capability.

Required Hardware

MDS 800 Intellec Microcomputer Development System with console device diskette drive(s) and controller, at least 48K RAM memory.

Required Software

ISIS-II Operating System

Input Parameters

Channel assignments equivalencing STAGE 2 channels with ISIS-II files and devices. This implementation of STAGE 2 supports nine non-dummy I/O channels.

Output Results

User defined.

Program offered on diskette only.

Registers Modified:	**Assembler/Compiler Used:** PL/M 80
RAM Required: 48K minimum	**Programmer:** Bruce W. Ravenel
ROM Required:	**Company:** Intel - MCD Software Development
Maximum Subroutine Nesting Level:	**Address:** 3065 Bowers Ave. Santa Clara, Ca. 95051

98-0348

Ref. # D30

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

PAPER TAPE LEADER I.D.

Function

This program allows up to a 72 character message at one time to be punched out on a paper tape punch. This makes possible a permanent identification of the program or data to be punched on the paper tape in lieu of writing on, or affixing labels to, the paper tape leader.

The program uses the PO, CO and CI routines of the MDS monitor. The program starts a 300H. After the operator inputs the G300 command, the program outputs the following message:

INPUT UP TO 72 CHAR. - END W/C.R.;

and then waits for the message to be input. The length of the allowed message can be changed by changing the BUFSIZ a quote statement. If more than the allowed 72 character message is required, the program can be rerun as many times as required.

Upon receipt of a carriage return, or an attempt to input more than 72 characters, the program takes each stored message character, accesses the table to convert the character to a string of ASCII characters, outputs the string to the punch and punches the string to form a readable alphanumeric character or symbol. When the buffer is empty, the program returns control to the MDS monitor.

Registers Modified: N/A	Programmer: Ken Paul MS*14
RAM Required: 1K	Company: Control Data Corporation
ROM Required: MDS Monitor (2K)	Address: 11615 I Street
Maximum Subroutine Nesting Level: N/A	City: Omaha
Assembler/Compiler Used: 8080 MDS Macro Assembler V1.0	State: Nebraska 68137

Ref. # D30

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

PAPER TAPE LEADER I.D.

Function

(see page 1)

Required Hardware

Hardware required is the standard MDS-800 with TTY (with tape reader and punch) connected to the MDS TTY port connector.

Required Software

MDS Monitor

Input Parameters

Read in program. Give starting address command (G300) and input message on console device.

Output Results

Readable alphanumerics and symbols punched into paper tape in paper tape punch.

Registers Modified: N/A	Programmer: Ken Paul MS*14
RAM Required: 1K	Company: Control Data Corporation
ROM Required: MDS Monitor (2K)	Address: 11615 I Street
Maximum Subroutine Nesting Level: N/A	City: Omaha
Assembler/Compiler Used: 8080 MDS Macro Assembler V1.0	State: Nebraska 68137

Ref. # D31

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title: LSORT

Function: This subroutine sorts an ASCII file in alphabetical order. The file is in memory and is terminated with a control Z (1AH) character. Each line, or record, is terminated with a LINE FEED (0AH).

Required Hardware: None

Required Software: None

Input Parameters: HL = Address of first character in file.

Output Results: Each record in the file is in alphabetical order.

Registers Modified: ALL	Programmer: Dick Springer
RAM Required: III HEX	Company: General Microwave Corp.
ROM Required: none	Address: 155 Marine Street
Maximum Subroutine Nesting Level: 2	City: Farmingdale
Assembler/Compiler Used:	State: New York 11735

```
                        ;REF. NO. D31
                        ;PROGRAM TITLE LSORT
                        ;
                        ;
                        ;
                        ;LINE SORT ROUTINE
                        ;
                        ;THIS ROUTINE ARRANGES THE LINES OF AN ASCII FILE
                        ;IN ALPHABETICAL ORDER.  THE FILE IS IN MEMORY.
                        ;EACH LINE MUST BE TERMINATED WITH A LINE FEED (0AH)
                        ;AND THE FILE MUST BE TERMINATED WITH A CONTROL-Z (1AH).
                        ;THE CONTROL-Z MUST BE IMMEDIATELY PRECEDED BY A LINE
                        ;FEED.  THE INITIAL ADDRESS OF THE FILE MUST BE IN
                        ;REGISTER PAIR HL.  THE MAXIMUM PERMISSIBLE LENGTH OF
                        ;A LINE IS 96 (60H) CHARACTERS.

000A                    LF      EQU     0AH
001A                    CTLZ    EQU     1AH

                        LSORT:
0000 22AB00                     SHLD    SPB
0003 22AF00                     SHLD    SPD
0006 CD5300                     CALL    LOAD
0009 CD9300                     CALL    COMP
000C C21500                     JNZ     $+9
000F DA2900                     JC      ENDG
0012 C30900                     JMP     LSORT+9
0015 DA1E00                     JC      $+9
0018 2AAD00                     LHLD    SPC
001B C30300                     JMP     LSORT+3
001E 2B                         DCX     H
001F 7E                         MOV     A,M
0020 FE0A                       CPI     LF
0022 23                         INX     H
0023 C21F00                     JNZ     $-4
0026 C30900                     JMP     LSORT+9

                        ENDG:
0029 22A900                     SHLD    SPA
002C 2AAF00                     LHLD    SPD
002F CD7F00                     CALL    CLOSE
0032 2AA900                     LHLD    SPA
0035 CD6000                     CALL    OPEN
0038 2AAB00                     LHLD    SPB
003B EB                         XCHG
003C 22AB00                     SHLD    SPB
003F 22AF00                     SHLD    SPD
0042 216000                     LXI     H,60H
0045 19                         DAD     D
0046 CD8200                     CALL    CLOSE+3
```

```
0049 2AAB00          LHLD    SPB
004C 3E1A            MVI     A,1AH
004E BE              CMP     M
004F CA0600          JZ      LSORT+6
0052 C9              RET

                LOAD:
0053 11B100          LXI     D,IBUF
0056 7E              MOV     A,M
0057 12              STAX    D
0058 13              INX     D
0059 23              INX     H
005A FE0A            CPI     LF
005C C25C00          JNZ     $
005F C9              RET

                OPEN:
0060 EB              XCHG
0061 216000          LXI     H,60H
0064 19              DAD     D
0065 44              MOV     B,H
0066 4D              MOV     C,L
0067 2AAB00          LHLD    SPB
006A 1A              LDAX    D
006B 02              STAX    B
006C 7C              MOV     A,H
006D BA              CMP     D
006E DA7300          JC      $+5
0071 7D              MOV     A,L
0072 BB              CMP     E
0073 1B              DCX     D
0074 0B              DCX     B
0075 DA6A00          JC      $-11
0078 13              INX     D
0079 21B100          LXI     H,IBUF
007C C35600          JMP     LOAD+3

               CLOSE:
007F CD5300          CALL    LOAD
0082 EB              XCHG
0083 2AAF00          LHLD    SPD
0086 1A              LDAX    D
0087 77              MOV     M,A
0088 FE1A            CPI     CTLZ
008A 23              INX     H
008B 13              INX     D
008C C28600          JNZ     $-6
008F 2AAF00          LHLD    SPD
0092 C9              RET
```

```
                COMP:
0093 22AD00             SHLD    SPC
0096 11B100             LXI     D,IBUF
0099 3E1A               MVI     A,CTLZ
009B BE                 CMP     M
009C 37                 STC
009D C8                 RZ
009E 1A                 LDAX    D
009F BE                 CMP     M
00A0 23                 INX     H
00A1 13                 INX     D
00A2 C0                 RNZ
00A3 FE0A               CPI     LF
00A5 C29900             JNZ     $-12
00A8 C9                 RET

00A9 0000       SPA:    DW      0
00AB 0000       SPB:    DW      0
00AD 0000       SPC:    DW      0
00AF 0000       SPD:    DW      0
00B1            IBUF:   DS      60H

0000                    END
```

Ref. # D33

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

OCTHEX

Function

Prints a table of the octal numbers 0 - 377 along with their hexadecimal equivalents

Required Hardware

TTY as device 8.

Required Software

None. This program includes a TTY print routine as well as binary to octal and binary to hexadecimal subroutines.

Input Parameters

None.

Output Results

The printed table of octal and hexadecimal numbers.

Program available in source listing only.

Registers Modified: N/A	Programmer: L. Leipuner
RAM Required: None	Company: Brookhaven National Laboratory
ROM Required: 105.	Address: Building 510
Maximum Subroutine Nesting Level: 3	City: Upton
Assembler/Compiler Used: PDP11 Macro's	State: New York 11973

Ref. # D34

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

SDK-80 PAPER TAPE PUNCH ROUTINE

Function

The program outputs the contents of a specified area of memory to paper tape in a format compatible with the SDK-80 'I' command. It may be used to store partially or fully developed programs in machine readable form. Tape produced by this program may be read into memory by the SDK-80 Monitor 'I' (Insert into memory) command.

Required Hardware

Standard SDK-80 board. The author's system has 512 bytes of RAM and uses an ASR 33 as console device.

Required Software

The program uses a number of SDK-80 Monitor subroutines; these are listed below, together with their start addresses:

GETNM	Ø257	CI	Ø1DØ
NMOUT	Ø2C3	HILO	Ø29C
CO	Ø1E3		

Input Parameters

Program execution is initiated via the system monitor using the G command. The program will immediately output CRLF to the console and wait for the operator to input the begin and end addresses of the memory to be punched. The addresses are input in the format

BBBB, EEEE CR

where B and E represent hexadecimal begin and end addresses respectively. SDK-80 Monitor conventions are followed regarding delimiters and addresses containing other than four digits. Input of a non-hex character will return control to the Monitor.

Output Results

When the addresses have been input the program waits for the operator to switch on the tape punch. The operator indicates that the punch is on by inputing any character to the console; the character is not echoed.
The program punches 32 spaces, CR, the contents of the indicated memory area (as ASC II coded hex digits), CR, 32 spaces, CR. Control is then returned to the monitor, which responds with CRLF and outputs the message MCS 80 KIT, followed by the prompt character. The tape should be cut just before the CRLF output by the Monitor.
The Monitor I command is used to read the tape into memory. The sequence is IXXXX CR followed by tape reader on (XXXX is the start address for the

Registers Modified:	Assembler/Compiler Used:
All	None
RAM Required:	**Programmer:**
~ 50 bytes + stack	Steve Richards
ROM Required:	**Company:**
--	VORTEK Industries Ltd.
Maximum Subroutine Nesting Level:	**Address:**
3	1820 Pandora St., Van., B.C., V5L 1M5

INPUT PARAMETERS (Cont'd)

memory insertion). When the tape stops the Escape character is input to the console to terminate the I command.
An example of the use of the program to punch a segment of the System Monitor to tape and read it back into RAM is included on a separate sheet.

```
                ;REF. NO. D34
                ;PROGRAM TITLE SDK-80 PAPER TAPE PUNCH ROUTINE
                ;
                ;
                ;
                ;
1200                    ORG     1200H
0257            GETNM   EQU     0257H
02C3            NMOUT   EQU     02C3H
01E3            CO      EQU     01E3H
01D0            CI      EQU     01D0H
029C            HILO    EQU     029CH


1200 0E02       TPNCH:  MVI     C,02H
1202 CD5702             CALL    GETNM
1205 D1                 POP D
1206 E1                 POP H
1207 CDD001             CALL    CI
120A CD1F12             CALL    SPCES
120D 7E         TP10:   MOV     A,M
 ?0E CDC302             CALL    NMOUT
1211 CD9C02             CALL    HILO
1214 DA1B12             JC      TP15
1217 23                 INX     H
1218 C30D12             JMP     TP10
121B CD1F12     TP15:   CALL    SPCES
121E CF                 RST     1
121F CD2C12     SPCES:  CALL    LNFN
1222 0620               MVI     B,20H
1224 48                 MOV     C,B
1225 CDE301     SP05:   CALL    CO
1228 05                 DCR     B
1229 C22512             JNZ     SP05
122C 0E0D       LNFN:   MVI     C,0DH
122E CDE301             CALL    CO
1231 C9                 RET
0000            END
```

Ref. # D35

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

Type K T.C. Linearizer (Procedure)

Function

Store the binary equivalent (1 Byte) of temperature in degrees Fahrenheit for a full scale of 1000°F referenced to ice point.* (* Using electronic compensation).

Required Hardware

Intellect 8/Mod 80 or any 8080 based Microprocessor

Required Software

PLM Compiler on Host Machine

Input Parameters

Two: Raw$Temp = Measured Junction (0.091 MV per Bit)
Ref$Temp = Reference Junction Compensation (0.091 MV per Bit)

Output Results

One: Comp$Temp = Binary Equivalent to Measured Temperature From 0° to 1000°F With Transfere Function Equal to 4.07°F Per Bit.

Registers Modified: A, B, C, D, E, H, L, & SP	Programmer: O. Leon Lindsey, Sr. Research Eng.
RAM Required: 5 Bytes (Variables) + 6 Bytes (Stack) = 11	Company: Solar, Div. of I.H.
ROM Required: 111 Bytes	Address: 2200 Pacific Highway
Maximum Subroutine Nesting Level: Six	City: San Diego
Assembler/Compiler Used: PLM	State: California 92138

```
  TCK.LPT
00001  1
00002  1       /*HOUSEKEEPING ENTRIES */
00003  1    DECLARE DCL LITERALLY 'DECLARE';
00004  1    DCL LIT LITERALLY 'LITERALLY';
00005  1    DCL PRC LIT 'PROCEDURE';
00006  1    DCL ADR LIT 'ADDRESS';
00007  1
00008  1      /* END OF HOUSEKEEPING */
00009  1    /* GLOBAL VARIABLES */
00010  1     DCL RAW$TEMP BYTE; /*BINARY INPUT = TO MEASURED JUNCTION
00011  1                        GAIN EQUAL 0.091 MV PER BIT*/
00012  1     DCL REF$TEMP BYTE; /* BINARY INPUT = TO REF JUNCT COMPEN
00013  1                        SATION GAIN EQUAL 0.091 MV PER BIT */
00014  1     DCL COMP$TEMP BYTE; /* OUTPUT BINARY = TO MEASURED TEMPER
00015  1                       ATURE   GAIN EQUAL 4.07 DEG PER BIT */
00016  1
00017  1
00018  1
00019  1    COMP$TEMP$GET:PRC;    /*PROCEDURE FROM LINE 190 TO 270 */
00020  2     DCL (MODTEMP, TEMPBY25) BYTE;
00021  2    DO;
00022  2      MODTEMP = RAW$TEMP + REF$TEMP;
00023  3      TEMPBY25 = (255 - MODTEMP) / 25 ;
00024  3      IF TEMPBY25 < 7 THEN COMPTEMP = TEMPBY25 +1 +  MODTEMP;
00025  3      ELSE COMPTEMP = MODTEMP  + 7;
00026  3    END;
00027  2    END COMP$TEMP$GET;
00028  1
00029  1    START: DO;    /* SHORT ROUTINE TO EXERCISE PROCEDURE */
00030  1             CALL COMP$TEMP$GET;
00031  2             OUTPUT (1) = COMPTEMP;
00032  2            END;
00033  1     EOF
NO PROGRAM ERRORS

READY

?V
?
```

Ref. # D36

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title
Enable Hold - Screen Mode

Function
Enable Hold-Screen-Mode of DEC Scope Model VT-52

Required Hardware
MDS 800 with DEC Scope Model VT-52

Required Software
No additional software required.

Input Parameters
None

Output Results
Enables Hold-Screen-Mode of DEC Scope Model VT-52

Registers Modified:	Programmer: D. Brenneman
RAM Required:	Company: General Electric
ROM Required:	Address:
Maximum Subroutine Nesting Level:	City: Waynesboro
Assembler/Compiler Used: ISIS 8080 MACRO Assembler V1.1	State: VA 22980

```
                ;REF. NO. D36
                ;PROGRAM TITLE ENHSM
                ;
                ;
                TITLE' ENHSM 1 5 77 DB '

                ; THIS WILL ENABLE THE DECSCOPE HOLD-SCREEN-MODE

00F6            CRTDP   EQU     0F6H    ;CRT DATA PORT
00F7            CRTS    EQU     0F7H    ;CRT STATUS PORT
0001            TBR     EQU     1       ;TRANS BUFFER READY
001B            ESC     EQU     1BH     ;ESCAPE KEY
0009            EXIT    EQU     9
0040            ISIS    EQU     40H

F500                    ORG     0F500H

F500 3131F5     START:  LXI     SP,STACK+12

F503 0E1B               MVI     C,ESC
F505 CD1AF5             CALL    CRTOU
F508 0E5B               MVI     C,'['   ;EN H-S-M
F50A CD1AF5             CALL    CRTOU

F50D 21FEFF             LXI     H,-2    ; SP=K
F510 39                 DAD     SP      ; HL=K-2
F511 E5                 PUSH    H       ; SP=K-2,(TOS)=K-2(STATUS)
F512 EB                 XCHG            ; DE=K-2
F513 0E09               MVI     C,EXIT
F515 CD4000             CALL    ISIS    ; RETURN CONTROL TO ISIS
F518 FB                 EI              ;(BACK HERE IF ISIS.CLI
F519 76                 HLT             ;  HAS BEEN DESTORYED)


F51A DBF7       CRTOU:  IN      CRTS
F51C E601               ANI     TBR
F51E CA1AF5             JZ      CRTOU   ; LOOP TILL READY
F521 79                 MOV     A,C
F522 D3F6               OUT     CRTDP
F524 C9                 RET

F525 25F5       STACK:  DW      $

F500                    END     START
```

Ref. # D37

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title
Disable Hold - Screen Mode

Function
Disable Hold-Screen-Mode of DEC Scope Model VT-52

Required Hardware
MDS 800 (with DOS) and DEC Scope

Required Software
No Additional software required

Input Parameters
None

Output Results
Disable Hold-Screen-Mode of DEC Scope

Registers Modified:	Programmer: D. Brenneman
RAM Required:	Company: General Electric
ROM Required:	Address:
Maximum Subroutine Nesting Level:	City: Waynesboro
Assembler/Compiler Used: ISIS 8080 MACRO Assembler V1.1	State: VA 22980

```
I_ 3 8080 MACRO ASSEMBLER, V1.1                          PAGE 1
 DIHSM 1 5 77 DB


                ;REF.  NO.  D37
                ;PROGRAM TITLE DIHSM
                ;
                ;
                ;
                TITLE' DIHSM 1 5 77 DB '

                ;  THIS WILL DISABLE THE DECSCOPE HOLD-SCREEN-MODE

 00F6           CRTDP   EQU     0F6H    ;CRT DATA PORT
 00F7           CRTS    EQU     0F7H    ;CRT STATUS PORT
 0001           TBR     EQU     1       ;TRANS BUFFER READY
 001B           ESC     EQU     1BH     ;ESCAPE KEY
 0009           EXIT    EQU     9
 0040           ISIS    EQU     40H

 F500                   ORG     0F500H

 F500 3131F5    START:  LXI     SP,STACK+12

 F503 0E1B              MVI     C,ESC
   505 CD1AF5           CALL    CRTOU
 F508 0E5C              MVI     C,'\'   ;DI H-S-M
 F50A CD1AF5            CALL    CRTOU

 F50D 21FEFF            LXI     H,-2    ; SP=K
 F510 39                DAD     SP      ; HL=K-2
 F511 E5                PUSH    H       ; SP=K-2,(TOS)=K-2(STATUS)
 F512 EB                XCHG            ; DE=K-2
 F513 0E09              MVI     C,EXIT
 F515 CD4000            CALL    ISIS    ; RETURN CONTROL TO ISIS
 F518 FB                EI              ; (BACK HERE IF ISIS.CLI
 F519 76                HLT             ;  HAS BEEN DESTORYED)


 F51A DBF7      CRTOU:  IN      CRTS
 F51C E601              ANI     TBR
 F51E CA1AF5            JZ      CRTOU   ; LOOP TILL READY
 F521 79                MOV     A,C
 F522 D3F6              OUT     CRTDP
 F524 C9                RET

 F525 25F5      STACK:  DW      $

 F500                   END     START
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

THERMOCOUPLE LINEARIZATION (TYPE J)

Function

PROGRAM WILL CONVERT MILLIVOLT VALUES FROM A TYPE J THERMOCOUPLE TO DEGREES CENTIGRADE. (REFERENCE JUNCTION Ø DEGREES CENTIGRADE.) RANGE FROM -220 DEG. C. TO 300 DEG. C.

Required Hardware

00B8H BYTES PROM/ROM FOR BASIC PROGRAM. 017DH BYTES INCLUDES ALL MATH ROUTINES REQUIRED (SIGNED 16 BIT DIVIDE, SIGNED 16 BIT MULTIPLY, ETC.) NO RAM IS REQUIRED.

Required Software

MAY USE OWN MATH PACKAGE IF COMPATIBLE.

Input Parameters

REGISTER PAIR D & E CONTAIN MILLIVOLT READINGS TIMES 1000.

Output Results

REGISTER PAIR H & L RETURN WITH DEGREES CENTIGRADE TIMES 10. ALL OTHER REGISTERS ARE MODIFIED. THERMOCOUPLE TABLE IS IN 10 DEG. C. INCREMENTS. LINEAR INTERPOLATION IS PERFORMED ON VALUES THAT FALL BETWEEN INCREMENTS. DIFFERENT THERMOCOUPLE TYPES MAY BE USED BY REPLACING THE TYPE J TABLE WITH THE DESIRED MILLIVOLT TABLE FROM N. B. S. REFERENCE TABLES.

Registers Modified: ALL	Programmer: PETER C. SCHMITT
RAM Required: NONE	Company: OMICRON, INC.
ROM Required: 017DH	Address: 1111 E. TEN MILE RD
Maximum Subroutine Nesting Level: ONE	City: MADISON HEIGHTS
Assembler/Compiler Used: PCS 8080 ASSEMBLER	State: MICHIGAN 48071

Ref.# D39

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

INVERT Data in RAM

Function

Invert Data in RAM

Required Hardware

INTELLECT 8/Mod 80

Required Software

INTELLECT 8/Mod 80 MONITOR (1975)

Input Parameters

- I
- ADDRESS (Hex) of Beginning
- , (comma)
- ADDRESS (Hex) of Finish
- CR

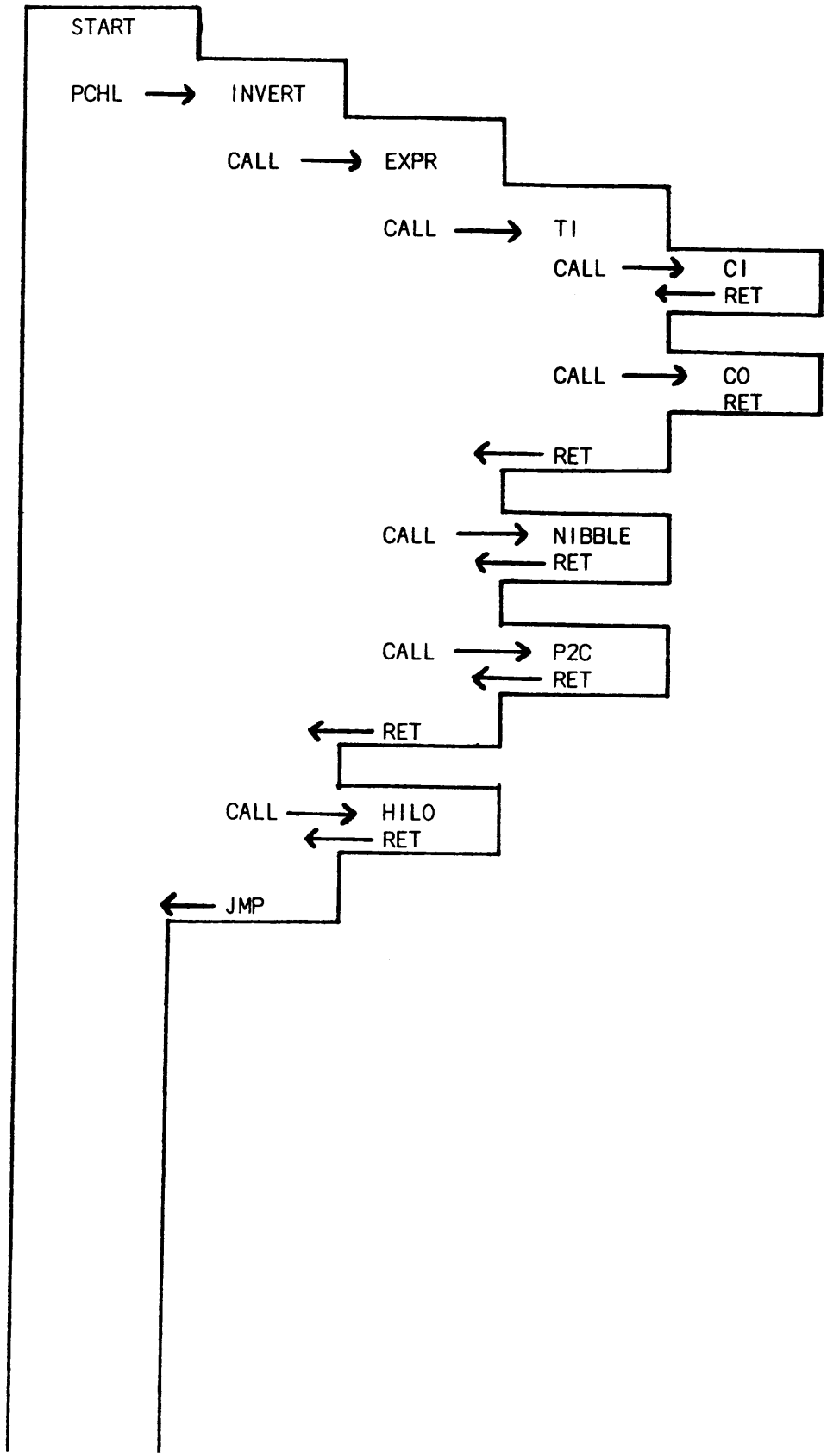
Output Results

Contents of RAM (from beginning thru finish) is complemented.

Registers Modified:	Programmer: D. Brenneman
RAM Required:	Company: General Electric
ROM Required:	Address: GE Drive
Maximum Subroutine Nesting Level:	City: Waynesboro
Assembler/Compiler Used:	State: Virginia 22980

6/8/77

98-034C


START
PCHL
INVERT
CALL
EXPR
CALL
TI
CALL
CI
RET
CALL
CO
RET
RET
CALL
NIBBLE
RET
CALL
P2C
RET
RET
CALL
HILO
RET
JMP

```
                ;REF. NO. D39
                ;PROGRAM TITLE INVERT
                ;
                ;
                ;
                ;
                TITLE ' INVERT  1/18/77  DB. '

                ; THIS IS DESIGNED TO BE A MODIFICATION TO THE
                ; INTELLEC8/MOD 80 MONITOR, VERSION 3.0, 14 APRIL 1975
                ;
                ; IT WILL COMPLEMENT THE DATA IN (RAM) MEMORY
                ; FROM BEGINNING ADDRESS THRU ENDING ADDRESS
                ; AND ISSUE A PROMPT CHARACTER WHEN FINISHED.
                ; IT IS INVOKED BY TYPING
                ; IBEGINNING ADDRESS,ENDING ADDRESS(CARRAIGE RETURN)
                ;
                ; IT WILL FIT INTO THE EPROMS USED FOR THE MONITOR
                ; BY CHANGING TWO EPROMS (3800H AND 3F00H)
                ;
                ;
                ;
                ;  EPROM 3800H CHANGES: 38A2H FROM 43H TO E4H
                ;                       38A3H FROM 3CH TO 3FH
                ;
                ;  EPROM 3F00H ADDITIONS: 3FE4H THRU 3FF4H ARE AS FOLLOWS

  3FE4                  ORG     3FE4H

  3FE4 CD703D           CALL    3D70H   ;EVALUATE EXPRESSION
  3FE7 D1               POP     D       ;FINISH ADDRESS
  3FE8 E1               POP     H       ;BEGINNING ADDRESS

  3FE9 7E               MOV     A,M     ;GET DATA
  3FEA 2F               CMA             ;INVERT DATA
  3FEB 77               MOV     M,A     ;RETURN DATA
  3FEC CDA43D           CALL    3DA4H   ;COMPARE HL WITH DE (AND INX H)
  3FEF D2E93F           JNC     $-6     ;LOOP TILL HL=DE
  3FF2 C36B38           JMP     386BH   ;START - ISSUE PROMPT WHEN DONE

  0000          END
```

Ref.# D40

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

RELATIVE JUMP ROUTINE

Function

This routine implements a two-byte relative jump instruction for saving memory in a large program. Three-byte unconditional jumps may be replaced with a two-byte sequence if the destination address lies within PC-127 and PC+128. Any available restart vector may be used.

Required Hardware

Required Software

THE CALLING SEQUENCE IS AS FOLLOWS:

```
RST X  ; RESTART TO RELATIVE JUMP ROUTINE
DB  N  ; ADDRESS OFFSET
```

Input Parameters

OR, WITH THE INCLUDED ADDRESS CONVERSION MACRO:

```
JR  ADDR  ;  JR CONVERTS ADDR TO OFFSET
```

Output Results

COMMENTS:

Stack depth increases by 6 bytes. Execution time for a relative jump is 144 microseconds compared with 10 microseconds for a regular jump. (clock frequency = 1.0 MHZ)

Registers Modified: NONE	**Programmer:** Bernard J. Verreau
RAM Required: NONE	**Company:** National Cash Register
ROM Required: 16 bytes	**Address:** P.O. Box 607
Maximum Subroutine Nesting Level: STACK USAGE: 6 bytes	**City:** Millsboro
Assembler/Compiler Used: ISIS 8080 MACRO ASSEMBLER, V1.1	**State:** DELAWARE 19963

```
                ;REF. NO. D40
                ;PROGRAM TITLE JR REATIVE JUMP ROUTINE
                ;
                ;
                ;
                        TITLE   'RELATIVE JUMP ROUTINE'
                ;
                ; NAME:  JR      B. J. VERREAU     N C R  CORPORATION      04/11/77
                ;
                ; RELATIVE JUMP ROUTINE
                ;
                ; ENTRY:
                ;       CALLED WITH AN 'RST 7' INSTRUCTION.  THE BYTE
                ;       FOLLOWING THE RESTART IS THE ADDRESS OFFSET.
                ;
                ;               EG:     RST     7       ; JUMP RELATIVE TO 'EG'
                ;                       DB      -1
                ;
                ; EXIT:
                ;       THE PROGRAM COUNTER IS SET EQUAL TO THE TRANSFER
                ;       ADDRESS IF IT LIES WITHIN THE RANGE (PC-127,PC+128).
                ;
                ; REGISTER USAGE:
                ;       A -
                ;       B -             C -
                ;       D -             E -
                ;       H -             L -
                ;       CARRY -         ZERO -
                ;       SIGN -          PARITY -
                ;       SP -            PC - S
                ;
                ; STACK USAGE:  6       LENGTH:  15
                ;
                JR      MACRO   ADDR    ; ADDRESS CONVERSION MACRO
                        RST     7
                        DB      (ADDR-$) AND 0FFH
                        ENDM
0038                    ORG     0038H   ; * RELATIVE JUMP ROUTINE *
0038 E3                 XTHL            ; FETCH OFFSET POINTER
0039 F5                 PUSH    PSW     ; STACK WORKING REGISTERS
003A D5                 PUSH    D
003B AF                 XRA     A
003C 57                 MOV     D,A     ; SET UP ADDRESS OFFSET
003D 5E                 MOV     E,M
003E B3                 ORA     E
003F F24300             JP      FORWD   ; BRANCH IF OFFSET POSITIVE
0042 15                 DCR     D       ; COMPLEMENT HI ADDRESS
0043 19         FORWD:  DAD     D
0044 D1                 POP     D       ; UNSTACK REGISTERS
0045 F1                 POP     PSW
```

```
0046 E3                 XTHL            ; STORE NEW RETURN ADDRESS
0047 C9                 RET
                ;
                ; TEST PROGRAM
                ;
0048 FF         TEST:   RST     7       ; JUMP TO NEXT INSTRUCTION
0049 01                 DB      1
               +        JR      LAB1    ; JUMP TO NEXT INSTRUCTION
004A FF        +        RST     7
004B 01        +        DB      (0004CH-$) AND 0FFH

               +LAB1:   JR      LAB3    ; JUMP TO LAST INSTRUCTION
004C FF        +        RST     7
004D 03        +        DB      (00050H-$) AND 0FFH

004E FF         LAB2:   RST     7       ; JUMP TO FIRST INSTRUCTION
004F F9                 DB      -7
               +LAB3:   JR      LAB2    ; JUMP TO PREVIOUS INSTRUCTION
0050 FF        +        RST     7
0051 FD        +        DB      (0004EH-$) AND 0FFH

0048                    END     TEST
```

Ref.# D41

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

JULIAN DATE ROUTINE

Function

Check validity of, and/or convert to Julian, dates from January 1, 1901 and December 31, 2099,

Required Hardware

8080 System

Required Software

Driver Routine

Input Parameters

H, L contains address of ASCII MMDDYY

Output Results

D, E contains Julain Date (integer)
Carry set if and only if invalid date

Registers Modified: D, E, PSW, A	Programmer: Leonard J. Jowers
RAM Required: Ø7BH	Company: Sullivan, Long & Hagerty
ROM Required: Ø	Address: P.O. Box 2247
Maximum Subroutine Nesting Level: 3 levels	City: Birmingham,
Assembler/Compiler Used: ISIS 8080 MACRO ASSEMBLER	State: AL 35201

6/8/77

```
                        ;REF. NO. D41
                        ;PROGRAM TITLE JULIAN DATE ROUTINE
                        ;
                        ;
                        ;
                        ;----------------------------------------
                        ;-------          JULIAN DATE
                        ;
                        ;       H,L ----- ASCII MMDDYY ADDR
                        ;       D,E <---- JULIAN DATE
                        ;       CARRY IFF ERROR
                        ;
0000 E5                 JULIAN: PUSH H;
0001 C5                         PUSH B;
0002 CD4700                     CALL JS1;       CARRY IFF ERROR
0005 57                         MOV D,A;        MONTH
0006 D44700                     CNC JS1;
0009 5F                         MOV E,A;        DAY
000A D44700                     CNC JS1;
000D DA4400                     JC JERR;        YEAR
0010 216300                     LXI H,JULTBL+1;
 113 0600                       MVI B,0;
0015 4A                         MOV C,D;
0016 09                         DAD B;
0017 09                         DAD B;
0018 46                         MOV B,M;
0019 2B                         DCX H;
001A 4E                         MOV C,M;
001B 2B                         DCX H;
001C C5                         PUSH B; PRELIM MAXDAY
001D 46                         MOV B,M;
001E 2B                         DCX H;
001F 4E                         MOV C,M;
0020 6B                         MOV L,E;
0021 2600                       MVI H,0;
0023 09                         DAD B;  PRELIM JULIAN DATE
0024 C1                         POP B;
0025 E603                       ANI 011B;       YEAR(MOD 4)
0027 7A                         MOV A,D;
0028 C23500                     JNZ J100;
002B FE02                       CPI 2;          FEB?
002D DA3500                     JC J100;
0030 CA3400                     JZ J000;
0033 23                         INX H;  MAR-DEC(INCR JULIAN DATE
0034 03                 J000:   INX B;  FEB-DEC(INCR MAXDAY)
0035 FE0D               J100:   CPI 13; MM<13?
^937 D24300                     JNC JERX;
 J3A EB                         XCHG;
003B 7A                         MOV A,D;
003C 2F                         CMA;
```

```
003D 67                 MOV H,A;
003E 7B                 MOV A,E;
003F 2F                 CMA;
0040 6F                 MOV L,A;
0041 23                 INX H;
0042 09                 DAD B;
0043 3F         JERX:   CMC;
0044 C1         JERR:   POP B;
0045 E1                 POP H;
0046 C9                 RET;
                ;-------
0047 CD5900     JS1:    CALL JS2;
004A D8                 RC;
004B 07                 RLC;
004C 47                 MOV B,A;
004D 07                 RLC;
004E 07                 RLC;
004F 80                 ADD B;
0050 47                 MOV B,A;
0051 CD5900             CALL JS2;
0054 D8                 RC;
 )55 80                 ADD B;
0056 FE01               CPI 1;
0058 C9                 RET;
                ;-------
0059 7E         JS2:    MOV A,M;
005A 23                 INX H;
005B D630               SUI '0';
005D D8                 RC;
005E FE0A               CPI 10;
0060 3F                 CMC;
0061 C9                 RET;
                ;-------
0062 00001F00 JULTBL: DW 000,031,059,090,120,151;       JAN-JUL
0066 3B005A00
006A 78009700
006E B500D400           DW 181,212,243,273,304,334,365; JUL-DEC,JAN
0072 F3001101
0076 30014E01
007A 6D01
0000             END;
```

insite T.M. **INTEL® USER'S LIBRARY SUBMITTAL FORM**

Ref.# D42

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

TERMINET* 1200

Function

Software Interface for Terminet* 1200

Required Hardware

MDS-800 & TERMINET* 1200 with Punch (Note: Hardware modification required to get MDS to operate at 1200 Baud)

Required Software

No additional software required

Input Parameters

None

Output Results

1. Provides necessary fill characters after Line-Feed
2. Provides necessary time delay after Form-Feed
3. Provides necessary time delays for paper punch to operate

*Req T.M.

Registers Modified:	Programmer: D. Brenneman
RAM Required:	Company: General Electric
ROM Required:	Address: GE Drive
Maximum Subroutine Nesting Level:	City: Waynesboro
Assembler/Compiler Used: ISIS 8080 MACRO ASSEMBLER V1.1	State: VA 22980

Ref:# D43

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

TERMINET* 300.

Function

Provide interface between Terminet* 300 (at 300 Band) and MDS800

Required Hardware

No additional (however, the MDS must be modified to operate at 300 Band)

Required Software

No additional

Input Parameters

Output Results

Provides the required 9 fill characters after each line-feed.
Provides 2.5 seconds delay after each form-feed.

Registers Modified:	Programmer: D. Brenneman
RAM Required:	Company: General Electric
ROM Required:	Address: GE Drive
Maximum Subroutine Nesting Level:	City: Waynesboro
Assembler/Compiler Used: ISIS 8080 MACRO ASSEMBLER V1.1	State: VA 22980

```
                ;REF. NO. D43
                ;PROGRAM TITLE TERMINET* 300
                ;
                ;
                TITLE 'TN300  2 7 77 DB'
                ;*****************************************************
                ;
                ; THIS IS A RAM RESIDENT SOFTWARE DRIVER
                ; FOR TERMINET 300
                ;
                ; PROVIDES FILL CHARACTERS AFTER LINE-FEED
                ;
                ; PROVIDES TIME DELAY AFTER FORM-FEED
                ;
                ; CREATES THESE ISIS FILES:
                ;       :L1: USER LIST DEVICE
                ;       :I1: USER CONSOLE INPUT
                ;       :O1: USER CONSOLE OUTPUT
                ;       :P1: USER PUNCH (FOR PAPER TAPE OUTPUT)
                ;
                ;*****************************************************
                ;
F600                    ORG     0F600H  ; FOR 65K MDS SYSTEM
                ;
                ;       PARAMETER DEFINITIONS

00F5            TTYS    EQU     0F5H    ; TTY STATUS
0001            TBE     EQU     1       ; TRANSMIT BUFFER EMPTY
0002            RBR     EQU     2       ; RECEIVE BUFFER READY
00F4            TTYDP   EQU     0F4H    ; TTY DATA PORT
000D            CR      EQU     0DH     ; CARRIAGE RETURN
000C            FF      EQU     0CH     ; FORM-FEED
000A            LF      EQU     0AH     ; LINE FEED
F81E            IODEF   EQU     0F81EH  ; MONITOR CALL
F815            IOCHK   EQU     0F815H  ; MONITOR CALL
F818            IOSET   EQU     0F818H  ; MONITOR CALL
F81B            MEMCK   EQU     0F81BH  ; MONITOR CALL
FE3B            DELAY   EQU     0FE3BH  ; 1.0 MS DELAY IN MONITOR
0040            ISIS    EQU     40H     ; ISIS SYSTEM CALL
0009            EXIT    EQU     9       ; EXIT CALL ID
0008            CONS    EQU     8       ; CONSOLE CALL ID
0003            CRTMSK  EQU     3       ; CONSOLE DEVICE MASK
0001            CRTDEV  EQU     1       ; ASSIGN CRT AS DEVICE

F600 31B0F6     START:  LXI     SP,STACK+16     ; DEFINE USER STACK

F603 0E01               MVI     C,1     ; INSTALL USER CONSOLE OUT
F605 114AF6             LXI     D,CO1
F608 CD1EF8             CALL    IODEF
```

```
F60B 0E06               MVI     C,6     ; INSTALL USER LIST DEVICE
F60D 114AF6             LXI     D,LO1
F610 CD1EF8             CALL    IODEF

F613 0E00               MVI     C,0     ; INSTALL USER CONSOLE IN
F615 117BF6             LXI     D,CI1
F618 CD1EF8             CALL    IODEF

F61B 0E07               MVI     C,7     ; INSTALL USER CONSOLE STATUS
F61D 1185F6             LXI     D,CS1
F620 CD1EF8             CALL    IODEF

F623 0E04               MVI     C,4     ; INSTALL USER PUNCH
F625 114AF6             LXI     D,PO1
F628 CD1EF8             CALL    IODEF
                ;
                ;       CHECK WHICH DEVICE IS COLD START CONSOLE
                ;
F62B CD15F8             CALL    IOCHK   ; MONITOR CALL
F62E E603               ANI     CRTMSK  ; CHECK IF CRT
F630 FE01               CPI     CRTDEV
F632 CA42F6             JZ      CRT     ; DO NOTHING
                ;
                ;       CONSOLE IS TERMINET, INSTALL USER CONSOLE
                ;
F635 0EE3               MVI     C,0E3H  ; MONITOR CONSOLE
F637 CD18F8             CALL    IOSET
F63A 0E08               MVI     C,CONS  ; ISIS CONSOLE
F63C 118EF6             LXI     D,KBLK
F63F CD4000             CALL    ISIS
                CRT:
F642 0E09               MVI     C,EXIT
F644 1192F6             LXI     D,EBLK
F647 CD4000             CALL    ISIS    ; RETURN TO ISIS

                PO1:                    ; OUTPUT TO PUNCH
                CO1:                    ; OUTPUT TO CONSOLE (IF TTY)
                LO1:                    ; OUTPUT TO LIST DEVICE
F64A DBF5               IN      TTYS
F64C E601               ANI     TBE     ; CHECK USART XMIT STATUS
F64E CA4AF6             JZ      CO1
F651 79                 MOV     A,C
F652 D3F4               OUT     TTYDP   ; OUTPUT CHARACTER
F654 E67F               ANI     7FH     ; STRIP PARITY BIT
F656 FE0C               CPI     FF      ; LOOK FOR FORM FEED
F658 CA6CF6             JZ      FFD     ; FORM FEED DELAY
F65B FE0A               CPI     LF      ; LOOK FOR LINE FEED
F65D C0                 RNZ
F65E C5                 PUSH    B
F65F 0609               MVI     B,9     ; NUMBER OF FILL REQUIRED
```

```
F661 0E7F               MVI     C,7FH   ; USING DELETE AS FILL
F663 CD4AF6   FILL:     CALL    CO1     ; OUTPUT FILL CHARACTER
F666 05                 DCR     B
F667 C263F6             JNZ     FILL
F66A C1                 POP     B
F66B C9                 RET

F66C C5       FFD:      PUSH    B
F66D 01C409             LXI     B,2500
F670 CD3BFE   FFDL:     CALL    DELAY   ; 1.0 MS DELAY IN MONITOR
F673 0B                 DCX     B
F674 78                 MOV     A,B
F675 B1                 ORA     C
F676 C270F6             JNZ     FFDL
F679 C1                 POP     B
F67A C9                 RET

             ;          KEYBOARD INPUT ROUTINE
             CI1:
F67B DBF5               IN      TTYS
F67D E602               ANI     RBR
  57F CA7BF6            JZ      CI1
r682 DBF4               IN      TTYDP
F684 C9                 RET

             ;          CONSOLE STATUS ROUTINE
             CS1:
F685 DBF5               IN      TTYS
F687 E602               ANI     RBR
F689 3E00               MVI     A,0
F68B C8                 RZ
F68C 2F                 CMA
F68D C9                 RET

             ;          PARAMETER BLOCK FOR ISIS CONSOLE ASSIGNMENTS

F68E 94F6     KBLK:     DW      INPUT   ; POINTER TO INPUT DEVICE STRING
F690 99F6               DW      OUTP    ; POINTER TO OUTPUT DEVICE STRING
F692 9EF6     EBLK:     DW      KSTAT   ; POINTER TO STATUS WORD
F694 3A49313A INPUT:    DB      ':I1: '
F698 20
F699 3A4F313A OUTP:     DB      ':O1: '
F69D 20
F69E          KSTAT:    DS      2

F6A0 A0F6     STACK:    DW      $

 300                    END     START
```

Ref.# D44

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

Sets Horizontal Tabs on TERMINET

Function

Required Hardware

MDS 800 System with MDS-DOS and TERMINET*

Required Software

No additional software required

Input Parameters

None

Output Results

Horizontal Tabs on connected TERMINET* are set at every 8 positions for this submittal

*Reg T.M.

Registers Modified:	**Programmer:** D. Brenneman
RAM Required:	**Company:** General Electric
ROM Required:	**Address:** GE Drive
Maximum Subroutine Nesting Level:	**City:** Waynesboro
Assembler/Compiler Used: ISIS 8080 MACRO ASSEMBLER V1.1	**State:** VA 22980

```
                        ;REF. NO. D44
                        ;PROGRAM TITLE SETHT
                        ;
                        ;
                        ;
                        TITLE 'SETHT 1/18/77 DB'

                        ; SETS HORIZONTAL TABS ON TERMINET AT EVERY 8 POSITIONS

00F5                    TTS     EQU     0F5H    ; TTY STATUS
0001                    TRDY    EQU     1       ; TRANSMIT BUFFER READY
00F4                    TTO     EQU     0F4H    ; TTY DATA PORT
001B                    ESC     EQU     1BH     ; ESCAPE KEY
000D                    CR      EQU     0DH     ; CARRIAGE RETURN
0009                    EXIT    EQU     9
0040                    ISIS    EQU     40H

F400                            ORG     0F400H

F400 3154F4             START:  LXI     SP,STACK+12

F403 2143F4                     LXI     H,TBL
F406 160E                       MVI     D,14    ; NUMBER OF SETTINGS
F408 1E05                       MVI     E,5

F40A 4E                 L1:     MOV     C,M
F40B CD38F4                     CALL    TTYOU
F40E 23                         INX     H
F40F 1D                         DCR     E
F410 C20AF4                     JNZ     L1

F413 1E08                       MVI     E,8     ; NUMBER OF SPACES BETWEEN SETTINGS
F415 0E20                       MVI     C,' '
F417 CD38F4             L2:     CALL    TTYOU
F41A 1D                         DCR     E
F41B C217F4                     JNZ     L2

F41E 2B                         DCX     H
F41F 2B                         DCX     H
F420 1E02                       MVI     E,2
F422 15                         DCR     D
F423 C20AF4                     JNZ     L1

F426 0E0D                       MVI     C,CR
F428 CD38F4                     CALL    TTYOU

F42B 21FEFF                     LXI     H,-2
F42E 39                         DAD     SP
F42F E5                         PUSH    H
F430 EB                         XCHG
```

```
F431 0E09              MVI     C,EXIT
F433 CD4000            CALL    ISIS
F436 FB                EI
F437 76                HLT

F438 DBF5      TTYOU:  IN      TTS
F43A E601              ANI     TRDY
F43C CA38F4            JZ      TTYOU
F43F 79                MOV     A,C
F440 D3F4              OUT     TTO
F442 C9                RET

F443 1B320D1B  TBL:    DB      ESC,'2',CR,ESC,'1'
F447 31

               STACK:

F400                   END     START
```

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/40 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ 3000 ☐ Other _____ (use additional sheets if necessary)

Program Title

GRAPH

Function

Plots on console a graph of one variable as well as axes and a grid.

Required Hardware

Teletype (or similar)

Required Software

'Console Output' subroutine

Input Parameters

A block of memory holding data bytes to be plotted.

Output Results

A graph on console.

Registers Modified: All	Programmer: Norm Campbell
RAM Required:	Company: City of Vancouver
ROM Required:	Address: 453 W. 12th Ave.
Maximum Subroutine Nesting Level:	City: Vancouver
Assembler/Compiler Used: MDS Macro Assembler	State: B.C. Canada V5Y 1V4

```
                ;REF.  NO.  D45
                ;PROGRAM TITLE GRAPH
                ;
                ;
                ;
                ;
                ;       SUBROUTINE GRAPH
                ;
                ;THIS SUBROUTINE WILL PLOT A GRAPH OF ONE FUNCTION
                ;OF ONE VARIABLE WHERE THE INDEPENDANT VARIABLE (X) INCREASES
                ;AT A FIXED STEP SIZE.  THE DATA TO BE PLOTTED IS TO BE STORED
                ;IN MEMORY AS A SET OF ONE-BYTE Y COORDINATES, THE FIRST
                ;CORRESPONDING TO X=0.  THE LAST DATA BYTE SHOULD BE FOLLOWED
                ;BY 0FFH.
                ;THE GRAPH IS PLOTTED WITH Y TO THE RIGHT AND X DOWNWARDS.  A SET
                ;OF AXES IS DRAWN, WITH MARKERS AT EACH 5TH DIVISION. IN ADDITION
                ;A GRID IS PLOTTED - BUT TO SAVE PLOTTING TIME, ONLY TO THE LEFT
                ;OF THE FUNCTION.  THE Y-AXIS IS DRAWN ONLY TO
                ;60 SPACES.  HOWEVER, Y-VALUES MAY BE PLOTTED UP TO A MAXIMUM OF
                ;128.  THE X-AXIS AND X-VALUES ARE UNLIMITED.
                ;EACH CALL TO GRAPH PRODUCES ONE LINE OF OUTPUT.  THEREFORE, THE
                ;USER MAY NUMBER THE AXES AS HE PLEASES.  AFTER PLOTTING THE LAST
                ;DATA BYTE, GRAPH RETURNS WITH THE CARRY FLAG SET TO INDICATE
                ;COMPLETION TO THE USERS CALLING PROGRAM.  ON FIRST CALL TO GRAPH
                ;H AND L SHOULD HOLD THE STARTING ADDRESS OF THE DATA TABLE, AND
                ;E SHOULD HOLD 0FFH.
                ;REGISTERS ARE USED AS FOLLOWS:
                ;  A - GENERAL
                ;  B - RECEIVES Y-COORDINATE FROM MEMORY, THEN DECREMENTED TO 0
                ;  C - HOLDS CHARACTER TO BE PRINTED
                ;  D - ON FIRST CALL, IS A Y-AXIS COUNTER,RANGING FROM -1 TO 4
                ;    - ON SUBSEQUENT CALLS, IS A Y-COORD.  COUNTER,RANGING FROM 0
                ;  E - ON FIRST CALL, A Y-AXIS SPACE COUNTER, INITIALLY 61
                ;    - ON SUBSEQUENT CALLS, A X-COORD.  COUNTER,  RANGING 0 TO 5
                ;  H,L - POINTER TO CURRENT DATA BYTE
                ;
F809            CO      EQU     0F809H
000D            CR      EQU     0DH
000A            LF      EQU     0AH

0100                    ORG 00100H

0100 1C         GRAPH:  INR  E          ;CHECK- 1ST LINE?
0101 C23A01             JNZ  GR05       ;NO
0104 113DFF             LXI  D,0FF3DH   ;YES
0107 46                 MOV  B,M        ;GET Y-COORD.
0108 05         GR01:   DCR  B          ;CHECK- REACHED Y YET?
0109 FA2201             JM   GR03       ;YES- JUMP
010C 14                 INR  D          ;INCR.  SPACE COUNTER
010D C21501             JNZ  GR02
```

```
0110 0E2B             MVI   C,'+'
0112 C32801           JMP   GR04

0115 0E2D     GR02:   MVI   C,'-'
0117 7A               MOV   A,D       ;EVAL D MODULO 5
0118 D604             SUI   004H
011A C21F01           JNZ   $+5
011D 2F               CMA             ;PUT -1 INTO D
011E 57               MOV   D,A
011F C32801           JMP   GR04

0122 0680     GR03:   MVI   B,80H     ;INSURE B NEVER GETS -VE AGAIN
0124 0E58             MVI   C,'X'     ;PLOT Y-COORD.
0126 14               INR   D
0127 23               INX   H

0128 CD09F8   GR04:   CALL  CO
012B 1D               DCR   E         ;FINISHED Y-AXIS?
012C C20801           JNZ   GR01      ;NO- JUMP
012F 0E0D             MVI   C,CR
0131 CD09F8           CALL  CO
 \34 0E0A             MVI   C,LF
0136 CD09F8           CALL  CO
0139 C9               RET             ;RETURN FROM 1ST CALL

013A 16FF     GR05:   MVI   D,0FFH    ;D=-1 INDICATES 'ON X AXIS'
013C 46               MOV   B,M       ;GET DATA
013D 04               INR   B         ;CHECK- FINISHED GRAPH?
013E C24301           JNZ   GR06      ;NO- JUMP
0141 37               STC             ;YES- SET CARRY
0142 C9               RET

0143 05       GR06:   DCR   B         ;RESTORE DATA
0144 7B               MOV   A,E       ;EVAL. LINE NO. MODULO 5
0145 D605             SUI   005H
0147 C24B01           JNZ   GR10
014A 5F               MOV   E,A

014B 05       GR10:   DCR   B         ;AT Y COORD. YET?
014C FA7201           JM    GR25      ;YES
014F 0E20             MVI   C,' '     ;NO- TYPE SPACE
0151 14               INR   D         ;CHECK- ON AXIS?
0152 C25801           JNZ   GR15      ;NO- JUMP
0155 0E2D             MVI   C,'-'     ;YES
0157 14               INR   D

^158 15       GR15:   DCR   D
 159 AF               XRA   A         ;CLEAR ACC.
015A 83               ADD   E         ;CHECK- LINE NO. MOD 5=0?
015B C26C01           JNZ   GR20      ;NO
```

```
015E 82              ADD  D          ;YES- CHECK- SPACES MOD 5=0?
015F C26401          JNZ  GR16       ;NO
0162 0E2B            MVI  C,'+'      ;YES

0164 14       GR16:  INR  D
0165 7A              MOV  A,D        ;EVAL. SPACES MODULO 5
0166 D605            SUI  005H
0168 C26C01          JNZ  GR20
016B 57              MOV  D,A

016C CD09F8   GR20:  CALL CO         ;OUTPUT SPACE,-,OR+
016F C34B01          JMP  GR10

0172 0E58     GR25:  MVI  C,'X'
0174 CD09F8          CALL CO         ;PLOT 'X'
0177 0E0D            MVI  C,CR
0179 CD09F8          CALL CO
017C 0E0A            MVI  C,LF
017E CD09F8          CALL CO
0181 23              INX  H
0182 C9              RET

0000                 END
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

Analog/Digital Polling Routine

Function

This program monitors a square-wave voltage connected to one bit of an 8-bit input port and counts the length of time the wave is in the low state (logic "0") during one complete cycle of the wave.

Required Hardware

Input port and circuit (see attached Fig. 1) to convert an analog signal, such as temperature, to a square wave which varies from 0 (logic "0") to +5 volts (logic "1").

Required Software

No other software required except the calling program. Additional software can be used, however, to convert the output count from this program into other units such as temperature.

Input Parameters

Any one of the eight bits of input port 1 can be used for input of the square wave. Register B should contain a value to mask off all bits except for the desired bit. For example, if the square wave is connected to bit 0 of input port 1, then B should contain a 01H.

Output Results

Registers D and E contain the double word digital count. The output count is dependent on the microprocessor clock speed. The counting loop in the program requires 29 clock time periods; therefore for a 2 MHz clock frequency, the output count resolution is 14.5 microseconds per count.

Registers Modified: A, B, D, E	Programmer: William G. Delinger Richard L. Petkiewicz
RAM Required: None	Company: Northern Arizona University
ROM Required: 17H Bytes	Address: Physics Department Box 6010
Maximum Subroutine Nesting Level: 0	City: Flagstaff
Assembler/Compiler Used: MAC80	State: AZ 86011

```
                        ;REF. NO. D46
                        ;PROGRAM TITLE ANALOG/DIGITAL POLLING ROUTINE
                        ;
                        ;
                        ;*****************************************************
                        ;*                                                   *
                        ;*  PROGRAM TITLE ANALOG/DIGITAL POLLING ROUTINE     *
                        ;*                                                   *
                        ;*  REGISTER B SHOULD CONTAIN A MASK TO PICK OUT     *
                        ;*  THE DESIRED INPUT BIT.                           *
                        ;*                                                   *
                        ;*  OUTPUT COUNT IS RETURNED IN REGISTER PAIR D&E     *
                        ;*                                                   *
                        ;*  OUTPUT COUNT RESOLUTION =                        *
                        ;*  (29 CLOCK PERIODS)/(CLOCK FREQUENCY)             *
                        ;*                                                   *
                        ;*****************************************************
0000                            ORG     0000H
0000 DB01       POLL:   IN      1               ;READ INPUT PORT 1
0002 A0                 ANA     B               ;MASK FOR DESIRED BIT
0003 CA0000             JZ      POLL            ;KEEP LOOPING IF BIT=0
 306 DB01       HIGH:   IN      1               ;READ INPUT PORT 1 AGAIN
0008 A0                 ANA     B               ;MASK FOR DESIRED BIT
0009 C20600             JNZ     HIGH            ;KEEP LOOPING IF BIT=1
000C 110100             LXI     D,0001H         ;INITIALIZE COUNTER
000F DB01       LOW:    IN      1               ;READ INPUT PORT 1 AGAIN
0011 A0                 ANA     B               ;MASK FOR DESIRED BIT
0012 13                 INX     D               ;INCREMENT COUNTER
0013 CA0F00             JZ      LOW             ;KEEP COUNTING IF BIT=0
0016 C9                 RET                     ;RETURN COUNT IN D&E
0000                    END
```

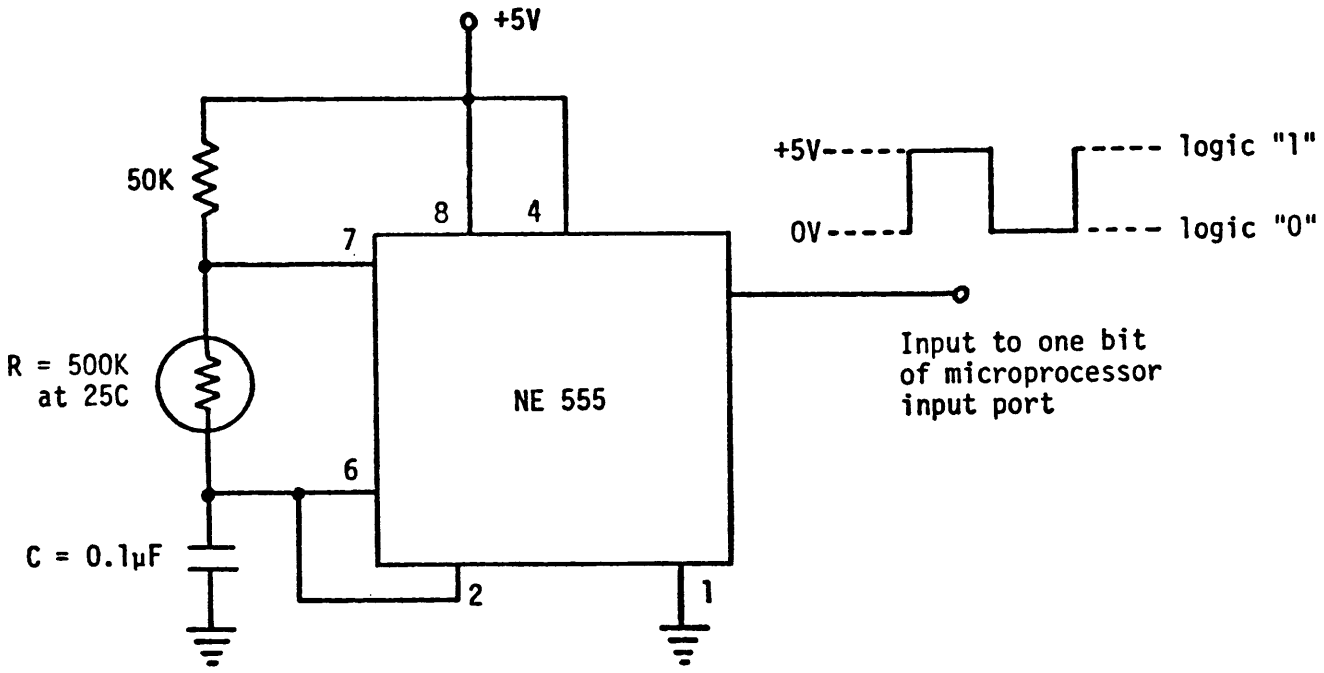


Fig. 1. The square wave produced by the 555 oscillator circuit is controlled by the temperature of the thermistor. During a complete cycle, the wave is in the low state (logic "0") for 0.693 RC seconds, where R is the resistance of the thermistor in ohms and C is the value of the timing capacitor in farads.

Ref. #D4/

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/40 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ 3000 ☐ Other _____ (use additional sheets if necessary)

Program Title

Thumbwheel SBC 80/10 Test Program

Function

Reads in six BCD digits from Thumbwheel switch settings and display them on Console. Program is stored in a 2708 Prom on the SBC board.

Required Hardware

SBC 80/10
TTY
6 digit Thumbwheel

Required Software

SBC 80P Monitor (PROM)

Input Parameters

This program reads into memory 6 BCD digits from a Thumbwheel Switch--- 2 digits per 8-Bit Port. The three ports 4, 5 & 6 (Address E8, E9 & EA) are configured together as 24 input lines (Mode 0). Each Port reads in 2 BCD digits, which are first separated, then converted to HEX-ASCII. The converted digits are then typed on the TTY. Program breaks to monitor to enable new Thumbwheel setting. A new Thumbwheel setting can be read in by typing a Monitor G Command.

Output Results

Digits are typed out on console, along with a prompt from the program saying you may change switch settings.

Registers Modified: ALL	Programmer: R. O. Christie
RAM Required: 10 BYTES + Stack	Company: Dept. of Environment Inland Water Directorate
ROM Required: 320 BYTES	Address: 131 Greber Blvd.
Maximum Subroutine Nesting Level:	City: Pte Gatineau, P. Q.
Assembler/Compiler Used: ISIS MACRO ASSEMBLER	State: Quebec, Canada

OUTPUT SAMPLES

```
SBC 80P MONITOR
.G800
                        SBC 80/10 TEST PROGRAM #1

READS IN SIX BCD DIGITS FROM THUMBWHEEL SWITCH SETTINGS
AND DISPLAYS THEM ON CONSOLE.

777777
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.G
123456
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.G
456789
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.G
987654
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.G
999999
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.G
000000
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.G
024680
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.G
135790
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.G
123456
YOU MAY CHANGE THUMBWHEEL SETTING.#0888
.D3FE0,3FE2                          Monitor used to display memory contents

3FE0 21 43 65            --- digit pairs from thumbwheel
.DF#
.D3FE3,3FE8

3FE3 31 32 33 34 35 36   --- digits separated and converted to ascii.
.
```

```
                ;REF. NO. D47
                ;PROGRAM TITLE THUMBWHEEL SBC 80/10 TEST PROGRAM
                ;
                ;
                ;
                ;               SBC 80/10 TEST PROGRAM #1
                ;
                ;THIS PROGRAM READS INTO MEMORY 6 BCD DIGITS FROM A
                ;THUMBWHEEL SWITCH --- 2 DIGITS PER 8-BIT PORT.
                ;THE THREE PORTS 4,5,&6 (ADDRESS E8,E9,&EA) ARE
                ;CONFIGURED TOGETHER AS 24 INPUT LINES (MODE 0).
                ;EACH PORT READS IN 2 BCD DIGITS, WHICH ARE FIRST
                ;SEPARATED, THEN CONVERTED TO HEX-ASCII.
                ;THE CONVERTED DIGITS ARE THEN TYPED ON THE TTY.
                ;A NEW THUMBWHEEL SETTING CAN BE READ IN BY
                ;TYPING A  -- G COMMAND --.


                ;-------INITIALIZATION-------------------------

0800                    ORG     0800H
0800 31FF3F             LXI     SP,3FFFH    ;TOP OF SBC80/10 RAM
03FD            CI      EQU     3FDH        ;SETS UP
03FA            CO      EQU     3FAH        ;USE OF
0400            RI      EQU     400H        ;MONITOR
0403            PO      EQU     403H        ;I/O ROUTINES
3FE0            STORE   EQU     3FE0H       ;BCD DIGIT STORAGE
3FE3            CONV    EQU     3FE3H       ;6 CONVERTED DIGITS
3FE9            COUNT   EQU     3FE9H       ;PASS COUNTER

                ;-------CONSOLE SIGN-ON MESSAGE----------------

0803 218B08             LXI     H,SINON     ;DISPLAY DATA ADDR.
0806 3E24       TYPE:   MVI     A,24H       ;STOP CHAR.
0808 4E                 MOV     C,M         ;GET FIRST CHAR.
0809 B9                 CMP     C
080A CA1408             JZ      EXIT
080D CDFA03             CALL    CO
0810 23                 INX     H
0811 C30608             JMP     TYPE
0814 00         EXIT:   NOP
0815 00                 NOP

                ;-------PORT CONFIGURATION--------------------

0816 3E9B               MVI     A,10011011B ;CONTROL WORD
```

```
0818 D3EB               OUT     0EBH        ;CONTROL REG.  ADDR.

                ;-------READ IN BCD DIGITS--------------------

081A 21E03F   START:    LXI     H,STORE
081D DBE8               IN      0E8H        ;FIRST BCD PAIR
081F 2F                 CMA
0820 77                 MOV     M,A
0821 23                 INX     H
0822 DBE9               IN      0E9H        ;SECOND PAIR
0824 2F                 CMA
0825 77                 MOV     M,A
0826 23                 INX     H
0827 DBEA               IN      0EAH        ;THIRD PAIR
0829 2F                 CMA
082A 77                 MOV     M,A
082B 00                 NOP

                ;-------SEPARATE BCD DIGITS & CONVERT---------

082C 21E93F             LXI     H,COUNT      ;PASS COUNTER#1
 32F 3603               MVI     M,3         ;3 PASSES
0831 21E03F             LXI     H,STORE     ;BCD DIGIT STORAGE
0834 11E33F             LXI     D,CONV      ;CONVERTED DIG.  STORAGE
0837 0E30               MVI     C,30H       ;USED FOR CONVERSION
0839 060F     NEXT:     MVI     B,0FH       ;USED FOR SEPARATION
083B 7E                 MOV     A,M         ;BCD PAIR
083C A0                 ANA     B           ;SEPARATE(ZERO 4 MSB.)
083D 81                 ADD     C           ;CONVERT
083E 12                 STAX    D           ;STORE IN CONV
083F 13                 INX     D
0840 7E                 MOV     A,M         ;SAME PAIR
0841 0F                 RRC                 ;SHIFT SECOND DIGIT -
0842 0F                 RRC                 ;OVER TO LOW ORDER -
0843 0F                 RRC                 ;BITS TO ENABLE -
0844 0F                 RRC                 ;CONVERSION.
0845 A0                 ANA     B           ;SEPARATE(ZERO 4 MSB.)
0846 81                 ADD     C           ;CONVERT
0847 12                 STAX    D           ;STORE IN CONV
0848 13                 INX     D
0849 23                 INX     H

084A E5                 PUSH    H           ;SAVE PRESENT CONV ADDR.
084B 21E93F             LXI     H,COUNT     ;PASS COUNTER #1
084E 7E                 MOV     A,M         ;COUNT-
084F 3D                 DCR     A           ;       DOWN
 950 C25608             JNZ     MINUS       ;NOT FINISHED.
 353 C35B08             JMP     SHOW        ;FINISHED.
0856 77       MINUS:    MOV     M,A         ;SAVE NEW PASS COUNT.
0857 E1                 POP     H           ;RESTORE PRESENT CONV ADDR.
```

```
0858 C33908            JMP     NEXT        ;BACK FOR NEXT CONVERSION.
085B 00        SHOW:   NOP
               ;-------OUTPUT CONVERTED DIGITS TO TTY--------

085C 0606              MVI     B,6         ;SIX DIGITS
085E 21E33F            LXI     H,CONV      ;FIRST DIGIT
0861 4E        TTY:    MOV     C,M
0862 CDFA03            CALL    CO
0865 23                INX     H

0866 05                DCR     B
0867 C26108            JNZ     TTY         ;OUTPUT FINISHED?
086A 0E0D              MVI     C,0DH       ;CR
086C CDFA03            CALL    CO
086F 0E0A              MVI     C,0AH       ;LF
0871 CDFA03            CALL    CO
0874 00                NOP

               ;-------TEST FOR NEW THUMBWHEEL SETTING------

0875 211D09            LXI     H,NEWTH
  378 3E24     THUMB:  MVI     A,24H       ;STOP CHAR.
087A 4E                MOV     C,M
087B B9                CMP     C
087C CA8608            JZ      LEAVE
087F CDFA03            CALL    CO
0882 23                INX     H
0883 C37808            JMP     THUMB
0886 00        LEAVE:  NOP


0887 CF                RST     1           ;BREAK TO MONITOR TO ENABLE -
                                           ;NEW THUMBWHEEL SETTING -
                                           ;THEN PROGRAM RESTARTS WITH -
                                           ;NEXT G COMMAND.
0888 C31A08            JMP     START

               ;-------CONSOLE DATA-----------------------

088B 20202020 SINON:   DB      '                    '
088F 20202020
0893 20202020
0897 20202020
089B 20202020
089F 2020
08A1 53424320          DB      'SBC 80/10 TEST PROGRAM #1'
^8A5 38302F31
 3A9 30205445
08AD 53542050
08B1 524F4752
```

```
08B5 414D2023
08B9 31
08BA 0D                 DB      0DH
08BB 0A                 DB      0AH
08BC 0A                 DB      0AH
08BD 52454144           DB      'READS IN SIX BCD DIGITS '
08C1 5320494E
08C5 20534958
08C9 20424344
08CD 20444947
08D1 49545320
08D5 46524F4D           DB      'FROM THUMBWHEEL SWITCH SETTINGS '
08D9 20544855
08DD 4D425748
08E1 45454C20
08E5 53574954
08E9 43482053
08ED 45545449
08F1 4E475320
08F5 0D                 DB      0DH
08F6 0A                 DB      0AH
 3F7 414E4420           DB      'AND DISPLAYS THEM ON CONSOLE.  '
08FB 44495350
08FF 4C415953
0903 20544845
0907 4D204F4E
090B 20434F4E
090F 534F4C45
0913 2E20
0915 0D                 DB      0DH
0916 0A                 DB      0AH
0917 0D                 DB      0DH
0918 0A                 DB      0AH
0919 24242424           DB      '$$$$'
091D 594F5520 NEWTH:    DB       'YOU MAY CHANGE THUMBWHEEL SETTING.'
0921 4D415920
0925 4348414E
0929 47452054
092D 48554D42
0931 57484545
0935 4C205345
0939 5454494E
093D 472E
093F 24242424           DB      '$$$$'

0000                    END
```

Ref. #D48

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other _____ (use additional sheets if necessary)

Program Title

Calculate a calendar for any year.

Function

To generate and print on a list device or console a calendar for any operator specified year.

Required Hardware

8080 system with console device and optional list device.

Required Software

Monitor console out (CO), console in (CI) and list out (LO).

Input Parameters

A year accepted from the console keyboard.

Output Results

See sample run.

Registers Modified: All	Programmer: William R. Ott
RAM Required: 708-Hex	Company: Applied Data Communications
ROM Required: Monitor I/O handlers or equal	Address: 1509 E. McFadden
Maximum Subroutine Nesting Level:	City: Santa Ana
Assembler/Compiler Used: 8080 Assembler	State: CA 92705

Ref.#D49

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

String Manipulation Package

Function

See Attached Sheets

Required Hardware

8080 Computer

Required Software

No Special Software Required

Input Parameters

See Attached Sheets

Output Results

See Attached Sheets

Registers Modified: All	Programmer: Frederick A. Stearns
RAM Required: See Attached Sheets	Company: Systems Consultants, Inc.
ROM Required: See Attached Sheets	Address: 3255 Wing Street
Maximum Subroutine Nesting Level: 7	City: San Diego
Assembler/Compiler Used: MDS-800 Assembler \ ISIS-I MACRO Assembler V1.0	State: California 92110

String Manipulation Package

Synopsis

The String Manipulation Package consists of four independent subroutines and two common utility routines whose purpose it is to manipulate character strings. Using these routines, it is possible to perform the following functions:

- Extract and store substrings (SUBST)
- Move the contents of one string (ASSGN) to another
- Compare one string to another (CMPST)
- Delimit a substring on the basis of inclusion in a set (DELMT)

Each string may be up to 255 characters long and is preceeded by a two byte prefix. The first byte contains the maximum length, 1 to 225; the second byte contains the current length, 0 to 255.

Parameters are passed to the procedures by writing their addresses after the procedure call. The utility routines transfer the information to the subroutines and skip over the parameters upon return.

Ref.#D50

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

Output Message Generator

Function

This program assembles messages and outputs them to the monitor's CO or LO device. The fixed part of the message is provided by a unique message table. The variable part(s) of the message are provided by the calling program which in addition, passes the number of the desired message.

Required Hardware

One CO-device (CRT,TTY) and/or one LO-device (Line Printer).

Required Software

Monitor (CO and LO routines)

Input Parameters

Address of the parameter list in B/C.
Contents of the parameter list:
- 1st byte: Message number in binary
- 2nd to last byte: insertions (variable parts of the message) in ASCII and terminated with an '@' char. The number of insertions must match the number of insertions required in the message definition in the message table. (See also the comments in the listing)

Output Results

A message (one or more lines) output to the CO or LO device.

Note

There are two modules. One is the OMG, the other one is the table containing the definition of the messages, i.e. their fixed part and the indications where the text segments provided by the calling program have to be inserted.
A third module allows to test the OMG by entering the parameter list on the CI-device. This module also shows how the OMG has to be called.

Registers Modified: none	Programmer: R. Genoud
RAM Required:	Company: Micro Control
ROM Required: 83 Bytes + Message Table	Address: 5107 Schinznach-Dorf Switzerland
Maximum Subroutine Nesting Level: 3	City:
Assembler/Compiler Used: ASM80	State:

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other SBC80/10 (use additional sheets if necessary)

Program Title

SBC 80P REAL TIME CLOCK

Function

MONITOR RESIDENT, REAL TIME CLOCK ROUTINE.
PROVIDES TIME AND CALENDAR INFORMATION.

Required Hardware

50 HERTZ INPUT PULSES SOURCE.

Required Software

SBC 80P MONITOR VER 1.1

Input Parameters

TIME: HOURS, MINUTES, SECONDS
DATE: DAY, MONTH, YEAR.

Output Results

UPDATES DATE AND TIME COUNTERS (STORED IN BCD AS SEVEN BYTES STARTING AT LOCATION DBUF).
DISPLAYS TIME AND DATE USING 'EXPANDED' MONITOR SPECIAL COMMANDS.

Registers Modified: NONE	Programmer: AVI KAHAN
RAM Required: 7 BYTES	Company: NUCLEAR RESEARCH CENTER
ROM Required: 418 BYTES	Address: P.O.B. 9001
Maximum Subroutine Nesting Level: 5	City: BEER-SHEBA
Assembler/Compiler Used: MACRO ASSEMBLER VER 3.0	State: ISRAEL

Ref. #D52

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

REAL TIME CLOCK SERVICE ROUTINE

Function

The program includes three routines. The first is used to initialize the system RTC and to store date and time into the appropriate buffer. The second is to write out the date and time on TTY. The third services the RTC interrupt.

Required Hardware

System real time clock

Required Software

ISIS-II V2.2
MDS MONITOR V2.Ø

Input Parameters

Example:
DATE: Ø6/Ø7/77 for JULY 6/77
TIME: 15:Ø5:ØØ for HOUR:MIN:SEC
Note that leap year is considered. With LEAP = 8ØH, the program is good until the end of 1983 by which time LEAP must be set equal to 84H.

Output Results

Time will be stored in appropriate buffer, i.e. from SEC to YEAR in buffer.

Registers Modified: ALL	Programmer: J.L. MARCEL LALONDE
RAM Required: 244H bytes	Company: AGRICULTURE CANADA
ROM Required:	Address: ENGINEERING RES. SERV., RESEARCH BRANCH
Maximum Subroutine Nesting Level: 4	City: OTTAWA, ONT. K1A 0C6
Assembler/Compiler Used: ISIS-II 8Ø8Ø/8Ø85 ASSEMBLER V1.Ø	State: ONTARIO

Ref. #D53

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

FIELD

Function

FIELD is a subroutine that provides zero suppression, check protection, floating dollar signs, and punctuation and text insertion which is useful in report preparation in both business and scientific applications. a source value is edited into a present mask. Edit control characters in the mask control value insertion. Non-edit control characters in the mask remain in the edited result.

Required Hardware

8080 System

Required Software

None

Input Parameters

DE = Low address of value to be edited
HL = Low address of edit mask
B = Length of edit mask

After

DE = High address of value +1
HL = High address of edit mask +1
B = Zero

Output Results

EXAMPLES:

```
VALUE   00000023          VALUE   6789012345
MASK    >>,>>>,>>>        MASK     $>,>>>,>>#.##
RESULT         23         RESULT  $67,890,123.45

VALUE   0000000009        VALUE   0000123456
MASK    $*,***,**#.##     MASK    **,***,**# DOLLARS AND ## CENTS
RESULT   *******$0.09     RESULT  *****1,234 DOLLARS AND 56 CENTS

VALUE   0789012345        VALUE   609LL51212PC1234GETTY, J.
MASK    $$,$$$,$$#.##     MASK    (>>>) >>>->>>> EXT. >>->>>> >>>>>>>>>>>
RESULT  $7,890,123.45     RESULT  (609) LL5-1212 EXT. PC-1234 GETTY, J.
```

Registers Modified: ABDEHL	Programmer: Austin C. Nester
RAM Required: 3+ Mask Length	Company: Coordinated Systems Corp.
ROM Required: B6	Address: 7300 Industrial Park
Maximum Subroutine Nesting Level: None	City: Pennsauken
Assembler/Compiler Used: 8080 Macro Assembler	State: NJ 08110

```
                ;REF. NO. D53
                ;PROGRAM NAME: FIELD
                ;
                ;CREATED: MARCH 3, 1977
                ;
                ;REVISED:
                ;
                ;BY COORDINATED SYSTEMS CORPORATION
                ;
                ;EDITING CONVENTIONS:
                ;       A SOURCE STRING IS EDITED INTO A PRESET MASK
                ;       ALL EDIT MASKS AND DATA ARE ASCII CHARACTERS
                ;       ONLY LEADING ASCII ZEROS ARE NOT SIGNIFICANT
                ;       ENCOUNTERING A . OR # IN A MASK MAKES THE CORRESPONDING
                ;               CHARACTER AND ALL FOLLOWING CHARACTERS SIGNIFICA
                ;       IF A MASK CHARACTER IS NOT . , $ > * OR # IT IS LEFT UNA
                ;       AFTER A SIGNIFICANT CHARACTER HAS BEEN ENCOUNTERED $ > *
                ;               EQUIVALENT TO #
                ;       $ MEANS INSERT A $ BEFORE THE FIRST SIGNIFICANT CHARACTE
                ;               OUT ALL $ > , BEFORE THE FIRST SIGNIFICANT CXHAR
                ;       > MEANS BLANK OUT ALL > , BEFORE THE FIRST SIGNIFICANT C
                ;       * MEANS INSERT A * OVER * , BEFORE THE FIRST SIGNIFICANT
                ;       , MEANS LEAVE THE , IF A SIGNIFICANT CHARACTER HAS BEEN
                ;       . MEANS LEAVE A . IN THE DESTINATION FIELD
                ;       # MEANS COPY THE NEXT SOURCE CHARACTER
                ;
                ;>>>>>>>>>>>>>>>>> SEE EXAMPLES IN THE TEST SECTION BELOW <<<<<<
                ;
                ;
                ;
                ;       IN  DE = SOURCE DIGIT FIELD IN ASCII, LEFT
                ;           HL = DESTINATION (MASK) FIELD, LEFT
                ;            B = LENGTH OF MASK FIELD
                ;       OUT DE = NEXT SOURCE DIGIT
                ;           HL = END OF MASK + 1
                ;            B = 0
                ;            C IS PRESERVED
                ;
                ;
                ;
 0900                   ORG     0900H
                EDITF:
 0900 AF                XRA     A
 0901 32B909            STA     FDF             ;RESET FLOATING $ FLAG
 0904 32BA09            STA     SDF             ;RESET SIGNIFICANT DIGIT FOUND F
 0907 32BB09            STA     ASF             ;RESET ASTERISK FLAG
                EDITG:
  30A 7E                MOV     A,M             ;GET MASK BYTE
 090B FE24              CPI     '$'
 090D CA2F09            JZ      DOLR
```

```
0910 FE3E               CPI     '>'
0912 CA5D09             JZ      ZSR
0915 FE2C               CPI     ','
0917 CA8409             JZ      COMR
091A FE2E               CPI     '.'
091C CA8E09             JZ      DOTR
091F FE23               CPI     '#'
0921 CA9E09             JZ      NINR
0924 FE2A               CPI     '*'
0926 CAAE09             JZ      ASTR
                ;               IF PROGRAM FELL THRU, LEAVE MASK BYTE IN RESULT
                EDITH:
0929 23                 INX     H
092A 05                 DCR     B               ;IF MORE MASK TO BE
092B C20A09             JNZ     EDITG           ; PROCESSED, DO NEXT BYTE
092E C9                 RET                     ; ELSE, EXIT
                ;
                DOLR:
092F 3ABA09             LDA     SDF
0932 B7                 ORA     A               ;IF NO SIGNIFICANT DIGIT HAS BEE
0933 CA3C09             JZ      DOLR3           ; EXAMINE THIS DIGIT
                DOLR1:
0936 1A                 LDAX    D               ;GET THIS DIGIT
                DOLR2:
0937 13                 INX     D               ;ADVANCE TO NEXT SOURCE DIGIT
                DOLRA:
0938 77                 MOV     M,A             ;STORE DIGIT OVER MASK BYTE
0939 C32909             JMP     EDITH           ;PROCESS NEXT DIGIT
                DOLR3:
093C 1A                 LDAX    D               ;GET THIS DIGIT
093D FE30               CPI     '0'             ;IF IT IS A 0
093F CA4C09             JZ      DOLR5           ; SUPRESS IT
0942 CD5409             CALL    DOLR6           ;INSERT THE $ 1 LOCATION LEFT
                DOLR4:
0945 3D                 DCR     A
0946 32BA09             STA     SDF             ;SET THE SIGNIFICANT DIGIT FOUND
0949 C33609             JMP     DOLR1           ;STORE DIGIT
                DOLR5:
094C 3E20               MVI     A,' '           ;GET A SPACE
094E 32B909             STA     FDF             ;SET FLOATING DOLLAR FLAG
0951 C33709             JMP     DOLR2           ;OVER WRITE MASK DIGIT
                DOLR6:
0954 2B                 DCX     H               ;BACK UP ONE LOCATION
0955 3624               MVI     M,'$'           ;INSERT THE $
0957 23                 INX     H               ;ADVANCE TO MASK DIGIT
0958 AF                 XRA     A
0959 32B909             STA     FDF             ;RESET FLOATING DOLLAR FLAG
095C C9                 RET
                ZSR:
095D 3ABA09             LDA     SDF
```

```
0960 B7              ORA     A          ;IF A SIGNIFICANT DIGIT HAS BEEN
0961 C23609          JNZ     DOLR1      ; COPY THIS DIGIT, ELSE
0964 1A              LDAX    D          ;GET THIS DIGIT
0965 FE30            CPI     '0'        ;IF IT IS A 0
0967 CA7709          JZ      ZSR1       ; SUPRESS IT
096A 3AB909          LDA     FDF
096D B7              ORA     A          ;IF NO FLOATING $ REQUIRED
096E CA4509          JZ      DOLR4      ; SET SIGNIFICANT DIGIT FOUND FL
0971 CD5409          CALL    DOLR6      ; INSERT THE $
0974 C34509          JMP     DOLR4      ; AND STORE THE DIGIT
                ZSR1:
0977 13              INX     D
                ZSR2:
0978 3ABB09          LDA     ASF
097B B7              ORA     A          ;IF CHECK PROTECT IS IN EFFECT
097C C2B409          JNZ     ASTR1      ; GET AN ASTERISK, ELSE
097F 3E20            MVI     A,' '      ;GET A SPACE
0981 C33809          JMP     DOLRA      ; OVER WRITE MASK BYTE
                COMR:
0984 3ABA09          LDA     SDF
0987 B7              ORA     A          ;IF NO SIGNIFICANT DIGIT HAS BEE
0988 CA7809          JZ      ZSR2       ; BLANK OUT MASK BYTE
098B C32909          JMP     EDITH      ; ELSE LEAVE THE COMMA
                DOTR:
098E 32BA09          STA     SDF        ;SET SIGNIFICANT DIGIT FOUND FLA
0991 3AB909          LDA     FDF
0994 B7              ORA     A          ;IF A FLOATING $ IS NOT REQUIRED
0995 CA2909          JZ      EDITH      ; LEAVE THE DECIMAL POINT
0998 CD5409          CALL    DOLR6      ; ELSE INSERT THE $ THEN
099B C32909          JMP     EDITH      ; LEAVE THE DECIMAL POINT
                NINR:
099E 32BA09          STA     SDF        ;SET SIGNIFICANT DIGIT FOUND FLA
09A1 3AB909          LDA     FDF
09A4 B7              ORA     A          ;IF A FLOATING $ IS NOT REQUIRED
09A5 CA3609          JZ      DOLR1      ; STORE THE DIGIT, ELSE
09A8 CD5409          CALL    DOLR6      ; INSERT THE $
09AB C33609          JMP     DOLR1      ; AND STORRE THE DIGIT
                ASTR:
09AE 32BB09          STA     ASF        ;SET THE CHECK PROTECT FLAG
09B1 C35D09          JMP     ZSR        ; DO CHECK PROTECTION
                ASTR1:
09B4 3E2A            MVI     A,'*'      ;GET AN ASTERISK
09B6 C33809          JMP     DOLRA      ; AND OVER WRITE THE MASK BYTE
                FDF:
09B9 00              DB      0H         ;FLOATING DOLLAR FLAG
                SDF:
09BA 00              DB      0H         ;SIGNIFICANT DIGIT FOUND FLAG
                ASF:
09BB 00              DB      0H         ;CHECK PROTECT FLAG
                ;
```

```
                        ;
                        ;       TEST    TEST    TEST    TEST    TEST    TEST    TEST
                        ;
                        T1:     ;                       23
09BC 3E3E2C3E                   DB      '>>,>>>,>>>'
09C0 3E3E2C3E
09C4 3E3E
                        T2:     ;                 *******$0.08
09C6 242A2C2A                   DB      '$*,***,**#.##'
09CA 2A2A2C2A
09CE 2A232E23
09D2 23
                        T3:     ;                *****1,234 DOLLARS AND 56 CENTS
09D3 2A2A2C2A                   DB      '**,***,**# DOLLARS AND ## CENTS'
09D7 2A2A2C2A
09DB 2A232044
09DF 4F4C4C41
09E3 52532041
09E7 4E442023
09EB 23204345
09EF 4E5453
                        T4:     ;                $7,890,123.45
09F2 24242C24                   DB      '$$,$$$,$$#.##'
09F6 24242C24
09FA 24232E23
09FE 23
                        T5:     ;                $67,890,123.45
09FF 20243E2C                   DB      ' $>,>>>,>>#.##'
0A03 3E3E3E2C
0A07 3E3E232E
0A0B 2323
                        T6:     ;                (609) LL5-1212 EXT. PC-1234 GETTY, J. SAUL
0A0D 283E3E3E                   DB      '(>>>) >>>->>>> EXT. >>->>>> >>>>>>>>>>>>>>>>>>>'
0A11 29203E3E
0A15 3E2D3E3E
0A19 3E3E2045
0A1D 58542E20
0A21 3E3E2D3E
0A25 3E3E3E20
0A29 3E3E3E3E
0A2D 3E3E3E3E
0A31 3E3E3E3E
0A35 3E3E3E3E
0A39 3E3E3E
                        V1:
0A3C 30303030                   DB      '00000023'
^940 30303233
                        V2:
0A44 30303030                   DB      '0000000008'
0A48 30303030
```

```
0A4C 3038
                V3:
0A4E 30303030           DB      '0000123456'
0A52 31323334
0A56 3536
                V4:
0A58 30373839           DB      '0789012345'
0A5C 30313233
0A60 3435
                V5:
0A62 36373839           DB      '6789012345'
0A66 30313233
0A6A 3435
                V6:
0A6C 3630394C           DB      '609LL51212PC1234GETTY, J. SAUL      '
0A70 4C353132
0A74 31325043
0A78 31323334
0A7C 47455454
0A80 592C204A
0A84 2E205341
 188 554C2020
0A8C 202020
                ;
                ;       SET THE FIELDS TO THE VALUES SHOWN
                ;
                TEST:
0A8F 113C0A             LXI     D,V1            ;GET SOURCE ADDRESS
0A92 21BC09             LXI     H,T1            ;GET DESTINATION ADDRESS
0A95 060A               MVI     B,10            ;GET DESTINATION SIZE
0A97 CD0009             CALL    EDITF           ;EDIT THE DATA
0A9A 060D               MVI     B,13
0A9C CD0009             CALL    EDITF
0A9F 061F               MVI     B,31
0AA1 CD0009             CALL    EDITF
0AA4 060D               MVI     B,13
0AA6 CD0009             CALL    EDITF
0AA9 060E               MVI     B,14
0AAB CD0009             CALL    EDITF
0AAE 062F               MVI     B,47
0AB0 CD0009             CALL    EDITF
0AB3 76                 HLT
0000                    END
```

Ref. #D55

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

SERIAL PROM PROGRAMMER

Function

Programs Intel 1K x 8 (2708) EPROMS on Prompt-80 using an external serial input.

Required Hardware

Prompt 80 modified for RS-232c (300 Baud) monitor V1.0

Required Software

External device must transmit to the Prompt-80 in Intel modified Hex Format.

Input Parameters

Data transmitted to Prompt-80 must be in Intel Hex Format (98-183A) modified to have frames 3-6 contain the start address of a 16 byte field in EPROM.

Output Results

A 16 byte Field in PROM is programmed and prompt of ASC II "L" sent to host device requesting next 16 bytes of data.

Registers Modified: ALL	**Programmer:** Michael C. Ebert
RAM Required: 17 Bytes	**Company:** Component Specialties, Inc.
ROM Required: 183 Bytes (2708)	**Address:** 8585 Commerce Park Dr. #590
Maximum Subroutine Nesting Level:	**City:** Houston
Assembler/Compiler Used: 8080/8085 Macro Assembler V2.0	**State:** Texas 77036

```
                        ;REF. NO. D55
                        ;PROGRAM TITLE SERIAL PROM PROGRAMMER
                        ;
                        ;
                        ;
                        ;       PROGRAM NAME: SERIAL PROMPT-80 PROGRAMMER
                        ;
                        ;       THIS PROGRAM ALLOWS THE USE OF AN INTEL PROMPT-80
                        ;       WITH ANY TERMINAL OR DEVICE CAPABLE OF GENERATING
                        ;       DATA IN INTEL'S STANDARD HEX FORMAT AND HAS A
                        ;       SERIAL PORT.
                        ;
                        ;       PROGRAM COMMANDS:
                        ;               9 - PROGRAM AND VERIFY EPROM
                        ;               A - VERIFY EPROM
                        ;       THESE COMMANDS ARE ENTERED ON THE HEX KEYBOARD
                        ;
00ED                    TTC     EQU     0EDH
00EC                    TXD     EQU     0ECH
00EC                    RXD     EQU     0ECH
0001                    TXRDY   EQU     1
  002                   RXRDY   EQU     2
0040                    RESET   EQU     40H
004F                    MODE    EQU     4FH
0027                    CMD     EQU     27H
003B                    ERR     EQU     3BH
08C2                    PROG    EQU     8C2H
0920                    CMPR    EQU     920H
3D00                    BFR     EQU     3D00H
3D10                    FLAG    EQU     3D10H

0C00                    ORG     0C00H
                        ;
                        ;       SET PROGRAM/VERIFY FLAG
                        ;
                        START:
0C00 C3060C             XP:     JMP     XP2
0C03 C30B0C             XC:     JMP     XC2
0C06 3EFF               XP2:    MVI     A,0FFH
0C08 C30C0C                     JMP     SAVE
0C0B AF                 XC2:    XRA     A
0C0C 32103D             SAVE:   STA     FLAG
                        ;
                        ;       PROGRAM USART FOR 10 BIT ASYNC FORMAT W/O PARITY
                        ;
0C0F 3E40                       MVI     A,RESET
0C11 D3ED                       OUT     TTC
0C13 3E4F                       MVI     A,MODE
0C15 D3ED                       OUT     TTC
0C17 3E27                       MVI     A,CMD
```

```
0C19 D3ED                       OUT     TTC
                        ;
                        ;       OUTPUT "L" COMMAND TO TERMINAL
                        ;
0C1B 0E4C               READY:  MVI     C,'L'
0C1D DBED               CO:     IN      TTC
0C1F E601                       ANI     TXRDY
0C21 CA1D0C                     JZ      CO
0C24 79                         MOV     A,C
0C25 D3EC                       OUT     TXD
                        ;
                        ;       READ 16 BYTE DATA BLOCK FROM TERMINAL
                        ;
0C27 CDA70C             READ:   CALL    CI
0C2A FE3A                       CPI     ':'
0C2C C2270C                     JNZ     READ
0C2F AF                         XRA     A
0C30 57                         MOV     D,A
0C31 CD760C                     CALL    BYTE
0C34 FE10                       CPI     10H
0C36 C23B00                     JNZ     ERR
  C39 5F                        MOV     E,A
0C3A CD760C                     CALL    BYTE
0C3D 67                         MOV     H,A
0C3E CD760C                     CALL    BYTE
0C41 6F                         MOV     L,A
0C42 E5                         PUSH    H
0C43 CD760C                     CALL    BYTE
0C46 FE00                       CPI     0
0C48 C23B00                     JNZ     ERR
0C4B 21003D                     LXI     H,BFR
0C4E CD760C             RD2:    CALL    BYTE
0C51 77                         MOV     M,A
0C52 23                         INX     H
0C53 1D                         DCR     E
0C54 C24E0C                     JNZ     RD2
0C57 CD760C                     CALL    BYTE
0C5A C23B00                     JNZ     ERR
0C5D D1                         POP     D
                        ;
                        ;       SET UP PROGRAM/VERIFY PARAMETERS
                        ;
 0C5E 211B0C                    LXI     H,READY
 0C61 E5                        PUSH    H
 0C62 21003D                    LXI     H,BFR
 0C65 E5                        PUSH    H
 0C66 210F3D                    LXI     H,BFR+15
   C69 E5                       PUSH    H
 0C6A D5                        PUSH    D
 0C6B 3A103D                    LDA     FLAG
```

```
0C6E FE00               CPI     0
0C70 C2C208             JNZ     PROG
0C73 C32009             JMP     CMPR
                ;
                ;       ASSEMBLE BYTE FROM TWO ASCII DIGITS
                ;
0C76 C5         BYTE:   PUSH    B
0C77 CDA70C             CALL    CI
0C7A CD8F0C             CALL    NIBBLE
0C7D 07                 RLC
0C7E 07                 RLC
0C7F 07                 RLC
0C80 07                 RLC
0C81 4F                 MOV     C,A
0C82 CDA70C             CALL    CI
0C85 CD8F0C             CALL    NIBBLE
0C88 B1                 ORA     C
0C89 4F                 MOV     C,A
0C8A 82                 ADD     D
0C8B 57                 MOV     D,A
0C8C 79                 MOV     A,C
  C8D C1                POP     B
0C8E C9                 RET
                ;
                ;       CONVERT ASCII DIGIT TO HEX DIGIT
                ;
0C8F D630       NIBBLE: SUI     '0'
0C91 DA3B00             JC      ERR
0C94 C6E9               ADI     '0'-'G'
0C96 DA3B00             JC      ERR
0C99 C606               ADI     6
0C9B F2A30C             JP      NIO
0C9E C607               ADI     7
0CA0 DA3B00             JC      ERR
0CA3 C60A       NIO:    ADI     10
0CA5 B7                 ORA     A
0CA6 C9                 RET
                ;
                ;       SERIAL INPUT ROUTINE
                ;
0CA7 DBED       CI:     IN      TTC
0CA9 E602               ANI     RXRDY
0CAB CAA70C             JZ      CI
0CAE DBEC               IN      RXD
0CB0 E67F               ANI     7FH
0CB2 C9                 RET
  C00                   END     START
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

Mailing Label Program - LABEL

Function

The program takes a list of names and addresses and outputs them to mailing labels on a line printer.

Required Hardware

MDS-800 and printer.

Required Software

ISIS

Input Parameters

A file containing the names and addresses to be printed.

The address file name is included with the program loading command, i.e.,

LABEL NAMES

The user is queried by the program for the number of names in the file.

Output Results

Prints two labels for each name in the file.

Registers Modified: All	Programmer: B. L. Masteller
RAM Required: 473H + Address File Storage	Company: Bendix - Mishawaka
ROM Required: ---	Address: 400 S. Beiger St.
Maximum Subroutine Nesting Level: 5	City: Mishawaka
Assembler/Compiler Used: PLM80	State: IN 46544

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE MAILINGLABELPGM
OBJECT MODULE PLACED IN :F1:LABEL.OBJ
COMPILER INVOKED BY:  PLM80 :F1:LABEL.SRC


           $TITLE('MAILING LABEL PROGRAM  VERSION 1.0')

            /*  THE PROGRAM TAKES A LIST OF NAMES AND ADDRESSES AND OUTPUTS THEM
               TO MAILING LABELS ON A LINEPRINTER.  THE NAME AND ADDRESS IS TYPED
               FROM A FILE ORGANIZED AS FOLLOWS:

                 (1ST LINE)          CLUB MEMBERSHIP LIST
               (2ND LINE)                          DATE
               (3RD LINE)  NAME    STREET      CITY STATE ZIP        MISC INFO
               (4TH LINE) ASHLEY,TOM  RR 2     DOWAGIAC, MI 46716     245 3722

               (NTH LINE) ZIMMER,JOE   104 SOUTH ST.   KEWANNA, IN 46166     357 2888

               THE PROGRAM IS INVOKED BY THE COMMAND

                  LABEL FILENAME

               WHERE FILENAME IS THE FILE COTAINING THE NAMES AND ADDRESS.

               THE USER IS QUERIED FOR THE NUMBER OF NAMES IN THE FILE.  THE MAXIMUM
               NUMBER IS 255 DECIMAL.  */

 1         MAILING$LABEL$PGM: DO;

 2  1       DECLARE FILENAME(16) BYTE;
 3  1      DECLARE IN$BFR(3) BYTE;
 4  1      DECLARE MSG1(*) BYTE DATA('HOW MANY NAMES IN FILE? ');
 5  1      DECLARE AFT$BB BYTE;
 6  1      DECLARE BUCKET(*) BYTE DATA(':BB:');
 7  1      DECLARE DD BYTE INITIAL(36);  /* DD IS THE NUMBER OF SPACES FROM THE START OF
                ONE LABEL TO THE BEGINNING OF THE NEXT */
 8  1      DECLARE (L,M,N,NMBR$CHARA,NMBR$TABS,NMBR$SPACES)BYTE;
 9  1      DECLARE NMBR$NAMES BYTE INITIAL (31);
10  1      DECLARE (K,ACTUAL$COUNT,STATUS,AFT$IN) ADDRESS;
11  1      DECLARE LABEL$BFR (36) BYTE;
12  1      DECLARE READ$ACCESS LITERALLY '1';
13  1      DECLARE LINE$PRNTR (*) BYTE DATA (':LP:');
14  1      DECLARE CRLF (2) BYTE DATA (0DH,0AH);

15  1      OPEN:   PROCEDURE (AFTPTR,FILE,ACCESS,MODE,STATUS) EXTERNAL;
16  2         DECLARE (AFTPTR,FILE,ACCESS,MODE,STATUS) ADDRESS;
17  2         END OPEN;

18  1      CLOSE:  PROCEDURE (AFT,STATUS) EXTERNAL;
19  2         DECLARE (AFT,STATUS) ADDRESS;
20  2         END CLOSE;

21  1      READ:   PROCEDURE (AFT,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL;
22  2         DECLARE (AFT,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
23  2         END READ;
```

```
 24  1     WRITE:  PROCEDURE (AFT,BUFFER,COUNT,STATUS) EXTERNAL;
 25  2        DECLARE (AFT,BUFFER,COUNT,STATUS) ADDRESS;
 26  2        END WRITE;

 27  1     EXIT:   PROCEDURE  EXTERNAL;
 28  2        END EXIT;

 29  1     ERROR:  PROCEDURE (ERRNUM) EXTERNAL;
 30  2        DECLARE (ERRNUM) ADDRESS;
 31  2        END ERROR;

 32  1     LINE$EDIT$BFR:
           PROCEDURE BYTE;
 33  2        CALL READ (1,.L,1,.ACTUAL$COUNT,.STATUS);
 34  2        RETURN L;
 35  2        END;

 36  1     CONVERT: PROCEDURE;
 37  2        NMBR$NAMES = 0;
 38  2        DO N = 0 TO 2;
 39  3           IF (IN$BFR(N) AND 0F0H) = 30H
                THEN NMBR$NAMES = (10 * NMBR$NAMES) + (IN$BFR(N) - 30H);
 41  3           ELSE RETURN;
 42  3           END;
 43  2        END CONVERT;

              /* START OF MAIN PROGRAM */

              /* READ FILE NAME FROM LINE EDIT BUFFER */

 44  1     DO N = 0 TO 15;
 45  2        FILENAME(N) = 20H;
 46  2        END;

 47  1     DO WHILE LINE$EDIT$BFR = ' ';
 48  2        END;

 49  1     FILENAME(0) = L;
 50  1     N = 1;

 51  1     DO WHILE LINE$EDIT$BFR <> 0DH AND N < 16;
 52  2        FILENAME(N) = L;
 53  2        N = N + 1;
 54  2        END;

              /* CLEAR BUFFER */

 55  1     CALL OPEN(.AFT$BB,.BUCKET,2,0,.STATUS);
 56  1     CALL READ (1,.BUCKET,122,.ACTUAL$COUNT,.STATUS);

              /* QUERY FOR NUMBER OF NAMES IN FILE */

 57  1     CALL WRITE (0,.MSG1,LENGTH(MSG1),.STATUS);
 58  1     CALL READ (1,.IN$BFR,3,.ACTUAL$COUNT,.STATUS);

 59  1     CALL CONVERT;
```

```
60  1     CALL OPEN (.AFT$IN,.FILENAME,READ$ACCESS,0,.STATUS);
61  1     IF STATUS <>0 THEN
62  1        DO;
63  2             CALL ERROR(STATUS);
64  2           CALL EXIT;
65  2           END;

              /* READ FILE OF NAMES AND ADDRESSES FROM INPUT FILE */

66  1     CALL READ (AFT$IN,.MEMORY,(255 * 80),.ACTUAL$COUNT,.STATUS);
67  1     CALL CLOSE (AFT$IN,.STATUS);
68  1     CALL OPEN (.AFT$IN,.LINE$PRNTR,2,0,.STATUS);

              /* SKIP HEADER LINES OF INPUT FILE */

69  1     K,N = 0;

70  1     DO WHILE N < 3;
71  2        IF MEMORY(K) = 0AH THEN
72  2           N = N + 1;
73  2        K = K + 1;
74  2        END;

              /* START OF PRINTING ROUTINE */

75  1     DO$ALL: DO N = 1 TO NMBR$NAMES;

76  2     DO$ONE$ROW: DO M = 1 TO 3;   /* WRITE 3 LINES OF DATA PER LABEL */
77  3           DO NMBR$CHARA = 0 TO DD;    /*FILL LABEL BUFFER WITH SPACES */
78  4              LABEL$BFR$(NMBR$CHARA) = 20H;
79  4              END;

80  3           NMBR$CHARA,NMBR$SPACES,NMBR$TABS = 0;

81  3     FILL$LABEL$BFR: DO WHILE (NMBR$CHARA < DD) AND (NMBR$SPACES < 2) AND (NMBR$TABS = 0);
82  4              IF MEMORY(K) = 09   /* TEST FOR TAB */
                    THEN NMBR$TABS = 1;
84  4              ELSE DO;
85  5                 IF MEMORY(K) = ' '
                       THEN NMBR$SPACES = NMBR$SPACES + 1;
87  5                 ELSE NMBR$SPACES = 0;
88  5                 LABEL$BFR(NMBR$CHARA) = MEMORY(K);
89  5                 NMBR$CHARA = NMBR$CHARA + 1;
90  5                 END;

91  4              K = K + 1;
92  4              END FILL$LABEL$BFR;

93  3           DO L = 1 TO 2;
94  4              CALL WRITE(AFT$IN,.LABEL$BFR,DD,.STATUS);
95  4              END;

96  3           CALL WRITE(AFT$IN,.CRLF,2,.STATUS); /* PRINT THE LINE */

              /* FIND THE START OF NEXT FIELD */

97  3           DO WHILE ((MEMORY(K) = ' ') OR (MEMORY(K) = 09));
```

```
 98  4                K = K + 1;
 99  4                END;

100  3            END DO$ONE$ROW;

               /* FIND NEXT LINE */

101  2      NEXT$LINE: DO WHILE MEMORY(K) <> 0AH;
102  3            K = K + 1;
103  3            END;

104  2      NEXT$LABEL: DO L = 1 TO 3;
105  3            CALL WRITE(AFT$IN,.CRLF,2,.STATUS);
106  3            END;

107  2         END DO$ALL;

108  1      CALL CLOSE(AFT$IN,.STATUS);
109  1      CALL EXIT;

110  1      END MAILING$LABEL$PGM;



MODULE INFORMATION:

     CODE AREA SIZE     = 0345H    837D
     VARIABLE AREA SIZE = 0048H     72D
     MAXIMUM STACK SIZE = 000AH     10D
     186 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

Ref. #D57

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☒8085 ☐Other_____ (use additional sheets if necessary)

Program Title

INTELLEC MDS CHECK BOOK BALANCING

Function

Allows user to maintain a file, complete with password, of checks and deposits - all with a description and data then the program returns the balance to the console. The information may be recalled and displayed at any time.

Required Hardware

MDS-800/220/230
MDS-DDS/2DS
Line Printer (optional)

Required Software

ISIS-II disc operating system version 2.0 or 3.4

Input Parameters

All inputs are user prompted at the system console as what to enter and on critical entries; length is also specified

**The source program may reside on any drive, but the user file is set up to be on drive 0 only.

Output Results

Computed balance will be displayed on the system console. The listing of past activities may be printed on either the system console, or both the console and the line printer past activities may all be printed or may be printed by month or check number.

**Note: When signing off this program, due to an error in the Fortran Compiler - if you answered no to the question "do you have a line printer" you will get an ISIS error and a Fortran error. Disregard these statements as they are meaningless in this situation.

Registers Modified: ALL	Programmer: Kerry Howell
RAM Required: 64K RAM	Company: Almac Stroum Elec.
ROM Required: 2K MDS system monitor	Address: 18760 NW Rock Creek Circle #134
Maximum Subroutine Nesting Level: N/A	City: Portland
Assembler/Compiler Used: Fortran-80 V1.0	State: Oregon 97229

Ref. #D58

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

FIFO - First-in, First-out Buffer Routine

Function

This program performs the function of a First-in, First-out buffer and consists of 2 sub-routines; STORE and LOAD. Up to 256 bytes of RAM can be reserved for buffer memory, with the first byte being defined by MNDTA. When a byte of data is to be stored, the main program (or an interrupt service routine) loads the accumulator with the data and calls STORE. When a byte of data is to be loaded from the FIFO, the main program calls LOAD and obtains data from memory location BUF. When the FIFO is empty and an attempt is made to load data from the FIFO, sub-routine LOAD returns to the calling program with the carry bit set.

Required Hardware

Required Software

CRT and TTY Software Drivers.

Input Parameters

Output Results

Registers Modified: ALL	Programmer: Mervin Doda
RAM Required: up to 259 bytes	Company: CANADAIR LTD.
ROM Required: 118 bytes	Address: P.O. Box 6087
Maximum Subroutine Nesting Level:	City: Montreal
Assembler/Compiler Used: MDS Macro Assembler Ver. 1.0	State: CANADA H3C 3G9

ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0 MODULE PAGE 1

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  1     ; F I F O
                  2
F80F              3 CO      EQU   0F80FH
F803              4 CI      EQU   0F803H
                  5
                  6     ;MNDTA DEFINES THE START OF FIFO MEMORY
                  7
0100              8         ORG   100H
0100 310002       9         LXI   SP,200H
                 10     ;INITIALIZE CONTENTS OF LIP = MNDTA.
                 11
0103 217A01      12         LXI   H,MNDTA
0106 227801      13         SHLD  LIP
                 14     ;MAIN PROGRAM RECEIVES ONE CHARACTER FROM CI.
                 15     ;IF THIS CHARACTER IS A SPACE, THE PROGRAM CALLS
                 16     ;LOAD SUB-ROUTINE.  IF THE CHARACTER IS NOT A SPACE,
                 17     ;THE CHARACTER IS PRINTED ON CO AND STORE
                 18     ;SUB-ROUTINE IS CALLED.
                 19
0109 CD03F8      20 FIFO1:  CALL  CI
010C E67F        21         ANI   7FH      ;MASK PARITY
010E FE20        22         CPI   20H      ;SPACE?
0110 CA1D01      23         JZ    FIFO2    ;YES
0113 CD2A01      24         CALL  STORE    ;NO, CHAR. IS STORED.
0116 4F          25         MOV   C,A
0117 CD0FF8      26         CALL  CO
011A C30901      27         JMP   FIFO1
011D CD3701      28 FIFO2:  CALL  LOAD
0120 3A7701      29         LDA   BUF      ;BUF CONTAINS FIRST-OUT
0123 4F          30         MOV   C,A      ;DATA FROM FIFO.
0124 CD0FF8      31         CALL  CO
0127 C30901      32         JMP   FIFO1
                 33
                 34     ;STORE SUB-ROUTINE
                 35
012A F5          36 STORE:  PUSH  PSW
012B E5          37         PUSH  H
012C 2A7801      38         LHLD  LIP      ;LIP POINTS TO FIFO
012F 77          39         MOV   M,A      ;MEMORY BYTE THAT WILL
0130 23          40         INX   H        ;STORE PRESENT CHARACTER.
0131 227801      41         SHLD  LIP
0134 E1          42         POP   H
0135 F1          43         POP   PSW
0136 C9          44         RET
                 45
                 46     ;LOAD SUB-ROUTINE
                 47
0137 C5          48 LOAD:   PUSH  B
0138 D5          49         PUSH  D
0139 E5          50         PUSH  H
                 51     ;BUF IS LOADED WITH FIRST-OUT DATA.
013A 3A7A01      52         LDA   MNDTA
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

013D 327701      53         STA   BUF
                 54     ;MNDTA IS SUBTRACTED FROM THE CONTENTS OF LIP.
0140 117A01      55         LXI   D,MNDTA
0143 2A7801      56         LHLD  LIP
0146 AF          57         XRA   A
0147 47          58         MOV   B,A
0148 7B          59         MOV   A,E      ;FORM 2'S COMPLIMENT
0149 2F          60         CMA            ;OF MNDTA.
014A 3C          61         INR   A
014B 5F          62         MOV   E,A
014C 7A          63         MOV   A,D
014D 2F          64         CMA
014E 88          65         ADC   B        ;CARRY BIT IS ADDED
014F 57          66         MOV   D,A      ;TO D.
0150 19          67         DAD   D
                 68     ;DIFFERENCE IS STORED IN C.
0151 4D          69         MOV   C,L
                 70     ;IF LIP POINTS TO MNDTA, SET CARRY AND RETURN.
0152 79          71         MOV   A,C
0153 FE00        72         CPI   00H
0155 CA7301      73         JZ    LD2
                 74     ;INITIALIZE MEMORY TO MEMORY TRANSFER.
0158 217A01      75         LXI   H,MNDTA
015B 117A01      76         LXI   D,MNDTA
015E 23          77         INX   H
                 78     ;FIFO RIPPLE THROUGH DATA TRANSFER
015F 7E          79 LD1:    MOV   A,M
0160 12          80         STAX  D
0161 13          81         INX   D
0162 23          82         INX   H
0163 0D          83         DCR   C
0164 C25F01      84         JNZ   LD1
                 85     ;DECREMENT CONTENTS OF LIP.
0167 2A7801      86         LHLD  LIP
016A 2B          87         DCX   H
016B 227801      88         SHLD  LIP
016E B7          89         ORA   A        ;CLEAR CARRY
016F E1          90 LD3:    POP   H
0170 D1          91         POP   D
0171 C1          92         POP   B
0172 C9          93         RET
0173 37          94 LD2:    STC
0174 C36F01      95         JMP   LD3
                 96
0001             97 BUF:    DS    1
0002             98 LIP:    DS    2
0100             99 MNDTA:  DS    256
                100         END
```

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

```
BUF    A 0177   CI     A F803   CO     A F80F   FIFO1  A 0109   FIFO2  A 011D   LD1    A 015F   LD2    A 0173
LD3    A 016F   LIP    A 0178   LOAD   A 0137   MNDTA  A 017A   STORE  A 012A
```

ASSEMBLY COMPLETE, NO ERRORS

Ref.#D59

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☒ 8008 ☐ 8080 ☐ 8048 ☐ 8085 ☐ Other_____ (use additional sheets if necessary)

Program Title

COS: A cassette operating system for the MDS-800

Function

Provides practical substitution of cassette storage for paper tape devices. Storage on cassette is buffer oriented and software routines provide for byte-by-byte reading and/or writing of data. It is completely integrated and compatable with the MDS monitor. The operating system allows file naming, search and directory, INTEL format HEX files, and opening and closing of ASCII files.

Required Hardware

Cassette I/O on ports E0 and E1 HEX)
Cassette remote control on port E3 HEX) schematic available
1 or 2 audio cassette recorders
MDS-800

Required Software

MDS Monitor Version 2.0 (may be adapted to others)

Input Parameters

Output Results

Registers Modified: ALL	Programmer: Robert A. McCormick
RAM Required: .75 Kbytes	Company: Frye Electronics, Inc.
ROM Required: 1.25 Kbytes	Address: PO Box 23391
Maximum Subroutine Nesting Level:	City: Tigard
Assembler/Compiler Used: ISIS-II MACRO ASS. VER. 2.0	State: OR 97223

Ref.#D60

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☐8080 ☐8048 ☐8085 ☒Other MDS-system (use additional sheets if necessary)

Program Title

INTELLEC MDS MAILING LIST - FORTRAN-80

Function

Allows the user to maintain a disc based mailing list of name, phone number, address and optional attributes.

The mailing list may then be printed on shipping labels or recalled to the console.

Required Hardware

MDS-800/220/230 MDS-PRN (line printer)
Tractor feed labels (Dennison #42-551-0, 3 1/2 X 15/16

Required Software

ISIS-II disc operating system version 2.2 or 3.4

Input Parameters

All inputs are user prompted at the system console as what to enter, and on critical entries, the length is also specified.

Output Results

The mailing list will be stored on the disc drive under the user assigned name.
Shipping labels are printed on the line printer when called in the program.

Registers Modified: All	Programmer: Kerry P. Howell
RAM Required: 32K Ram	Company: Almac/Stroum
ROM Required: 2K MDS System monitor	Address: 18760 NW Rock Creek Circle #134
Maximum Subroutine Nesting Level: N/A	City: Portland
Assembler/Compiler Used: Fortran-80 V1.0	State: OR 97229

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☐ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

PLOTA

Function

This program plots up to loo coordinates, by the TTY (of other console device), using 64 columns by 64 lines.
All coordinates must be integer , positive, from 0 to 1023

Required Hardware

Basic Intelec MDS 800, i.e., Central Processor Module, Front Panel Control Module, Monitor Module, one 16K RAM module, connected to a teletype.

Required Software

Routines CO, CRLF, TI, PARAM, LBYTE from Intellec MDS Monitor, version V2.0, the DIVIDE routine on page 55 of the "8080 Assembly Language Programming Manual"-Rev. C, and any BCD to Binary (BCDB) conversion routine (given BCD in HL reg.pair, returns Binary in DE reg.pair)

Input Parameters

The program is conversational, by the TTY (console device).
The scale that the program asks is the greatest value in the axis.
After answering 'YES' on 'NO' to the two questions, before pressing Carriage Return, the line can be used for writing remarks up to its end. In these answers, only an 'N' as the first letter means 'NO'; any other character means 'YES'.
Note: When entering the data NEVER press 'SPACE' immediately after a '?' or a ',', because the routine PARAM doesn't accept the NULL character (if it happens, PARAM returns to the Monitor!)

Output Results

A plot of the given coordinates is printed on the TTY (of console device), using the character '*' to indicate a point and the character '>' to indicate that a point is greater than the given maximum Y value.

Registers Modified:		Programmer:	Fernando Jordan
RAM Required:		Company:	IPT - AIA
ROM Required:		Address:	P. O. box 7141
Maximum Subroutine Nesting Level:		City:	Sao Paulo
Assembler/Compiler Used:	Intellec MDS Macro Assembler V2.2	State:	Brazil

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

SORT - GENERAL SORTING PROCEDURES

Function

General sorting procedure: working on complete lines or on fields (e.g. locate file: 1234H - PUB -- NAME)

Required Hardware

MDS 800, diskette driver (DD), console, line printer (optional)

Required Software

ISIS-II, monitor

Input Parameters

Any file to be sorted

- SORT :Fi:FILENAME.EXTENSION

Questions (see listing)

Output Results

:Fi:FILENAME.CLS on disk
and on :CO: or :LP: as well

Registers Modified:	All	Programmer:	Maessen JL
RAM Required:	1000H (CODE + ISIS) 6500H (DATA)	Company:	Bell Telephone Mfg C° ITT
ROM Required:	Monitor	Address:	Bell Telephonelaan 2
Maximum Subroutine Nesting Level:	2	City:	B-2440 GEEL
Assembler/Compiler Used:	MDS 8080/8085 Macro Assembler	State:	Belgium

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

D63

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

MAILING LIST MERGE

Function

This program will merge two mailing lists that were created by Intellec MDS Mailing List (Ref.no. D60) into one file. Program will check for names repeated between the two files and will not append repeats to the destination file.

Required Hardware

MDS 800/220/230
Disc drive

Required Software

ISIS-II disc operating system version 2.2 or 3.4
Intellec MDS Mailing List (Ref.No. D60

Input Parameters

User is asked to enter a destination file and a source file. The program will append from the source file to the destination file but does not change or destroy the source file in any way. The two files must have been created by using program Intellec MDS Mailing List.

Output Results

The system console will show what records are being appended to the destination file from the source file.

Registers Modified: ALL	Programmer: Kerry Howell
RAM Required: 32K RAM	Company: Almac/Stroum Electronics
ROM Required: 2K system monitor	Address: #134 18760 NW Rock Creek Circle
Maximum Subroutine Nesting Level: NA	City: Portland
Assembler/Compiler Used: FORTRAN80, V1.0	State: Oregon 97229

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 XX 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

FILES

Function

This module consists of a group of utility procedures which ease file oriented I/O under ISIS-II.

Required Hardware

MDS with Disk System, Console Device

Required Software

ISIS-II
PL/M 80 Complier

Input Parameters

Output Results

Registers Modified:	ALL	Programmer:	D.M. Brockman/E. Kozel
RAM Required:	380 Bytes	Company:	Boeing Commercial Airplane Company
ROM Required:	656 Bytes	Address:	P.O. Box 3707 MS:25-09
Maximum Subroutine Nesting Level:	1	City:	Seattle
Assembler/Compiler Used:	PL/M 80	State:	Washington 98124

SECTION 8

GAMES

---

*PROGRAM HAS BEEN CONVERTED TO RUN ON THE SDK-80 AND IS AVAILABLE TO INSITE MEMBERS UPON REQUEST.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. E1

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

THE GAME NIM

Function

Please see attachment

Required Hardware

Intellec 8/80 with 4K RAM (Minimum) and TTY (or equivalent) Console.

Required Software

None - The program runs indpendently of other software or ROMs. However, a Hex tape loader such as in Monitor is required to read in the Hex paper tape.

Input Parameters

Output Results

Registers Modified: All	**Assembler/Compiler Used:** 8080 Assembler Ver. 3.0
RAM Required: 700 Hex Bytes	**Programmer:** William R. Ott
ROM Required: None	**Company:** Applied Data Communications
Maximum Subroutine Nesting Level: N/A	**Address:** 1509 E. McFadden Santa Ana, Ca. 92705

A P P L I E D D A T A C O M M U N I C A T I O N S

THE GAME NIM MAY 1 1975

NIM IS AN ANCIENT GAME PLAYED BETWEEN TWO PLAYERS WITH READILY AVAILABLE MARKERS THAT COULD BE CHIPS, STONES, COINS ETC. THE GAME IS PLAYED WITH TWO PLAYERS AND IS STARTED BY ONE PLAYER SETTING UP THREE PILES OF CHIPS. THEN, IN TURN, EACH PLAYER TAKES HIS TURN AND REMOVES AS MANY CHIPS AS HE WISHES FROM A SINGLE PILE. HE MUST TAKE AT LEAST ONE CHIP. THE PLAY CONTINUES UNTIL ALL CHIPS ARE TAKEN AND THE PLAYER THAT TAKES THE LAST CHIP WINS THE GAME.

IT IS THE TYPE OF GAME IN WHICH THERE ARE 3 STATES ONLY; PLAYER 1 ADVANTAGE, NEUTRAL AND PLAYER 2 ADVANTAGE. EACH MOVE CAN ONLY MOVE ONE STEP. E.G. FROM NEUTRAL TO A PLAYER'S ADVANTAGE OR A PLAYER ADVANTAGE TO NEUTRAL. SO, IN GENERAL, IT CAN BE SEEN THAT ONCE AN ADVANTAGE IS OBTAINED AND NO MOVES ARE MADE TO PRODUCE A MOVE FROM NEUTRAL TO THE OPPONENTS ADVANTAGE, YOU WILL WIN. NOW THE COMPUTER IS A VERY GOOD PLAYER IN THAT THROUGH A SMALL ALGORITHM IT WILL NEVER MAKE A "MISTAKE" AND THE ONLY WAY FOR AN OPPONENT TO WIN IS TO START RIGHT (CORRECT NUMBER OF CHIPS PRODUCING A PLAYER 1 ADVANTAGE) SO THAT THE COMPUTER WILL ALWAYS BE FORCED TO MOVE SO THAT THE ADVANTAGE IS RETURNED TO NEUTRAL AND NOT HIS (PLAYER 2).

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. E2

☐ 4004 ☐ 4040 ☒ 8008 ☐ 8080 (use additional sheets if necessary)

Program Title

NIM (A computer game)

Function

The object of the game is to select the last counter from the last pile. As each game starts, the computer will load each pile with a quasi-random number of counters, checked to insure that if you play exactly right you will win.

Required Hardware

Intellec 8 Mod 8 with standard TTY connected to ports 0, 1, and 8H.

Required Software

None.

Input Parameters

With program loaded jump to 0100H to run.

Use TTY to input 2 characters for each player turn.

Output Results

TTY will show
Number of counters in each pile and game status.

Registers Modified: All	**Assembler/Compiler Used:** Intellec 8 Mod 8 Macro Assembler Ver 2.0
RAM Required: 260H	**Programmer:** B. Weston
ROM Required: None	**Company:** Transcom Inc.
Maximum Subroutine Nesting Level: 4	**Address:** 580 Spring Street Windsor Locks, Conn. 06096

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel®

Ref. No. E3

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

BLACKJACK

Function

INTELLEC/8 PLAYS BLACKJACK WITH OPERATOR

Required Hardware

INTELEC/8 MOD 80
TELETYPE

Required Software

Input Parameters

RESPONSES TO INITIAL QUESTIONS ARE TYPED IN USING A "RETURN" AS A TERMINATING CHARACTER

WHILE PLAYING THE GAME, 'H' MEANS "HIT", AND 'S' MEANS "STICK"?

Output Results

BLACKJACT IS PLAYED, AND THE RESULTS OF EACH GAME ARE PRINTED.

CONTINUOUS 'BELL' CHARACTERS AT ANY TIME INDICATE THAT THE PROGRAM IS "SHUFFLING" THE CARDS.

Registers Modified: ALL	**Assembler/Compiler Used:** 5
RAM Required: APROX. 700 HEX	**Programmer:** 8080 MACRO ASSEMBLER, VER.3.0
ROM Required:	**Company:** ALAN ROSENBAUM
Maximum Subroutine Nesting Level:	**Address:** General Microwave Corp. 155 Marine St. - Farmingdale NY

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other_____ (use additional sheets if necessary)

Program Title

THE WORD GAME

Function

Player No. 1 loads a 16-word dictionary with words of his choice.
The computer then selects one of these "at random"
Player No. 2 then attempts to ascertain the word chosen by
successive "guesses" on the teletype. The computer calculates
the number of letters correct in each guess.

Required Hardware

TTY on PORT Ø - -1

Required Software

Input Parameters

Initially 16 words of up to 16 letters in length are input via the
teletype and stored in RAM
(Locations 9ØØ - 1ØØØH)
Thereafter each guess (input via teletype) is stored in 1Ø2Ø → 1Ø3Ø

Output Results

The computer calculates and outputs via the teletype the number of
letters correct, and in the right place, in each guess, followed
by the number of letters correct and not in the right place, if any.

Registers Modified: ALL	Programmer: J. M. Milby
RAM Required: ½K for Program Approx. ½K for Storage Buffer	Company: B.O.C. Ltd. Small Scale Automation
ROM Required:	Address: 2 Morris Rd.
Maximum Subroutine Nesting Level:	City: Daventry, Northhamptonshire
Assembler/Compiler Used: Intellec 8/Mod80 Macro Assembler V.2.0	State: ENGLAND

THE COMPUTER SELECTS ONE WORD FROM THE SIXTEEN WORD DICTIONARY PREVIOUSLY LOADED BY PLAYER NUMBER ONE .

PLAYER NUMBER TWO NOW ATTEMPTS TO ASCERTAIN WHICH WORD THE COMPUTER HAS SELECTED BY AS FEW GUESSES AS POSSIBLE.

A CHOSEN WORD OF , SAY , CATAPULT WOULD ,PERHAPS, GIVE RISE TO PRINTOUT OF THE FORM:-

ABCDE. 0,2
FGHIJKL. 1,0
MNOPQRST. 1,1
UVWXYZ. 0,1
ABC. 0,2
CAD. 2,0
ETCETERA UNTIL
CATAPULT. 8,0*

AFTER THE CORRECT WORD HAS BEEN INPUT THE COMPUTER SELECTS AGAIN FROM THE SIXTEEN WORD DICTION ARY.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. E5

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

GAMBOL

Function

A demonstration game for the Intellec 8 Mod 80 based on the throwing of two dice.

Required Hardware

Intellec 8 Mod 80 (imm 8/84A), teletype on port 0 and 1

Required Software

System Monitor Ver. 2.0

Input Parameters

Operators guess as to result of next throw, input on demand via teletype.

Output Results

Result of throw, operators number of chips. Message on conclusion of game.

Registers Modified: All	**Assembler/Compiler Used:** 8080 Macro Assembler, Vers. 3.0
RAM Required: 5 bytes	**Programmer:** R. E. Hendtlass
ROM Required: 623 bytes	**Company:** Applied Physics Dept., R.M.I.T.
Maximum Subroutine Nesting Level: 2	**Address:** 124 La Trobe Street, Melbourne, Australia 3000

98-034B

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. E6

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

Mastermind

Function

A game of logic

RULES

The computer sets up a 4-digit code. Each digit is in the range 0 to 5. The game is to break the code by typing in a series of guesses. For each guess, the program returns the following information: an "*" for each correct digit in correct position, and a "." for each correct digit in a wrong position. For example, if the computer sets up 5025 and the player enters 3215, the program will type *

Required Hardware

Intellec 8, Mod 80, TTY.

Required Software

TTY input and output routines in the resident monitor, version 1.0.

Input Parameters

Output Results

Registers Modified: All	**Assembler/Compiler Used:** PL/M1, version 1.3
RAM Required: 110H	**Programmer:** M. Cheeseman
ROM Required: 5D0H	**Company:** AERE HARWELL
Maximum Subroutine Nesting Level: 3	**Address:** Oxfordshire, ENGLAND

```
00001  1
00002  1   /*REF. NO. E6 */
00003  1   /*PROGRAM TITLE MASTERMIND */
00004  1
00005  1
00006  1   /* PROGRAM MASTERMIND. M. CHEESEMAN AUG 1975 */
00007  1
00008  1   20: DECLARE (STARS,DOTS,TRYS,I,J) BYTE ,XN ADDRESS;
00009  1       DECLARE (TEMP,SETUP) (4) BYTE;
00010  1       DECLARE BUFFER(72) BYTE;
00011  1       DECLARE M1 DATA ('*MASTERMIND*',0CH,0AH,
00012  1       0AH,'DO YOU KNOW HOW TO PLAY',3FH,' :');
00013  1       DECLARE M21 DATA (0AH,0AH,'/THE COMPUTER SETS UP A ',
00014  1       '4-DIGIT CODE./',0DH,0AH,'/EACH DIGIT IS IN THE RANGE',
00015  1       ' 0 TO 5.  /',0DH,0AH,'/THE GAME ISSTO BREAK THE CODE',
00016  1       ' BY    /',0DH,0AH,'/TYPING IN A SERIES OF GUESS',
00017  1       'FOR  /',0DH,0AH,'/EACH GUESS THE PROGRAM RETURNS ',
00018  1       'THE   /',0DH,0AH,'/FOLLOWING INFORMATION: AN * FOR ',
00019  1       'EACH/',0DH,0AH);
00020  1       DECLARE M22 DATA ('/CORRECT DIGIT IN CORRECT POSITION',
00021  1       ',  /',0DH,0AH,'/AND A  . FOR EACH CORRECT DIGIT IN A ',
00022  1       '/',0DH,0AH,'/WRONG POSITION. E.G. IF THE COMPUTER/',
00023  1       0DH,0AH,'/SETS UP 5025 AND THE PLAYER ENTERS  /',0DH,
00024  1       0AH,'/3215 , THE PROGRAM WILL TYPE *.   /',0DH,0AH);
00025  1       DECLARE M3 DATA (0AH,'PLEASE TYPE A NUMBER FOR THE '
00026  1       ,'RANDOM ROUTINE : ');
00027  1       DECLARE M4 DATA (0DH,0AH,0AH,'    /');
00028  1       DECLARE M5 DATA (0DH,0AH,0AH,07H,'CORRECT. ',
00029  1       'YOU TOOK ',07H);
00030  1       DECLARE M6 DATA (' ATTEMPTS.',0DH,0AH,'ANOTHER GAME',3FH,' :');
00031  1       DECLARE M7 DATA (0AH,0AH,0AH,0AH,0AH);
00032  1       DECLARE M8 DATA ('....');
00033  1       DECLARE M9 DATA ('****');
00034  1
00035  1      TTYIN:PROCEDURE BYTE;
00036  2       GOTO 3803H;
00037  2       END TTYIN;
00038  1
00039  1      TTYOUT:PROCEDURE(CHAR);
00040  2       DECLARE CHAR BYTE;
00041  2       GOTO 3809H;
00042  2       END TTYOUT;
00043  1
00044  1      RANDOM:PROCEDURE;
00045  2       DECLARE I BYTE;
00046  2            DO I=0 TO 3;
00047  2            XN=XN*107;
00048  3            TEMP(I)=XN/10923;
00049  3            END;
00050  2       END RANDOM;
00051  1
00052  1      MESSAGE:PROCEDURE(N,PTR);
00053  2       DECLARE (I,N) BYTE;
00054  2       DECLARE PTR ADDRESS,CHAR BASED PTR BYTE;
00055  2       IF N=0 THEN
00056  2            DO;
00057  2            CALL TTYOUT(CHAR(0));
00058  3            RETURN;
00059  3            END;
00060  2            DO I=0 TO N;
```

```
00061  2            CALL TTYOUT(CHAR(I));
00062  3            END;
00063  2         END MESSAGE;
00064  1       READ:PROCEDURE BYTE;
00065  2         DECLARE (CHAR,I) BYTE;
00066  2         I,CHAR=0;
00067  2              DO WHILE (CHAR<>0DH) AND (I<72);
00068  2    L1:       CHAR=TTYIN AND 7FH;
00069  3              CALL TTYOUT(CHAR);
00070  3              /* CTRL K */
00071  3              IF CHAR=0BH THEN
00072  3                   DO;
00073  3                   CALL TTYOUT(23H);
00074  4                   IF I>0 THEN I=I-1;
00075  4                   GOTO L1;
00076  4                   END;
00077  3              /* CTRL L */
00078  3              IF CHAR=0CH THEN
00079  3                   DO;
00080  3                   CALL MESSAGE(LAST(M4),.M4);
00081  4                   I=0;
00082  4                   GOTO L1;
00083  4                   END;
00084  3              BUFFER(I)=CHAR AND 0FH;
00085  3              I=I+1;
00086  3              END;
00087  2         RETURN BUFFER(0);
00088  2         END READ;
00089  1
00090  1         /* START MAIN PROGRAM PROGRAM */
00091  1
00092  1
00093  1         CALL MESSAGE(LAST(M1),.M1);
00094  1         IF READ=0FH THEN
00095  1              DO;
00096  1              CALL MESSAGE(LAST(M21),.M21);
00097  2              CALL MESSAGE(LAST(M22),.M22);
00098  2              END;
00099  1           CALL MESSAGE(LAST(M3),.M3);
00100  1           I=READ;
00101  1
00102  1           I=0; XN=1;
00103  1           /* EVALUATE RANDOM NUMBER SEED */
00104  1    L3:    IF BUFFER(I)=0DH THEN GOTO L5;
00105  1           XN=XN*10+(15 AND BUFFER(I));
00106  1           I=I+1;
00107  1           GOTO L3;
00108  1           /* SET UP THE GAME */
00109  1    L5:    CALL RANDOM;
00110  1           TRYS,STARS=0;
00111  1                DO WHILE STARS<4;
00112  1                     DO I=0 TO 3;
00113  2                     SETUP(I)=TEMP(I);
00114  3                     END;
00115  2                CALL MESSAGE(LAST(M4),.M4);
00116  2                /* READ THE ATTEMPT */
00117  2                I=READ;
00118  2                TRYS=TRYS+1;
00119  2                STARS,DOTS=0;
00120  2                     /* SCORE THE ATTEMPT */
```

```
00121  2                  DO I=0 TO 3;
00122  2                  IF BUFFER(I)=SETUP(I) THEN
00123  3                       DO;
00124  3                       STARS=STARS+1;
00125  4                       BUFFER(I)=0FFH;
00126  4                       SETUP(I)=0F0H;
00127  4                       END;
00128  3                   END;
00129  2                   DO I=0 TO 3;
00130  2                       DO J=0 TO 3;
00131  3                       IF BUFFER(I)=SETUP(J) THEN
00132  4                            DO;
00133  4                            DOTS=DOTS+1;
00134  5                            BUFFER(I)=0FFH;
00135  5                            SETUP(J)=0F0H;
00136  5                            END;
00137  4                        END;
00138  3                   END;
00139  2              CALL TTYOUT(0);
00140  2              CALL TTYOUT(0);
00141  2              /* OUTPUT THE SCORE */
00142  2              IF STARS<>0 THEN CALL MESSAGE(STARS-1,.M9);
00143  2              IF DOTS<>0 THEN CALL MESSAGE(DOTS-1,.M8);
00144  2              END;
00145  1         CALL MESSAGE(LAST(M5),.M5);
00146  1         /* >9 TRYS IS NOT GOOD ENOUGH */
00147  1         IF TRYS>9 THEN TRYS=21H;
00148  1         ELSE TRYS=TRYS+30H;
00149  1         CALL TTYOUT(TRYS);
00150  1         CALL MESSAGE(LAST(M6),.M6);
00151  1         IF READ=9 THEN
00152  1              DO;
00153  1              CALL MESSAGE(LAST(M7),.M7);
00154  2              GOTO L5;
00155  2              END;
00156  1         GOTO 3800H;    /* RETURN TO MONITOR */
00157  1         EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. E7

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 (use additional sheets if necessary)

Program Title

MAZE

Function

This is a game program to generate random mazes.

Required Hardware

8080 System. A line printer can be used, but it is not required.

Required Software

INTELLECT Monitor. (*Any version*).
TTY Input (CI), TTY Output (CO), and List Output (LO).

Input Parameters

Keyboard inputs (*See sample run*).

Output Results

The Maze is printed on the device selected as the LIST device.

Registers Modified: ALL	**Assembler/Compiler Used:** 8080 MACRO ASSEMBLER VER 2.0
RAM Required: 9EE (*HEX*)	**Programmer:** C Vincent Phillips
ROM Required: NONE	**Company:** ALKON CORPORATION
Maximum Subroutine Nesting Level:	**Address:** 5329 N HIGH ST. COLUMBUS, OHIO 43214

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other_____ (use additional sheets if necessary)

Program Title

LIFE, GAME OF

Function

Demonstration of Biological Growth on a two-dimensional plane by utilizing cellular Automata Model. (ref: Scientific American, Oct. 1970).

Required Hardware

Intellec/MDS System & Datapoint 3300 CRT

Required Software

Intellec/MDS Monitor I/O

Input Parameters

(X,Y) coordinates of initial cell pattern, entered via CRT keys:

↑, ↓, →, ←, CR, LF, SPACE, AND '*!

Input is terminated with CTRL-C

Output Results

Successive generations of cell organism, based on initial pattern and reproduction laws. Very entertaining.

Registers Modified: ALL	Programmer: K. Burgett
RAM Required: ~ zk bytes	Company: Dharma Systems
ROM Required:	Address:
Maximum Subroutine Nesting Level: 8 bytes	City: San Jose
Assembler/Compiler Used: PL/M, Ver. 2.0	State: California

```
00001  1
00002  1   /*REF. NO. E8*/
00003  1   /*PROGRAM NAME GAME OF LIFE*/
00004  1   200H:
00005  1
00006  1   /* GAME OF LIFE */
00007  1
00008  1   DECLARE BELL LITERALLY '7';
00009  1   DECLARE CR LITERALLY '0DH';
00010  1   DECLARE LF LITERALLY '0AH';
00011  1   DECLARE DOWN LITERALLY '0BH';
00012  1   DECLARE UP LITERALLY '1AH';
00013  1   DECLARE LEFT LITERALLY '19H';
00014  1   DECLARE RIGHT LITERALLY '18H';
00015  1   DECLARE ERASE$EOF LITERALLY '1FH';
00016  1   DECLARE HOME LITERALLY '1DH';
00017  1   DECLARE FALSE LITERALLY '0';
00018  1   DECLARE TRUE LITERALLY '0FFH';
00019  1   DECLARE BREAK$KEY LITERALLY '5AH';
00020  1   DECLARE ARRAY$SIZE LITERALLY '216';
00021  1   DECLARE POPULATION(ARRAY$SIZE) BYTE;
00022  1   DECLARE NEXT$GENERATION(ARRAY$SIZE) BYTE;
00023  1   DECLARE FOREVER LITERALLY 'WHILE 1';
00024  1   DECLARE POPULATION$COUNT ADDRESS;
00025  1   DECLARE GENERATION ADDRESS;
00026  1   DECLARE MASK DATA (128,64,32,16,8,4,2,1);
00027  1   DECLARE MS0 DATA ('GAME OF LIFE, GENERATION ');
00028  1   DECLARE MS1 DATA ('POPULATION ');
00029  1
00030  1   CI:
00031  1     PROCEDURE BYTE;
00032  2       DECLARE IOCI LITERALLY '0F803H';
00033  2
00034  2       GO TO IOCI;
00035  2     END CI;
00036  1
00037  1   CO:
00038  1     PROCEDURE (CHAR);
00039  2       DECLARE CHAR BYTE;
00040  2       DECLARE IOCO LITERALLY '0F809H';
00041  2
00042  2       GO TO IOCO;
00043  2     END CO;
00044  1
00045  1   PRINT$STRING:
00046  1     PROCEDURE (PNAME,PNLST);
00047  2       DECLARE (PNAME,PNLST,PNI) ADDRESS;
00048  2       DECLARE (CHAR BASED PNAME) BYTE;
00049  2
00050  2       DO PNI = 0 TO PNLST;
00051  2         CALL CO(CHAR(PNI));
00052  3       END;
00053  2     END PRINT$STRING;
00054  1   DECIMAL$PRINT:
00055  1     PROCEDURE (VALUE);
00056  2       DECLARE VALUE ADDRESS;
00057  2       DECLARE POWER ADDRESS;
00058  2
00059  2       POWER = 10000;
00060  2       DO WHILE POWER > 0;
```

```
00061  2          IF VALUE >= POWER THEN
00062  3          CALL CO(LOW((VALUE/POWER) MOD 10) + '0');
00063  3        POWER = POWER/10;
00064  3       END;
00065  2    END DECIMALSPRINT;
00066  1
00067  1    CSTS:
00068  1      PROCEDURE BYTE;
00069  2        DECLARE IOCSTS LITERALLY '0F812H';
00070  2
00071  2        GO TO IOCSTS;
00072  2      END CSTS;
00073  1
00074  1    BREAK:
00075  1      PROCEDURE BYTE;
00076  2
00077  2          IF NOT CSTS THEN RETURN FALSE;
00078  2          IF (CI AND 7FH) <> BREAK$KEY THEN RETURN FALSE;
00079  2          RETURN TRUE;
00080  2       END BREAK;
00081  1
00082  1    PROCREATE:
00083  1      PROCEDURE (X,Y);
00084  2        DECLARE (X,Y) BYTE;
00085  2        DECLARE (WORD,PLACE) BYTE;
00086  2
00087  2        IF Y >= 24 THEN RETURN;
00088  2        IF X >= 72 THEN RETURN;
00089  2        WORD = SHL(Y,3) + Y + (ROR(X,3) AND 1FH);
00090  2        PLACE = MASK(X AND 7);
00091  2        NEXT$GENERATION(WORD) = NEXT$GENERATION(WORD) OR PLACE;
00092  2        POPULATION$COUNT = POPULATION$COUNT + 1;
00093  2      END PROCREATE;
00094  1
00095  1    CENSUS:
00096  1      PROCEDURE (X,Y) BYTE;
00097  2        DECLARE (X,Y) BYTE;
00098  2
00099  2        IF Y >= 24 THEN RETURN FALSE;
00100  2        IF X >= 72 THEN RETURN FALSE;
00101  2        Y = POPULATION(SHL(Y,3)+Y+(ROR(X,3) AND 1FH));
00102  2        X = MASK(X AND 7);
00103  2        IF (Y AND X) = 0 THEN RETURN FALSE;
00104  2        RETURN 1;
00105  2      END CENSUS;
00106  1
00107  1    SETUP:
00108  1      PROCEDURE;
00109  2        DECLARE (X,Y,CHAR) BYTE;
00110  2        DECLARE SIZE BYTE;
00111  2
00112  2        CALL CO(HOME);
00113  2        POPULATION$COUNT = 0;
00114  2        DO SIZE = 0 TO LAST(NEXT$GENERATION);
00115  2          NEXT$GENERATION(SIZE) = 0;
00116  3        END;
00117  2        CALL CO(ERASE$EOF);
00118  2        X,Y = 0;
00119  2        DO WHILE (CHAR=(CI AND 7FH)) <> BREAK$KEY;
00120  2          IF CHAR = CR THEN X = 0;
```

```
00121  3          ELSE
00122  3          IF CHAR = LF AND Y <> 23 THEN Y = Y + 1;
00123  3          ELSE
00124  3          IF CHAR = DOWN AND Y <> 23 THEN Y = Y + 1;
00125  3          ELSE
00126  3          IF CHAR = UP AND Y <> 0 THEN Y = Y - 1;
00127  3          ELSE
00128  3          IF CHAR = LEFT AND X <> 0 THEN X = X - 1;
00129  3          ELSE
00130  3          IF CHAR = RIGHT AND X <> 71 THEN X = X + 1;
00131  3          ELSE
00132  3          IF CHAR = '*' AND X <> 71 THEN
00133  3          DO;
00134  3            CALL PROCREATE(X,Y);
00135  4            X = X + 1;
00136  4          END;
00137  3          ELSE
00138  3          IF CHAR = ' ' AND X <> 71 THEN X = X + 1;
00139  3          ELSE CHAR = BELL;
00140  3          CALL CO(CHAR);
00141  3        END;
00142  2        CALL CO(HOME);
00143  2
00144  2      END SETUP;
00145  1
00146  1    DISPLAY:
00147  1      PROCEDURE;
00148  2        DECLARE SIZE BYTE;
00149  2        DECLARE (X,Y,COLUMN) BYTE;
00150  2
00151  2        DO SIZE = 0 TO LAST(NEXTSGENERATION);
00152  2          POPULATION(SIZE) = NEXTSGENERATION(SIZE);
00153  3          NEXTSGENERATION(SIZE) = 0;
00154  3        END;
00155  2        CALL CO(ERASESEOF);
00156  2        DO Y = 0 TO 23;
00157  2          COLUMN = 71;
00158  3          DO WHILE NOT CENSUS(COLUMN,Y) AND (COLUMN <> 0);
00159  3            COLUMN = COLUMN - 1;
00160  4          END;
00161  3          IF COLUMN <> 0 THEN
00162  3          DO;
00163  3            DO X = 0 TO COLUMN;
00164  4              IF CENSUS(X,Y) THEN CALL CO('*');
00165  5              ELSE CALL CO(' ');
00166  5            END;
00167  4            CALL CO(CR);
00168  4          END;
00169  3          CALL CO(LF);
00170  3        END;
00171  2        CALL PRINTSSTRING(.MS0,LAST(MS0));
00172  2        CALL DECIMALSPRINT(GENERATION);
00173  2        CALL PRINTSSTRING(.MS1,LAST(MS1));
00174  2        CALL DECIMALSPRINT(POPULATIONSCOUNT);
00175  2        CALL CO (HOME);
00176  2      END DISPLAY;
00177  1
00178  1    GENERATE:
00179  1      PROCEDURE;
00180  2        DECLARE (SIZE,X,Y,CELL) BYTE;
```

```
00181  2
00182  2        DO SIZE = 0 TO LAST(NEXT$GENERATION);
00183  2          NEXT$GENERATION(SIZE) = 0;
00184  3        END;
00185  2        POPULATION$COUNT = 0;
00186  2        DO X = 0 TO 71;
00187  2          DO Y = 0 TO 23;
00188  3            CELL = CENSUS(X+1,Y)
00189  4                 + CENSUS(X+1,Y+1)
00190  4                 + CENSUS(X,Y+1)
00191  4                 + CENSUS(X-1,Y+1)
00192  4                 + CENSUS(X-1,Y)
00193  4                 + CENSUS(X-1,Y-1)
00194  4                 + CENSUS(X,Y-1)
00195  4                 + CENSUS(X+1,Y-1);
00196  4              IF (CELL=3) OR (CENSUS(X,Y)+CELL=3) THEN
00197  4              DO;
00198  4                CALL PROCREATE (X,Y);
00199  5              END;
00200  4            END;
00201  3          END;
00202  2          CALL CO(HOME);
00203  2        END GENERATE;
00204  1
00205  1        /* MAIN LOOP */
00206  1
00207  1        DO FOREVER;
00208  1          GENERATION = 1;
00209  2          CALL SETUP;
00210  2          DO WHILE (POPULATION$COUNT <> 0) AND NOT BREAK;
00211  2            CALL DISPLAY;
00212  3            GENERATION = GENERATION + 1;
00213  3            CALL GENERATE;
00214  3          END;
00215  2        END;
00216  1    EOF
NO PROGRAM ERRORS
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. E9

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

NUMBERS

Function

NUMBERS is a game. It generates a random number. The user types in guesses. The computer responds by indicating the guess was too big or too small. If the guess was equal to the random number, the user is congratulated and a new random number is generated. This game was originally done in BASIC. It appeared in "What to do after You Hit Return", a book of computer games.

Required Hardware

It takes a teletype on ports 0 and 1, as the Intellec system does.

Required Software

It is a stand alone program.

Input Parameters

Type in numeric guesses from 1 to 100, terminating each with a carriage return.

Output Results

Error messages for incorrect guesses, and congratulatory messages for correct ones.

Registers Modified: All	**Assembler/Compiler Used:** 8080 Macro Assembler, Ver 3.0
RAM Required: 1K	**Programmer:** Bernard R. Greening
ROM Required: None	**Company:** Peoples Computer Company
Maximum Subroutine Nesting Level: 4 levels	**Address:** P.O. Box 310 Menlo Park, CA 94025

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☒ 8008 ☐ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

KALAH

Function

PLAYS THE GAME OF KALAH AGAINST A USER

Required Hardware

PREFERABLY AN OMRON 8025 CRT TERMINAL, BUT ANY OTHER 8008-BASED COMPUTER WILL DO; $2K_{10}$ BYTES MEMORY FOR PROGRAM; $2K_{10}$ BYTES FOR DISPLAY (OPTIONAL)

Required Software

Input Parameters

THE WITS OF THE USER

Output Results

THE THRILL OF VICTORY;
ELSE
THE AGONY OF DEFEAT;

Registers Modified:	Programmer: ROBERT SILBERSTEIN
RAM Required:	Company: OMRON
ROM Required:	Address: 432 TOYAMA DR
Maximum Subroutine Nesting Level:	City: SUNNYVALE
Assembler/Compiler Used: PL/M 8008 VERSION 3.2	State: CALIFORNIA

```
/*
/*
/*        K   K    AAAA    L          AAAA    H    H
/*        K  K    A    A   L         A    A   H    H
/*        K K     A    A   L         A    A   H    H
/*        KK      A    A   L         A    A   HHHHHH
/*        K K     AAAAAA   L         AAAAAA   H    H
/*        K  K    A    A   L         A    A   H    H
/*        K   K   A    A   LLLLLL    A    A   H    H
/*
/*
/*     D E S C R I P T I O N   O F   T H E   G A M E
/*
/*
/*   KALAH IS PLAYED ON A BOARD WHICH CONSISTS OF FOURTEEN SHALLOW
/*   INDENTATIONS (CALLED 'PITS') ARRANGED AS FOLLOWS:
/*
/*
/*              13    12    11    10    9     8
/*
/*             (3)   (3)   (3)   (3)   (3)   (3)
/*
/*     14 (0)                                      (0) 7
/*
/*             (3)   (3)   (3)   (3)   (3)   (3)
/*
/*              1     2     3     4     5     6
/*
/*
/*   PITS 1 THROUGH 7 BELONG TO ONE PLAYER; PITS 8 THROUGH 14 BELONG
/*   TO THE OTHER.  PITS 7 AND 14 ARE SPECIAL PITS CALLED 'KALAHS'.
/*
/*   AT THE BEGINNING OF THE GAME EACH PIT CONTAINS THREE STONES,
/*   WHILE THE KALAHS ARE EMPTY, AS INDICATED IN THE DIAGRAM ABOVE.
/*   THE OBJECT OF THE GAME IS TO HAVE MORE STONES IN YOUR KALAH THAN
/*   YOUR OPPONENT HAS IN HIS AT THE END OF THE GAME.
/*
/*   AFTER DECIDING WHO GOES FIRST, THE PLAYERS TAKE TURNS MAKING MOVES
/*   ACCORDING TO THE FOLLOWING RULES:
/*
/*     1)  A PLAYER CHOOSES A NON-EMPTY PIT ON HIS OWN SIDE OF THE
/*         BOARD (EXCLUDING HIS KALAH) AND TAKES ALL THE STONES FROM THE
/*         PIT.  HE THEN PUTS ONE OF THESE STONES IN EACH PIT (WITH THE
/*         EXCEPTION OF HIS OPPONENT'S KALAH) STARTING WITH THE PIT
/*         IMMEDIATELY FOLLOWING THE EMPTIED ONE IN A COUNTER-
/*         CLOCKWISE DIRECTION UNTIL ALL THE STONES FROM THE CHOSEN PIT
/*         HAVE BEEN REDISTRIBUTED.
/*
/*     2)  IF, IN REDISTRIBUTING STONES ACCORDING TO RULE 1) THE LAST PIT
/*         TO RECEIVE A STONE IS A PLAYER'S OWN KALAH, HE GETS ANOTHER
/*         TURN.
/*
/*     3)  IF THE LAST PIT TO RECEIVE A STONE MEETS ALL THE FOLLOWING
/*         CONDITIONS, A CAPTURE OCCURS.
/*
/*         A)  IT IS ON THE PLAYER'S OWN SIDE OF THE BOARD,
/*
/*         B)  IT IS EMPTY, AND
/*
/*         C)  THE OPPONENT'S PIT DIRECTLY OPPOSITE IT ON THE BOARD
/*             IS NOT EMPTY.
```

```
/*
/*          WHEN A CAPTURE OCCURS, THE PLAYER TAKES THE CAPTURED STONES
/*          FROM HIS OPPONENTS PIT AS WELL AS HIS OWN CAPTURING STONE
/*          AND PLACES THEM IN HIS KALAH.
/*
/*    THE GAME IS OVER WHEN EITHER SIDE OF THE BOARD IS EMPTY.  AT THIS
/*    POINT ANY STONES REMAINING ON A PLAYER'S SIDE OF THE BOARD ARE
/*    PLACED IN THAT PLAYER'S KALAH.
/*
/*    THE PLAYER HAVING MORE STONES IN HIS KALAH WINS.
/*
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☒ 8008 ☐ 8080 ☐ 8048 ☐ 8085 ☐ Other_____ (use additional sheets if necessary)

Program Title

An Adaptive Game Program

Function

This program is a primitive example of a computer learning by experience. The program plays two variations of the game of NIM: Last One Wins and Last One Loses. The player and the machine alternate in taking 1, 2, or 3 "eggs" from a "basket". The contestant who takes the last "egg" wins or loses, depending on the game being played.

Required Hardware

8008, 540 bytes of memory, including 18 bytes of RAM,
Console input/output

Required Software

None.
Monitor I/O routines may be used if available.

Input Parameters

Console Input.

Output Results

Console Output.

Registers Modified: All	Programmer: C. E. Ohme
RAM Required: 18 bytes	Company: OHME
ROM Required: 523 bytes--491 bytes if standard I/O routines are used.	Address: 44750 Winding Lane
Maximum Subroutine Nesting Level: 3	City: Fremont
Assembler/Compiler Used: 8008 Macro Assembler Ver. 2.0	State: California 94538 (415)657-8326

THE GAMES OF
LAST ONE WINS AND
LAST ONE LOSES

THESE TWO GAMES ARE VARIATIONS OF THE GAME OF NIM, DIFFERING ONLY IN THE WAY THE WINNER IS DETERMINED. THEY BEGIN WITH A "BASKET" CONTAINING A RANDOM NUMBER(11-18) OF "EGGS". TWO PLAYERS ALTERNATE TAKING 1, 2, OR 3 "EGGS" FROM THE "BASKET". THE PLAYER REMOVING THE LAST "EGG" FROM THE "BASKET" EITHER WINS OR LOSES THE GAME, AS INDICATED BY THE NAME OF THE GAME BEING PLAYED.

TO USE THIS PROGRAM, THE CONSOLE OPERATOR CHOOSES BETWEEN THE TWO GAMES BY SELECTING THE LOCATION (WINS OR LOSES) AT WHICH EXECUTION BEGINS. THE COMPUTER THEN DISPLAYS THE NUMBER OF EGGS IN THE BASKET AND WAITS FOR THE OPERATOR TO INDICATE HIS MOVE(IN ALTERNATE GAMES THE COMPUTER MOVES FIRST). THE BASKET IS AGAIN DISPLAYED, THE COMPUTER MAKES ITS MOVE, ETC, UNTIL THE GAME IF COMPLETED.

FOR EITHER GAME, THE COMPUTER USES A GAME STRATEGY EMBODIED IN A MATRIX IN MEMORY. THE MATRIX IS POTENTIALLY MODIFIED BY THE OUTCOME OF EACH GAME. THIS PROCESS EVENTUALLY LEADS TO THE OPTIMUM STRATEGY FOR THE VARIATION OF THE GAME CURRENTLY BEING PLAYED.

THE OPTIMUM STRATEGIES FOR THE TWO VARIATIONS OF THE GAME ARE QUITE DIFFERENT. SINCE A SINGLE MATRIX IS USED TO RECORD THE COMPUTER'S STRATEGY, THE OPERATOR CAN IMPROVE HIS ODDS BY CHANGING GAMES WHEN THE COMPUTER GETS TOO SMART.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. No. E12

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

Match Game

Function

See additional sheet

Required Hardware

Intellec 8, TTY on port 0 and 1

Required Software

Routines in resident monitor, version 1.0, TTY input, TTY output, carriage return and linefeed, convert ASCII to HEX

Input Parameters

Program accepts ASCII characters on port 0. Only the characters "1", "2" or "3" will cause the program to continue. Any other character will initiate a "fault" sequence.

Output Results

All output results are displayed on the TTY. The actual number of matches remaining is indicated by that number of "/" characters. The program echoes the character typed by the player, as well as printing the number which the program has subtracted. Three messages are printed on the TTY:

player wins	"YOU WIN"
program wins	"I WIN - SORRY"
invalid character typed	"NO CHEATING"

Registers Modified: A,B,C,D,E,H,L	**Assembler/Compiler Used:** 8080 Macro Assembler, version 2.0
RAM Required: 5 bytes (exclusive of program)	**Programmer:** Michael M. Dodd
ROM Required: None	**Company:**
Maximum Subroutine Nesting Level: 2	**Address:** 291 Waples Estates Fairfax, Va. 22030

98-034C

Michael M. Dodd
234 Waples Estates
Fairfax, Virginia 22030

PROGRAM TITLE: Match Game

FUNCTION: The purpose of this program is to match a player against the processor, in a test of logic. Fifteen matches are provided at the start of the game; each player may remove 1, 2, or 3 matches per turn. The player who must remove the last match loses the game.

The program prints 15 matches on the TTY and awaits the player's move; the player indicates that he wants to remove 1, 2, or 3 matches by pressing the appropriate key on the TTY. Upon receipt of the number from the player, the program subtracts that number from the matches, prints the remaining matches, prints and subtracts its own number, and prints the remaining matches again. The program then awaits the player's next move, at which time the sequence begins again.

If the program must remove the last match, it prints a "lose" message on the TTY and then prints 15 matches, in preparation for the next game.

If the player must remove the last match, the program prints a "win" message on the TTY and then prints 15 matches, in preparation for the next game.

If, at any time, the player presses any key except 1, 2, or 3, the program prints a "fault" message on the TTY and then prints the same number of matches the player had before he pressed the wrong key.

Since, after playing the game several times, it becomes apparent that there is only one sequence the player can use to win, a "randomizer" was built into the program. The randomizer routine is in effect only when it is the program's turn and there are ten or more matches remaining. It causes the program to sometimes remove a number different from what might be expected. This performs the dual function of making the player think more about his next move and also of giving the player a better chance to win the game.

Unless otherwise directed by the randomizer, the program will remove a certain number of matches, depending on the number remaining, as shown below.

Michael M. Dodd
234 Waples Estates
Fairfax, Virginia 22030

PROGRAM TITLE: Match Game

FUNCTION
(Continued)

Matches Remaining	Removed by Program
14	1
13	1
12	3
11	2
10	1
9	1
8	3
7	2
6	1
5	1
4	3
3	2
2	1
1	0 (Lose)

Under some conditions, as directed by the "ODD" subroutine, the program will take 2, instead of the number indicated, for 10 - 14 matches remaining.

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

intel

Ref. No. E13

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000

(use additional sheets if necessary)

Program Title

Maze

Function

User attempts to find his way out of an invisible (i.e. dark) maze.

Required Hardware

TTY on ports 0 and 1.

Required Software

System monitor version 3.0.

Input Parameters

Characters from TTY. Instructions contained in program.

Output Results

Messages from program.

Registers Modified:	Assembler/Compiler Used:
All	8080 Macro Assembler version 3.0
RAM Required:	**Programmer:**
0000H -- 0300H	J.B. Miller-Smith
ROM Required:	**Company:**
System monitor, Version 3.0	Racal-Milgo Ltd.
Maximum Subroutine Nesting Level:	**Address:** Portman House. Loverock Rd. Reading, Berks., ENGLAND
4	

```
                ;REF. NO. E13
                ;PROGRAM TITLE MAZE
                ;
                ;
                ;
                ;MAZE GAME - MAZE TABLE ADDRESS 40H TO 7FH
                ;CURRENT POSITION HELD IN ADDRESS 80H
                ;PROGRAM STARTS AT ADDRESS 100H
0040                    ORG     0040H

0040 FFFFFF44   DB 0FFH,0FFH,0FFH,44H,50H,48H
0044 5048
0046 404C44FF   DB 40H,4CH,44H,0FFH,0FFH,0FFH
004A FFFF
004C FFFF44FF   DB 0FFH,0FFH,44H,0FFH,0FFH,44H
0050 FF44
0052 54FF58FF   DB 54H,0FFH,58H,0FFH,5CH,50H
0056 5C50
0058 FF54FFFF   DB 0FFH,54H,0FFH,0FFH,0FFH,60H
005C FF60
005E FF545C64   DB 0FFH,54H,5CH,64H,0FFH,0FFH
0062 FFFF
0064 60FFFF68   DB 60H,0FFH,0FFH,68H
0068 FF0F64FF   DB 0FFH,0FH,64H,0FFH
0100                    ORG     0100H
3F73            TI      EQU     3F73H   ;SYSMON I/P ROUTINE
3809            CO      EQU     3809H   ;   "     O/P     "
0080            POSN    EQU     0080H   ;CURRENT POSITION ADDRESS
0100 313F00     START:  LXI     SP,003FH
0103 3E40               MVI     A,40H   ;SET...
0105 328000             STA     POSN    ;..INITIAL POSITION
0108 218401             LXI     H,MSG0
010B CD7801             CALL    OUTPT   ;O/P INTRO' MSG.
010E CD733F     ENTRY:  CALL    TI      ;AWAIT I/P
0111 FE4C               CPI     'L'     ;LOST ?
0113 CA0001             JZ      START   ;YES, RESTART
0116 FE4E               CPI     'N'     ;NORTH ?
0118 C22001             JNZ     SOUTH   ;NO
011B 0E00               MVI     C,00H   ;YES,SET TABLE DISPLACEMENT..
011D C34701             JMP     MOVE    ;..POINTER, & CHECK MOVEMENT
0120 FE53       SOUTH:  CPI     'S'     ;REPEAT FOR S,E, AND W
0122 C22A01             JNZ     EAST
0125 0E01               MVI     C,01H
0127 C34701             JMP     MOVE
012A FE45       EAST:   CPI     'E'
012C C23401             JNZ     WEST
012F 0E02               MVI     C,02H
0131 C34701             JMP     MOVE
0134 FE57       WEST:   CPI     'W'
```

```
0136 C23E01          JNZ     ERROR   ;NOT N,S,E,OR W
0139 0E03            MVI     C,03H
013B C34701          JMP     MOVE
013E 214F02  ERROR:  LXI     H,MSG1  ;OUTPUT...
0141 CD7801          CALL    OUTPT   ;..ERROR MSG.
0144 C30E01          JMP     ENTRY   ;TRY MORE I/P
0147 2A8000  MOVE:   LHLD    POSN    ;GET CURRENT POSITION
014A 7D              MOV     A,L     ;ADD IN...
014B 81              ADD     C       ;..DISPLACEMENT...
014C 6F              MOV     L,A     ;..AND CLEAR H OF...
014D 2600            MVI     H,00H   ;..ANY RUBBISH
014F 7E              MOV     A,M     ;GET TABLE ENTRY
0150 FEFF            CPI     0FFH    ;IS IT WALL MARKER ?
0152 CA6601          JZ      WALL    ;YES
0155 FE0F            CPI     0FH     ;IS IT EXIT MARKER ?
0157 CA6F01          JZ      EXIT    ;YES
015A 328000          STA     POSN    ;NEITHER,STORE NEW POSN.
015D 217802          LXI     H,MSG2
0160 CD7801          CALL    OUTPT   ;O/P "OK" MSG.
0163 C30E01          JMP     ENTRY   ;NEXT TRY
0166 218002  WALL:   LXI     H,MSG3
  69 CD7801          CALL    OUTPT   ;O/P "WALL" MSG.
  6C C30E01          JMP     ENTRY   ;TRY AGAIN
016F 219002  EXIT:   LXI     H,MSG4
0172 CD7801          CALL    OUTPT   ;O/P "CONGRAT'S" MSG.
0175 C30001          JMP     START   ;RESTART NEW GAME
                     ;
0178 7E      OUTPT:  MOV     A,M     ;GET CHARACTER
0179 FE00            CPI     00H     ;END MARKER ?
017B C8              RZ              ;YES, RETURN
017C 4F              MOV     C,A     ;COPY TO C
017D CD0938          CALL    CO      ;DO O/P
0180 23              INX     H       ;POINT TO NEXT CHAR.
0181 C37801          JMP     OUTPT   ;LOOP
                     ;
0184 0D0A0A0A MSG0:  DB 0DH,0AH,0AH,0AH,0AH
0188 0A
0189 594F5520 DB 'YOU ARE IN A DARK MAZE (WITH A COMPASS !).
018D 41524520
0191 494E2041
0195 20444152
0199 4B204D41
019D 5A452028
01A1 57495448
01A5 20412043
01A9 4F4D5041
01AD 53532021
  31 292E
01B3 0D0A    DB 0DH,0AH
01B5 4645454C DB 'FEEL YOUR WAY OUT BY SPECIFYING MOVES NORTH,
```

```
01B9 20594F55
01BD 52205741
01C1 59204F55
01C5 54204259
01C9 20535045
01CD 43494659
01D1 494E4720
01D5 4D4F5645
01D9 53204E4F
01DD 5254482C
01E1 0D0A       DB 0DH,0AH
01E3 534F5554   DB 'SOUTH, EAST, & WEST (USING INITIAL LETTERS).'
01E7 482C4541
01EB 53542C20
01EF 26205745
01F3 53542028
01F7 5553494E
01FB 4720494E
01FF 49544941
0203 4C204C45
0207 54544552
020B 53292E
020E 0D0A       DB 0DH,0AH
0210 49462048   DB 'IF HOPELESSLY LOST, TYPE "L" - GAME RESTARTS.'
0214 4F50454C
0218 4553534C
021C 59204C4F
0220 53542C20
0224 54595045
0228 20224C22
022C 202D2047
0230 414D4520
0234 52455354
0238 41525453
023C 2E
023D 0D0A       DB 0DH,0AH
023F 474F4F44   DB 'GOOD LUCK !'
0243 204C5543
0247 4B2021
024A 0D0A0A0A   DB 0DH,0AH,0AH,0AH
024E 00         DB 00H
024F 2020204E   MSG1:   DB '   N,S,E,W (OR L) ONLY PLEASE - RETYPE'
0253 2C532C45
0257 2C572028
025B 4F52204C
025F 29204F4E
0263 4C592050
0267 4C454153
026B 45202D20
026F 52455459
```

```
0273 5045
0275 0D0A00    DB 0DH,0AH,00H
0278 2020204F MSG2:    DB '    OK',0DH,0AH,00H
027C 4B0D0A00
0280 07072020 MSG3:     DB 07H,07H,'    --WALL--',0DH,0AH,00H
0284 202D2D57
0288 414C4C2D
028C 2D0D0A00
0290 0D0A0A43 MSG4:    DB 0DH,0AH,0AH,'CONGRATULATIONS !'
0294 4F4E4752
0298 4154554C
029C 4154494F
02A0 4E532021
02A4 0D0A594F DB 0DH,0AH,'YOU ARE CLEAR OF THE MAZE.'
02A8 55204152
02AC 4520434C
02B0 45415220
02B4 4F462054
02B8 4845204D
02BC 415A452E
02C0 0D0A0A00 DB 0DH,0AH,0AH,00H
              ;
  00          END
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Dec. 1976
Ref. E15

☐ 4004 ☐ 4040 ☐ 8008 ☑ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

Lewthwaite's game

Function

A 5*5 matrix has cells which are alternately filled with two types of counters (X and O) except for the center cell which is blank. A player and the computer alternate turns by moving their counter toward the blank cell (thereby filling that cell and creating a new blank cell) until the player or the computer has no legal move. Whoever makes the last move wins. A legal move is made from any orthogonal (not diagonal) cell adjacent to the blank cell toward the blank cell.

Required Hardware

MDS 800 & ADDS 580 CRT (or any cursor addressable CRT)

Required Software

Monitor routines

Input Parameters

Row & column of the counter to be moved toward the blank cell.

Output Results

CRT will update the matrix in place and trace the moves.

Registers Modified: All	**Assembler/Compiler Used:** MDS 800 Macro Assembler V2.2
RAM Required: Less than 3K bytes	**Programmer:** Tom McHugh
ROM Required: Monitor	**Company:**
Maximum Subroutine Nesting Level:	**Address:** 3130 N Bartlett Milw. WI. 53211

98-034C

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Dec. 1976

Ref. E16

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080

(use additional sheets if necessary)

Program Title

TIC-TAC-TOE

Function

This game lets two players play against each other, while the machine supervises the game.

Required Hardware

Intellec/MDS system & CRT-console

Required Software

Intellec/MDS monitor I/O

Input Parameters

Keyboard inputs (1-9)
Square to be deleted and square to be filled.

Output Results

The square is drawn on the console with Xs and Os and error messages.

Registers Modified: All	**Assembler/Compiler Used:** 8080Macro assembler V1.0
RAM Required: 7E0H	**Programmer:** Rune Larsen
ROM Required: None	**Company:** The Norwegian telecom. research establishment
Maximum Subroutine Nesting Level:	**Address:** P.O.B. 83 N-2007 Kjeller NORWAY

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. # E17

intel®

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

Bandit, static display version 1.

Function

Game:- (fruit machine). Produces a static display on VDU screen. Selects random numbers to choose 'fruit' from preprogrammed odds. Programme attempts to simulate a fruit machine; as far as possible.

Required Hardware

MDS80 and Hazeltine 1200 VDU; else any system with a VDU and a CO and CI routine. NB Following ASC11 code used with Hazeltine. Space (20H), destructive space (10H), clear screen (0CH), and top of page (0BH). This code may vary with other VDU's.

Required Software

Resident MDS800 system monitor or other suitable CO & CI software.

Input Parameters

Parameters from VDU keyboard:- Money entered as 5 or 10, and CR used to start a game. Held reels are entered as 1, 2 or 3. Other parameters used for clearing held reels and obtaining pay out are permanently written on the VDU screen.

Output Results

Game machine layout drawn on VDU screen. Programme updates display each game with random selection of 'fruit', and amends the number of games left. This operation repeats when initiated by the player for subsequent games, until his money runs out. A display of winning combinations can be requested when the programme is first entered.

Registers Modified: ALL	**Assembler/Compiler Used:** 8080 Macro Assembler
RAM Required: 2000H to 2952H for programme. IFF4H to IFFFH for stack	**Programmer:** P.G.R. Kitson MSc
ROM Required: None	**Company:** Marconi Radar Systems Ltd.
Maximum Subroutine Nesting Level: 3	**Address:** New Parks, Leicester, England

98-034C

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. # E18

intel®

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 (use additional sheets if necessary)

Program Title

Kill the Rotating Bit

Function

Demo Game for Prompt 80
Two versions of the same game are given. One version uses the exclusive-or instruction and is therefore shorter.

Required Hardware

Prompt 80

Required Software

Prompt 80 monitor which does the initialization of Port E8 and E9.

Input Parameters

The bit (or bits) input through Port E9 of the Prompt 80.

Output Results

See Listings for complete instructions

Registers Modified: A, B, H, L, E (2nd program only)	**Assembler/Compiler Used:** MDS 800 Macro VI.I
RAM Required: 45 bytes (2nd program) 25 bytes (1st program)	**Programmer:** Lenard Persin
ROM Required: None	**Company:**
Maximum Subroutine Nesting Level: 0	**Address:** 418 W. Oakridge Rd. Apt 110 Orlando, Fla 32809

98-034C

```
                ;REF. NO. E18
                ;PROGRAM TITLE KILL THE ROTATING BIT
                ;
                ;
                ;
                ; ***************************************************
                ;
                ; LENARD PERSIN              SEPTEMBER 20, 1976
                ;
                ; THIS IS A DEMO PROGRAM FOR THE PROMPT 80. THIS
                ; PROGRAM USES THE EXCLUSIVE-OR INSTRUCTION AND IS
                ; THEREFORE SHORTER THAN THE PREVIOUS VERSION. THE
                ; OBJECT IS TO KILL THE ROTATING BIT. IF YOU
                ; MISS THE  LIT BIT ANOTHER ONE AT THAT SWITCH
                ; POSITION WILL TURN ON, NOW LEAVING YOU TWO BITS
                ; TO DESTORY. IN ORDER TO KILL A BIT TOGGLE THE
                ; SWITCH (PRESS ONE OF THE INPUT SWITCHES AND THEN
                ; THE RST SWITCH) AS THE ROTATING BIT MOVES BY.
                ; DON'T LEAVE THE SWITCH IN THE "ON" POSITION, BE
                ; SURE TO HIT THE PORT RESET SWITCH AFTER EACH
                ; TOGGLE. BEFORE STARTING MAKE SURE ALL THE PORT
                ; SWITCHES ARE IN THE "OFF" (RESET) POSITION.
                ;
                ; IF YOU WIN AND KILL THE BIT TO RESTART MERELY
                ; TOGGLE ONE OF THE SWITCHES TO START A NEW BIT.
                ;
                ; START THE PROGRAM AT 3D00. AS YOU GET BETTER
                ; SPEED UP THE ROTATING BIT AND TRY AGAIN.
                ;
                ; ***************************************************
                ;
3D00                    ORG     3D00H
                ;
                ;       PORT ASSIGNMENTS
                ;
00E9            SWTCH   EQU     0E9H    ; INPUT SWITCHES
00E8            LGHTS   EQU     0E8H    ; OUTPUT LIGHTS
                ;
                ; REGISTER USE
                ;
                ; B-LIGHT PATTERN
                ; H-HIGH ORDER DELAY TIME
                ; L-LOW ORDER DELAY TIME
                ;
                ;       INITIALIZE B REGISTER WITH BLINK PATTERN
                ;
                START:
3D00 0680               MVI     B,80H   ; ONE LIGHT ON
                ;
```

```
                ; ROTATE THE LIGHT PATTERN USING THE DELAY TIME
                ; SET BY THE H AND L REGISTERS.
                ;
                ROTATE:
3D02 218877             LXI     H,7788H ; LOAD SPEED DATA (LOWER THE
                                        ; VALUE THE FASTER THE ROTATING
                                        ; RATE).
                ;
                ;       DELAY LOOP
                ;
                LOOP:
3D05 2B                 DCX     H       ; DECREMENT H & L REGISTER PAIR
                                        ; (DELAY TIME).
3D06 7D                 MOV     A,L     ; CHECK IF H & L = 0.
3D07 B4                 ORA     H       ; "OR" H & L REGISTERS
3D08 C2053D             JNZ     LOOP    ; IF NOT ZERO DECREMENT
                                        ; DELAY TIME BY ONE AND
                                        ; TEST AGAIN.
3D0B 78                 MOV     A,B     ; GET LIGHT PATTERN
3D0C 07                 RLC             ; ROTATE THE LIGHT PATTERN
3D0D 47                 MOV     B,A     ; SAVE NEW LIGHT PATTERN
3D0E 2F                 CMA             ; ADJUST FOR COMPLIMENT
                                        ; OF OUTPUT.
3D0F D3E8               OUT     LGHTS   ; OUTPUT TO LIGHTS
                ;
                ;       READ SWITCHES FOR INPUT PATTERN
                ;
3D11 DBE9               IN      SWTCH   ; READ SWITCHES FOR PATTERN
3D13 2F                 CMA             ; ADJUST FOR COMPLEMENT
                                        ; OF INPUT.
3D14 A8                 XRA     B       ; EXCLUSIVE "OR" SWITCH
                                        ; PATTERN AND LIGHT PATTERN.
3D15 47                 MOV     B,A     ; UPDATE LIGHT PATTERN
3D16 C3023D             JMP     ROTATE  ; JUMP TO "ROTATE" AND OUTPUT
3D00                    END     START
```

MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref. # E19

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 X PROMPT 80 (use additional sheets if necessary)

Program Title

PROMPT PONG

Function

A GAME OF PONG USING THE PROMPT 80

Required Hardware

PROMPT 80

Required Software

PROMPT 80 MONITOR

Input Parameters

TWO PLAYERS ONE DEPRESSING THE "," KEY AND ONE DEPRESSING "3" KEY OF THE PROMPT AT THE CORRECT TIME.

Output Results

SCORE WILL BE DISPLAYED ON THE RIGHT MOST NUMERICAL DISPLAY GROUP OF THE PROMPT 80.

Registers Modified: ALL	**Assembler/Compiler Used:** ISIS 8080 MACRO ASSEMBLER VI.I
RAM Required: SEE LISTING	**Programmer:** BOB A. MC NAMARA LENARD PERSIN
ROM Required: SEE LISTING	**Company:**
Maximum Subroutine Nesting Level: 4	**Address:** 6318 LUZION DR ORLANDO, FLA 32809

98-034C

Ref. # E20

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

REACT

Function

Game to test your reaction time

Required Hardware

MDS, Output console with "bell" that is audible.

Required Software

MAC80 Assember or Cross Assembler

Input Parameters

Finger presses interrupt 7 button ASAP after hearing bell. Occurance of bell is randomized in time between 1 and 2 seconds.

Output Results

At console output get displayed the reaction time in 1/100 sec. units; i.e. if the value is 19 this means 0.19 seconds. Max value or for "jumping the gun" is 0.99 sec.

Registers Modified: All	Programmer: V. Grinich
RAM Required: 3 bytes for VSA	Company: Stanford University
ROM Required: Monitor + 345 bytes	Address: 101 GRL
Maximum Subroutine Nesting Level: 2	City: Stanford
Assembler/Compiler Used: Mac 80	State: California 94305

```
                        ;REF. NO. E20
                        ;PROGRAM TITLE REACT(REACTION TIME)
                        ;
                        ;
                        ;
                        ;REACTION TIME ON ISIS SYSTEM
F809                    CO      EQU     0F809H
F803                    CI      EQU     0F803H
000D                    CR      EQU     0DH
000A                    LF      EQU     0AH
0007                    BELL    EQU     07H
007E                    MASK    EQU     7EH     ;RST 7 AND RST 0 ALLOWED
0038                            ORG     38H     ;RST 7 AREA
0038 C36531             JMP     SRVC7
3100                            ORG     3100H
                        ENTRY:
3100 3E12               MVI     A,12H   ;INIT OF INTERRUPT MASK
3102 D3FD               OUT     0FDH
3104 3E00               MVI     A,00H
3106 D3FC               OUT     0FCH
3108 3E7E               MVI     A,MASK
310A D3FC               OUT     0FCH

310C 310033             LXI     SP,3300H
310F 1E93               MVI     E,LMES1 AND 0FFH

3111 21C131             LXI     H,MES1
3114 4E                 MOV     C,M
3115 CD09F8             CALL    CO
3118 23                 INX     H
3119 1D                 DCR     E
311A C21431             JNZ     RPT1
311D FB                 EI              ;TO CATCH TOO-EARLY STARTERS
311E 21C031             LXI     H,FLAG
3121 36FF               MVI     M,0FFH
3123 CDA231             CALL    CRLF
3126 1E09               MVI     E,09H
                        CNTDN:
3128 214D01             LXI     H,10000/30      ;10000 USEC=10 MILLISEC
                                                ;DELAY LOOP IS 30 USEC
312B CDAD31             CALL    DELAY
312E CDB631             CALL    COASC   ;PRINT OUT 9,8,...,1,0
3131 1D                 DCR     E
3132 F22831             JP      CNTDN
3135 CDA231             CALL    CRLF
3138 211482             LXI     H,10000/30*100  ;1000 MILLISEC
313B CDAD31             CALL    DELAY   ;GET ONE SEC OF DELAY
313E 2ABE31             LHLD    LO
3141 65                 MOV     H,L
```

```
3142 CDAD31             CALL    DELAY
3145 21C031             LXI     H,FLAG
3148 3600               MVI     M,00H
314A 0E07               MVI     C,BELL
314C CD09F8             CALL    CO
314F 1600               MVI     D,0         ;ZERO OUR COUNTER
3151 7A         LUP1:   MOV     A,D
3152 C601               ADI     1
3154 27                 DAA
3155 57                 MOV     D,A
3156 FE99               CPI     99H         ;MAX VAL THAT CAN BE DISPAYED
3158 C25C31             JNZ     MORE
315B 76                 HLT     ;WAIT FOR INT 7 HERE
                MORE:
315C 214D01             LXI     H,10000/30      ;10 MILLISEC
315F CDAD31             CALL    DELAY
3162 C35131             JMP     LUP1
                ;SRVCE SUB-ROUTINE FOR RST 7
                SRVC7:
3165 F5                 PUSH    PSW
3166 C5                 PUSH    B
3167 D5                 PUSH    D
 168 E5                 PUSH    H
3169 21C031             LXI     H,FLAG
316C AF                 XRA     A
316D BE                 CMP     M
316E CA7331             JZ      OK          ;IF FLAG IS ZERO NOT A FALSE START
3171 1699               MVI     D,99H
                OK:
3173 7A                 MOV     A,D
3174 21BE31             LXI     H,LO
3177 7A                 MOV     A,D
3178 E60F               ANI     0FH
317A 07                 RLC
317B 07                 RLC
317C 07                 RLC
317D 07                 RLC
317E 77                 MOV     M,A         ;RANDOMIZE THE START GUN
317F 7A                 MOV     A,D
3180 E6F0               ANI     0F0H        ;CONVERT H.O. NIBLLE TO ASCIVALUE
3182 0F                 RRC
3183 0F                 RRC
3184 0F                 RRC
3185 0F                 RRC
3186 C630               ADI     30H
3188 4F                 MOV     C,A
3189 CD09F8             CALL    CO
 18C 7A                 MOV     A,D
 18D E60F               ANI     0FH;GET L.O.  NIBBLE INTO ASCII VAL NOW
 18F C630               ADI     30H
```

```
3191 4F                 MOV     C,A
3192 CD09F8             CALL    CO
3195 F3                 DI
3196 E1                 POP     H
3197 D1                 POP     D
3198 C1                 POP     B
3199 3E20               MVI     A,20H
319B D3FD               OUT     0FDH
319D F1                 POP     PSW
319E F1                 POP     PSW     ;EXTRA POP TO CLEAR RETURN ADDRESS
                                        ;ON STACK FROM RST 7
319F C31D31             JMP     NWTRY

                ;
                CRLF:
31A2 0E0D               MVI     C,CR
31A4 CD09F8             CALL    CO
31A7 0E0A               MVI     C,LF
31A9 CD09F8             CALL    CO
31AC C9                 RET
                ;
                DELAY:
                                ;SUB ROUT THAT GIVES 30 USEC DELAY FOR
                                ; EVERY UNIT THAT IS IN THE H RP
31AD 2B         MORE1:  DCX     H
31AE E3                 XTHL            ;USE SINCE LONGEST DELAY PER BYTE
31AF E3                 XTHL            ;RESTORE STACK AND H RP
31B0 7C                 MOV     A,H
31B1 B5                 ORA     L       ;IF BOTH H AND L ARE =0 ACC IS =0
31B2 C2AD31             JNZ     MORE1
31B5 C9                 RET
                ;
                COASC:
31B6 7B                 MOV     A,E
31B7 C630               ADI     30H;    CONVERT VALUE IN REG. E TO  '(E)'
31B9 4F                 MOV     C,A
31BA CD09F8             CALL    CO
31BD C9                 RET
                ;
31BE            LO:     DS      2
31C0            FLAG:   DS      1
31C1 41465445   MES1:   DB      'AFTER COUNDOWN (9,8,...,1,0) THERE WILL BE '
31C5 5220434F
31C9 554E444F
31CD 574E2028
31D1 392C382C
 1D5 2E2E2C31
.1D9 2C302920
31DD 54484552
```

```
31E1 45205749
31E5 4C4C2042
31E9 4520
31EB 41203120          DB      'A 1 SEC PAUSE, THEN A BELL. '
31EF 53454320
31F3 50415553
31F7 452C2054
31FB 48454E20
31FF 41204245
3203 4C4C2E20
3207 0D0A              DB      CR,LF
3209 48495420          DB      'HIT INT 7 BUTTON ASAP AND SEE YOUR REACTION'
320D 494E5420
3211 37204255
3215 54544F4E
3219 20415341
321D 5020414E
3221 44205345
3225 4520594F
3229 55522052
322D 45414354
?231 494F4E
 234 2054494D          DB      ' TIME IN TENS OF MILLISECONDS.'
3238 4520494E
323C 2054454E
3240 53204F46
3244 204D494C
3248 4C495345
324C 434F4E44
3250 532E
3252 0D0A              DB    CR,LF
0093            LMES1  EQU     $-MES1
3100                   END     ENTRY
```

Ref. # E21

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

8080 SLOT MACHINE

Function

Simulates 3 Wheel Slot Machine
Described in Book
Game playing with computers
by Donald Spencer

Required Hardware

Written for Intellec 8/MOD 80 with
TTY on Ports Ø and 1 or
CRT on Ports 4 and 5

Required Software

I/O routines in System Monitor Ver. 3.Ø

Input Parameters

(see sample printout)

Output Results

Registers Modified: ALL	Programmer: Mark D. Hansen
RAM Required: 686 hex bytes	Company: ISCO
ROM Required:	Address: P. O. Box 5347
Maximum Subroutine Nesting Level: 4	City: Lincoln
Assembler/Compiler Used: MACRO ASSEMBLER V 3.Ø	State: Nebraska 68505

Ref. # E22

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

MATCH (Resembles Invicta Corp's "Mastermind")

Function

A Game for the Intellec 8/Mod 80 which "Matches" the logical skills of the player against the random "imagination" of the Intellec. The Intellec selects a code consisting of some combination of six letters. The player, aided by feedback information, receives five chances to decipher this code.

Required Hardware

Intellec 8/Mod 80, Teletype on Port 0 and 1

Required Software

Program is self-contained.

Input Parameters

Conversational Mode only: (7-Bit ASCII)
User Input: Some combination of 4 letters selected from: A,B,C,D,E,F

See attached documentation for further information regarding the play of the game.

Output Results

Conversational Mode Only: (7-Bit ASCII)
1) "Welcome," "Ready" Messages,
2) Score, Accumulating Machine "Laughter" at Player's Losses,
3) "Waiting..." Message - Every 55 sec.,
4) Illegal Input Reject Message,
5) Feedback Codes - "$" and "*",
6) Correct Code Shown when Player Loses Game,
7) "You Win", "You Lose" Messages.

Registers Modified: A, B, C, D, E, H, L	Programmer: PATRICK F. CASTELAZ
RAM Required: 50 Bytes (includes 29 for stack)	Company: Marquette University - Engineering
ROM Required: 957 Bytes (1K)	Address: 1515 West Wisconsin Avenue
Maximum Subroutine Nesting Level: 3	City: Milwaukee
Assembler/Compiler Used: Intellec 8 MACRO Assembler	State: Wisconsin 53233

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

MASTERMIND 8080

Function

The computer selects a random code in the range 000000 to 999999. The player attempts to break this code by entering via the keyboard a series of guesses, based on "clues" provided by the computer after each guess.

Required Hardware

INTEL SBC 80/10 with TTY interface.
Alternatively INTELLEC MDS-800 with TTY or CRT interface.

Required Software

Console Input/Output routines resident in the System Monitors.

Input Parameters

To begin program execution, enter'G3C3D'. To restart a game, enter any character via the keyboard, The computer then prompts the player to select: a) the length of code (i.e. number of columns) to be guessed.
-Enter one digit in the range 1 to 6.
b) the range of numbers to be used in the code (0-4 minimum, 0-9 maximum) The computer adjusts the original code appropriately if the full range 0-9 is not selected.
-Enter one digit in the range 4 to 9.
c) The maximum number of guesses allowed.
-Enter two digits in the range 01 to 99.
Program execution may be aborted at any stage by inputting the character 'A'.

Output Results

After each attempt at guessing the code, the computer replies by outputting:

i) the number of digits correct in both value and position.
ii) the number of remaining digits having correct value but incorrect position.

Appropriate messages are outputted when the code is guessed correctly, or when the maximum number of attempts has been reached, as in (c) above, or when aborted.

Registers Modified: ALL	Programmer: C. J. Williams
RAM Required: 39F (HEX) = 927 bytes prog. 20 (HEX) = 32 bytes workspace	Company: Cable & Wireless Ltd.
ROM Required: -	Address: 114 Great Suffolk St.
Maximum Subroutine Nesting Level: 1	City: London, SE1.
Assembler/Compiler Used: ISIS 8080 Macro Assembler V1.1	State: ENGLAND

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/40 ☐8008 ☒8080 ☐8048 ☐8085 ☐3000 ☐Other _____ (use additional sheets if necessary)

Program Title

"MASTERMIND 8080" Run on SBC 80/10 Prototyping Board

Function

The computer selects a random code in the range 000000 to 999999. The player attempts to break this code by entering via the keyboard a series of guesses, based on "clues" provided by the computer after each guess.

Required Hardware

INTEL SBC 80/10 with TTY interface.
Alternatively INTELLEC MDS-800 with TTY or CRT interface.

Required Software

Console Input/Output routines resident in the System Monitors.

Input Parameters

To begin program execution, enter 'G3C3D'. To restart a game, enter any character via the keyboard. The computer then prompts the player to select: a) the length of code (i.e. number of columns) to be guessed.
-Enter one digit in the range 1 to 6.
b) the range of numbers to be used in the code (0-4 minimum, 0-9 maximum) The computer adjusts the original code appropriately if the full range 0-9 is not selected.
-Enter one digit in the range 4 to 9.
c) The maximum number of guesses allowed.
-Enter two digits in the range 01 to 99.
Program execution may be aborted at any stage by inputting the character 'A'

Output Results

After each attempt at guessing the code, the computer replies by outputting:
i) the number of digits correct in both value and position.
ii) the number of remaining digits having correct value but incorrect position.

Apropriate messages are outputted when the code is guessed correctly, or when the maximum number of attempts has been reached, as in (c) above, or when aborted.

Registers Modified: ALL	Programmer: C. J. Williams
RAM Required: 39F(HEX) = 927 bytes program 20(HEX) = 32 bytes workspace	Company: Cable & Wireless Ltd.
ROM Required: -	Address: 114 Great Suffolk St.
Maximum Subroutine Nesting Level:	City: London, SE1.
Assembler/Compiler Used: ISIS 8080 Macro Assembler V1.1	State: ENGLAND

Ref. #E25

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☐ 8080 ☒ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

LANDER

Function

A MOON LANDER GAME WHICH IS PLAYED ON THE INTEL PROMPT 48.

Required Hardware

PROMPT 48

Required Software

RESIDENT MONITOR SUBROUTINES

Input Parameters

KEYBOARD ENTRIES OF:
Ø - 9 FOR THRUST,
A FOR ALTITUDE DISPLAY,
F FOR REMAINING FUEL DISPLAY.

Output Results

DISPLAY SHOWS VELOCITY AND ALTITUDE OR REMAINING FUEL.

VELOCITY IS "FROZEN" AT TOUCHDOWN.

Registers Modified: A,RØ,R1,R2, R6,R7,FØ, RØ',R1',R2',R3',R6',R7',P2	Programmer: FRANK FAFF
RAM Required: 18H	Company: ATLANTIC RESEARCH CORPORATION
ROM Required: 15DH	Address: 5390 CHEROKEE AVENUE
Maximum Subroutine Nesting Level: 3	City: ALEXANDRIA
Assembler/Compiler Used: ISIS-II 8048 ASSEMBLER, V1.2	State: VIRGINIA, 22314

intel® MICROCOMPUTER USER'S LIBRARY SUBMITTAL FORM

Ref.#E26

☐ 4004 ☐ 4040 ☐ 8008 XX ☐ 8080

(use additional sheets if necessary)

Program Title

TIC TAC TOE - 3 Dimensional

Function

A GAME PROGRAM
GIVEN A CUBE OF 4 TIMES 4 TIMES 4 POINTS. TRY TO SET 4 POINTS IN LINE. COMPUTER TRIES THE SAME. EACH POINT CAN BE OCCUPIED EITHER BY THE COMPUTER OR BY THE HUMAN BEING. FIRST TO FINISH WINS. COORDINATES ARE GIVEN BY A SINGLE LETTER FOLLOWED BY A NUMBER. MATRIX IS OUTPUT AFTER EVERY ACTION. TYPING 'X' AFTER STARTING THE PROGRAM ALLOWS HUMAN TO START, ANY OTHER CHARACTER MAKES MACHINE START .

Required Hardware

no special hardware required

Required Software

Monitor Routines: OUTSP prints space on TTY
TTYIN inputs one character to A Register
TTYOU outputs one character from A register
CRLF does carriage return/line feed on TTY

Input Parameters

PROGRAM COMMUNICATES WITH TTY ONLY

Output Results

GAME MATRIX AFTER EACH ACTION

X=HUMAN ACTION
O=COMPUTER ACTION

P4 J2

	ABCD	EFGH	IJKL	MNOP
1	----	X---	----	----
2	----	-OX-	-O--	----
3	----	--O-	--X-	----
4	---O	---O	----	-X-X

Registers Modified: all	**Assembler/Compiler Used:** INTELLEC 8/80 Macro Assembler
RAM Required: 1 k Bytes	**Programmer:** R.Vogel/W.Vollenweider
ROM Required: none except for monitor routines	**Company:** HAMDATA
Maximum Subroutine Nexting Level: 4	**Address:** Bahnhofstrasse 46c CH-8305 Dietlikon Switzerland

Ref. #E27

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

CRAP'S (Dice Game)

Function

Allows user to play game of chance.

Required Hardware

SDK-80 with terminal

Required Software

SDK-80 monitor
Subroutines used: (Crout) (Typeout) (Echo) (Break) (CO)

Input Parameters

To start the game, type G CØØ on the console device. Follow instructions listed on the sign on message.

Output Results

On the console device a game of crap's will be demonstrated.

Registers Modified: ALL	Programmer: VAN HERNDON AND DAVE YONICH
RAM Required: 20 Bytes	Company: J. M. PERRY INSTITUTE
ROM Required: 1K ROM #3	Address: 2011 West Washington Avenue
Maximum Subroutine Nesting Level: 8	City: Yakima
Assembler/Compiler Used: 8080 Assembler	State: Washington 98903

```
MCS-80 KIT
•G0C00
     THIS IS A DICE GAME.TO ROLL PRESS THE ESCAPE KEY.
FIRST ROLL:  7 YOU WIN
            11 YOU WIN
             2 YOU HOSE
             3 YOU LOSE
            12 YOU LOSE
     IF NONE OF THE ABOVE THEN THIS NUMBER IS YOUR POINT.
ROLL AGAIN: AFTER ESTABLISHINGYOUR POINT YOU MUST ROLL THIS NUMBER
 TO WIN! BUT,  ROLL A 2 OR 7 YOU LOSE.
GOOD LUCK !!

;
;***

1,4 5 ROLL AGAIN
5,3 8 ROLL AGAIN
2,6 8 ROLL AGAIN
6,5 11 ROLL AGAIN
2,6 8 ROLL AGAIN
3,6 9 ROLL AGAIN
4,2 6 ROLL AGAIN
6,2 8 ROLL AGAIN
3,4 7 YOU LOSE

5,1 6 ROLL AGAIN
2,1 3 ROLL AGAIN
5,4 9 ROLL AGAIN
3,1 4 ROLL AGAIN
4,6 10 ROLL AGAIN
4,1 5 ROLL AGAIN
2,6 8 ROLL AGAIN
3,6 9 ROLL AGAIN
3,4 7 YOU LOSE

2,6 8 ROLL AGAIN
5,4 9 ROLL AGAIN
1,5 6 ROLL AGAIN
4,1 5 ROLL AGAIN
4,3 7 YOU LOSE

5,6 11 YOU WIN

4,6 10 ROLL AGAIN
1,6 7 YOU LOSE

4,5 9 ROLL AGAIN
6,4 10 ROLL AGAIN
4,1 5 ROLL AGAIN
4,2 6 ROLL AGAIN
1,6 7 YOU LOSE

4,3 7 YOU WIN

5,5 10 ROLL AGAIN
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

VDU DARTS

Function

A game of Darts on the MDS for two players with the board displayed on the VDU. Throws made by depressing a char on the console, the score being deduced from a time-slot for singles, doubles, trebles, outers, bulls and misses. Scoring starts at 2Ø1. (This game was originally written for a freestanding 8080 System and VDU).

Required Hardware

MDS and VDU

Required Software

MDS Monitor I/O Routines

Input Parameters

AL = C
(AC = C)
Read Ø
G50
Any console char to start game or to score.

Output Results

Dartboard Display
Target Indicator
Score per throw
Players totals per turn
"Well Done" or "Bust" messages.

Registers Modified: ALL	Programmer: Gerard L. Dooley
RAM Required: 32 Bytes	Company: Plessey Radar Limited
ROM Required: 1K	Address: Cheapside
Maximum Subroutine Nesting Level: 4	City: Liverpool
Assembler/Compiler Used: MDS MACRO ASSEMBLER V.1	State: Lancaster, ENGLAND

Ref. #E29

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

HANG

Function

Hangman Word Guess Game

Required Hardware

MDS-800
MDS-2DS
Beehive Mini B-2 (CRT)

Required Software

Intellec/MDS Monitor V2.0
MDS-DOS V1.2

Input Parameters

Comes with a list of 20 secret words which may be easily expanded or modified.

Output Results

The image of a gallows is constructed on the CRT, and the secret word appears as underlined blanks underneath.
The player enters his guess on the keyboard. A correct guess causes one or more blanks to be filled in the word.
A wrong guess is displayed at the top of the screen, and causes a part to be added to the picture of the hanged man.
The object of the game is to guess the word before the picture is completed.

Registers Modified: ALL	Programmer: Bernard J. Verreau
RAM Required: 734 Bytes	Company: NCR Corporation
ROM Required:	Address: P.O. Box 607 Mitchell Rd.
Maximum Subroutine Nesting Level: 12 Bytes	City: Millsboro
Assembler/Compiler Used: ISIS 8080 Macro Assembler, V1.1	State: Delaware 19966

Ref.# E30

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

BIORIM

Function

To print a graphical calendar plotting the operator's 'BIORHYTHM'

Required Hardware

MDS 800, Console Input, Line Printer, Disk Drive.

Required Software

PL/M-80, ISIS II.

Input Parameters

Operator's name and date of birth.

On loading under ISIS II program supplies its own instructions.

Output Results

Graphic calendar.
Information about Biorhythms is now quite widely published.

Registers Modified:	Programmer: David N. Larkins
RAM Required: 64 K Bytes	Company: R.C.A. Jersey Ltd.
ROM Required: None	Address: Rue des Pres, Longueville,
Maximum Subroutine Nesting Level: 8	City: JERSEY
Assembler/Compiler Used: PL/M-80 V3.0	State: Channel Islands.

insite T.M. **INTEL® USER'S LIBRARY SUBMITTAL FORM**

Ref.#E31

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

Slalom Version 1.4

Function

Game

Swiss Ski Champions Worldcup

Required Hardware

MDS with Disk and CRT (9600 recommended)
Line Printer optional

Required Software

ISIS-II

Input Parameters

Numeric Keys 1 to 9 on CRT for Tabulation
Text in Dialog

Output Results

on CRT
optional on Line Printer

Registers Modified:	**Programmer:** Ulrich E. Spörri
RAM Required: ISIS + 10K	**Company:** UES electronics & software
ROM Required:	**Address:**
Maximum Subroutine Nesting Level:	**City:** CH 8143 Stallikon
Assembler/Compiler Used: PL/M-80	**State:** Switzerland

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

HORSERACE (game of,)

Function

Will play five games of horse race on the MDS-CRT
or the built in CRT in Series II.

Required Hardware

MDS-800/220/230
MDS-CRT (or any CRT that has compatable escape functions
for cursor movement)

Required Software

ISIS-II disc operating system version 2.2 or 3.4

Input Parameters

All inputs are user prompted at the system console as
what to enter.

Output Results

All outputs are displayed on the system console

Registers Modified: All	Programmer: Kerry Howell
RAM Required: 48K	Company: Almac/Stroum
ROM Required: 2K MDS System Monitor	Address: 18760 NW Rock Creek Circle
Maximum Subroutine Nesting Level: N/A	City: Portland #134
Assembler/Compiler Used: Fortran-80 V1.0	State: Oregon 97229

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☐ 8080/8085 ☒ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

MASTERMIND FOR SDK-86

Function

A game of logic

RULES

The computer sets up a 4-digit code. Each digit is in the range 9 to 5. The game is to break the code by typing in a series of guesses. For each guess, the program returns the following information: an "*" for each correct digit in correct position. and a "." for each correct digit in a wrong position. For example. if the computer sets up 5025 and the player enters 3215, the program will type *

Required Hardware

SDK-86

Required Software

PLM86 or ASM86, LINK-86, LOQ86, OH86, SDK86, SDKIOS. LIB

Input Parameters

(This program is available in PL/M86 or ASM86. Please specify version when ordering. Each version is considered one program.)

Output Results

Registers Modified: All	Programmer:
RAM Required: 5K or SDK-86 Board	Company:
ROM Required: None	Address:
Maximum Subroutine Nesting Level:	City:
Assembler/Compiler Used: PLM86 or ASM86	State:

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

TECH-NEL FRUIT MACHINE GAME, V1.0

Function

Game program - performing functions of a fruit machine (one-armed bandit)
- including: Random Hold, Full reel display and Plays/wins count

Required Hardware

MDS - Series II (210 or greater, with CRT display)

Required Software

MDS Monitor - Console out
Console status
Console in

Input Parameters

Key functions

Output Results

Display of reels/winning line/holds, etc

Registers Modified:	Programmer: Andy Belton
RAM Required: 1.1K (object)	Company: Tech-nel Data Products Ltd
ROM Required:	Address: PO Box 1, 95 High St
Maximum Subroutine Nesting Level:	City: Brackley
Assembler/Compiler Used:	State: Northants,England NN13 5NN

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

MOUSE

Function

A mouse can be controlled on its way through the maze. Mouse can map the maze when bumps against the wall, but the less bumps the better.

Required Hardware

MDS 800, CRT /MiniBee/ cursor commands: ESC,H - home pos., ESC,J-screen erazing, ESC,A-up, ESC,B-down, ESC,C-right, ESC,D-left.

Required Software

CI and CO from MONITOR and subroutines from SYSTEM.LIB.

Input Parameters

Movement on the screen is controlled by typing "F", "B", "L" and "R" standing for ↑ , ↓ , ← , → . "H" displays the whole maze. One out of four gates must be chosen at the start.

Output Results

On the screen is displayed the outline of the maze with an exit and chosen gate. Free movement is displayed as "+" and wall as "@" with a bell. When the exit is successfully reached the score /steps and bumps/ is displayed.

Registers Modified: ALL	Programmer: Dalibor NEMEC
RAM Required: cca 3.2K	Company:
ROM Required: None	Address: Pelhrimovska 339/9
Maximum Subroutine Nesting Level: 8	City: 145 00 PRAHA-4, Michle
Assembler/Compiler Used: PL/M-80 V3.0	State: Czechoslovakia

SECTION 9

RESIDENT LANGUAGE TRANSLATORS

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☒8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

SMAL: A Symbolic Microcontroller Assembly Language

Function

This program is an editor/assembler combination for a simple microcontroller which can be constructed from only a few TTL packages. The microcontroller is intended for use in peripheral ROM driven circuitry in the microcomputer environment. The assembler produces machine code in the Intel "hex" format, suitable for ROM programming.

Required Hardware

Intellec 8 or Intellec /Mod 80 (or equivalent) with at least 8K

Required Software

No additional software is required for run-time support, although the 8008 or 8080 resident monitor is useful for loading the program.

Input Parameters

Assembler language source, along with editing commands

Output Results

Annotated assembler language listing, along with two "hex" tapes for high and low order bytes for each location.

NOTE:

This program is not supported by Intel. For more information or assistance in operation contact the author.

PAPER TAPE AVAILABLE IN OBJECT CODE WITH A SOURCE LISTING. DISKETTE IS OFFERED IN SOURCE CODE.

Registers Modified: All	Programmer: Gary A. Kildall
RAM Required: Variable - 8K nominal	Company: Naval Postgraduate School
ROM Required: None	Address: Code 72Kd
Maximum Subroutine Nesting Level: unknown - Stack size 16 bytes	City: Monterey, California
Assembler/Compiler Used: PL/M (8080, Vers 1.0)	State:

Ref. No. F2

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

ML80 Structured Assembler for the 8080

Function

Macroprocessor, assembler, and loader which translate 8080 assembly language with control structure to relocatable object code.

Required Hardware

8080 developmental system with disk storage

Required Software

Disk operating system which provides a file structure allowing sequential access to at least four files

Input Parameters

ML80 assembly language source

Output Results

Macroprocessor produces an ASCII file which is read and translated to relocatable code.

NOTE:

This program is not supported by Intel. For more information or assistance in operation contact the author.

Program offered on diskette only.
DISKETTE IS OFFERED IN SOURCE CODE.

Registers Modified: All	Programmer: Luiz Pedroso (contact Gary Kildall)
RAM Required: 12K (excluding DOS)	Company: Naval Postgraduate School
ROM Required: 0	Address:
Maximum Subroutine Nesting Level:	City: Monterey
Assembler/Compiler Used: PL/M version 2.0	State: California 93940

Ref. No. F3

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

BASIC/M Translator and Interpreter

Function

Translator compiles BASIC/M programs to pseudo machine code; interpreter loads and executes the pseduo code. BASIC/M programs are stored in a diskette file.

Required Hardware

MDS system with 32K RAM, CRT, and floppy disk drive(s).

Required Software

ISIS, version 1.0 or above.

Input Parameters

BASIC/M program (see BASIC/M user's manual)

Output Results

Program execution.

NOTE:

This program is not supported by Intel. For more information or assistance in operation contact the author.

PAPER TAPE AVAILABLE IN OBJECT CODE WITH A SOURCE LISTING. DISKETTE IS OFFERED IN SOURCE CODE.

Registers Modified: All	Programmer: T.J. Logan and K.A. Kildall
RAM Required:	Company: Digital Research
ROM Required: 0	Address: Box 579
Maximum Subroutine Nesting Level: All	City: Pacific Grove
Assembler/Compiler Used: PL/M and 8080 Macro Assembler	State: Ca. 93950

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

LLL BASIC-II INTERPRETER

Function

This BASIC interpreter was designed to operate with the MCS-8080 microprocessor. It consists of an 8K-byte-ROM resident interpreter used for program generation and debug.

The goal in developing the 8080 BASIC was to provide a high-level, easy-to-use language for performing both control and computation functions in the MCS-8080 microprocessor. It was necessary, therefore, to limit the commands to those considered the most useful in microprocessor applications. With not too many exceptions, it conforms to the standards expressed by the ANSI committee, X3J2, on Minimal Basic.

Required Hardware

Required Software

8080 Microcomputer System Terminal Input
Device independent I/O

Input Parameters

Manual supplied with program

Revised 12/78

Output Results

Manual supplied with program

NOTE:

This program is not supported by Intel. For more information or assistance in operation contact the author.

Available in source code on diskette for $35.00

Registers Modified:	ALL	Programmer:	Eugene Fisher
RAM Required:	As required for user space	Company:	Lawrence Livermore Lab.
ROM Required:	8K byte	Address:	Box 808 L-403
Maximum Subroutine Nesting Level:		City:	Livermore
Assembler/Compiler Used:	ISIS-II ASM-80	State:	CA 94550

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

OCTAL DEBUGGING PROGRAM (ODT-80) FOR THE MCS-80 COMPUTER

Function

ODT-80 is an octal debugging routine for use on the Intel 8080 microprocessor. This routine provides the capability to examine and modify all of the memory that is available to the microcomputer and transfer program control to the created program. ODT-80 makes use of simple keyboard commands from any terminal---such as a teletypewriter---that is attached to the system.

Required Hardware

The minimum system requirements for using ODT are as follows:

- MCS-80 computer set
- ODT programmable read only memory (PROM) at memory page 000_8
- 256 word (RAM) at page 010_8
- Teletype interface

Required Software

None

Input Parameters

See attached description.

Output Results

See attached description.

NOTE:

This program is not supported by Intel. For more information or assistance in operation contact the author.

PAPER TAPE AVAILABLE IN OBJECT CODE WITH A SOURCE LISTING. DISKETTE IS OFFERED IN SOURCE CODE.

Registers Modified: All	Programmer: Eugene Fisher
RAM Required: Stack in ADR 010_8000_8	Company: Lawrence Livermore Laboratory
ROM Required: 256 Bytes	Address: Box 808 L-403
Maximum Subroutine Nesting Level: ---	City: Livermore,
Assembler/Compiler Used: 8080 Assembler 2.2	State: Ca. 94550

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/40 ☐ 8008 ☐ 8080 ☐ 8048 ☐ 8085 ☐ 3000 ☐ Other _____ (use additional sheets if necessary)

Program Title

4040 Cross Assembler for Intellec 8/Mod 80 and MDS-800

Function

Required Hardware

Required Software

REVISED 8/8/77

Input Parameters

CAUTION:

1. This program is not supported by Intel Corporation.
2. This program works only with the 8080 MDS Macro Assembler Version 1.0 and the Intellec 8/Mod 80 Macro Assembler Version 3.0. Modifications for other versions of the assemblers may be very difficult and requires knowledge of PL/M.

Output Results

NOTE:

This program is not supported by Intel. For more information or assistance in operation contact the author. Diskette available in object code only.

Registers Modified:	Programmer:
RAM Required:	Company:
ROM Required:	Address:
Maximum Subroutine Nesting Level:	City:
Assembler/Compiler Used: 8080 MDS Macro Assembler Version 1.0	State:

Ref. # F10

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☒ 8008 ☐ 8048 ☐ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

ASM08 MACRO ASSEMBLER

Function

MACRO assembler for 8008 on Intellec® MDS

Required Hardware

Intellec® MDS

Required Software

Monitor, ISIS-I

Input Parameters

Operation is identical to operation of 8080 assembler on Intellec® MDS
User's standard Intel 8008 Assembly language

Output Results

No source code is available for this product. Diskette and Paper Tape available in object code only. Source listing can be ordered from Insite for a prepaid $15.00 handling fee.

Registers Modified:	Programmer:
RAM Required: 8K	Company:
ROM Required:	Address:
Maximum Subroutine Nesting Level:	City:
Assembler/Compiler Used:	State:

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title
8080 Macro Assembler Version 4.1

Function
Assembles 8080 Assembly Language Programs

REVISED 4/8/77

Required Hardware
Intellec MDS

Required Software
MDS Monitor

Input Parameters
An Assembly Language Program on Paper Tape.

Output Results
Assembled Text Listing (with error codes, if any)
Machine Code/Hex Object Code

Source listing available from Insite for a prepaid $15.00 handling fee. Paper tape is not available except when ordering system. Program Diskette not offered.

Registers Modified: All	Programmer:
RAM Required: 8K bytes	Company:
ROM Required:	Address:
Maximum Subroutine Nesting Level:	City:
Assembler/Compiler Used: PL/M	State:

Ref. # F12

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☒8008 ☐8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title
8008 Macro Assembler Version 2.0

Function
Assembles 8008 Assembly Language Programs

Required Hardware
Intellec 8/Mod 80

Required Software
8008 Monitor I/O

Input Parameters
Assembly Language Program File

Output Results
Listing of Assembled Text (with error codes, if any)
Machine Code
Hex Object Tape

Source listing available from Insite for a prepaid $15.00 handling fee. Paper tape is not available except when ordering system. Program Diskette not offered.

Registers Modified: All	Programmer:
RAM Required: 8K bytes	Company:
ROM Required:	Address:
Maximum Subroutine Nesting Level:	City:
Assembler/Compiler Used: PL/M 8ØØ8	State:

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004 ☐4040 ☐8008 ☒8080 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

PAS80 Ver 1.0, Sequential PASCAL Compiler

Function

To provide a sequential PASCAL compiler and virtual machine implementation for the Intel 8080A microcomputer.

Required Hardware

Intel Intellec Microcomputer Development System (MDS) with:
64K RAM
Dual Floppy Disks

Required Software

All the required software is provided to compile and execute sequential PASCAL programs within the MDS ISIS-II environment. However, in order to modify the PAS80 program the user will need the PL/M-80 compiler, the ISIS-II Link and Locate programs.

Input Parameters

The input to the PAS80 program is a virtual machine code file which is to be executed. When the virtual machine code file for the sequential PASCAL compiler is executed by PAS80, a source file written in the sequential PASCAL language is then expected as input.

Output Results

Compiled listings and virtual machine code files are produced by the sequential PASCAL compiler.

Paper tape not available. Program available on Diskette only. User's Manual and Language Definition Manual included.

Registers Modified:	Programmer: Thomas A. Rolander
RAM Required:	Company:
ROM Required:	Address: 1012 Smith Ave.
Maximum Subroutine Nesting Level:	City: Campbell
Assembler/Compiler Used:	State: California 95008

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

PAS80: (Continued)

Description:

PASCAL is a general purpose language for structured programming developed by Niklaus Wirth. The sequential PASCAL compiler, written by Per Brinch Hansen and Alfred C. Hartmann of Caltech, generates code for a virtual machine. The author has simulated the virtual machine with a real machine, the Intel Intellec Microcomputer Development System (MDS). The PAS80 program, which is the implementation of the virtual machine, is written in PL/M-80.

The software is distributed on two non-system disks containing: the PAS80 program, the sequential PASCAL compiler in virtual machine code form, the PL/M-80 source code for PAS80, and the source code for the entire 7 pass sequential PASCAL compiler written in sequential PASCAL. Thus, PAS80 will enable the user to self-compile the sequential PASCAL compiler.

Implementation Considerations:

Emulating a 16 bit virtual machine using PL/M-80 on the Intel 8080A certainly does not produce a high-speed real machine. However, compilation and execution speeds are tolerable, particularily for environments in which the language PASCAL is used for educational purposes.

Ver 1.0 of PAS80 does not support floating point operations. However, all of the required hooks have been incorporated, facilitating future implementation.

Compiler Support:

No warranty is expressed or implied by either the author or the writers of the sequential PASCAL compiler.

Compiler documentation is supplied in the form of syntax graphs, a programming example and the source code for the compiler.

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

RIA80

Function

Interpretive Assembler that runs on an Intellec 8/MOD80 or on any SBC system with an SBC-910 system monitor. (See additional sheets.)

Required Hardware

Intellec 8/MOD80 or any SBC system with an SBC-910 system monitor, e.g. system 80/10 or SBC-80P.

Teletype.

Required Software

Intellec system monitor vers. 3.0 or SBC-910 system monitor.

Input Parameters

Starting address of the program to be assembled.
Assembly language program.

Output Results

Assembled program in RAM at address specified.
Complete listing of address, machine code, and assembly language nmemonic for each instruction.

REVISED 4/78

Registers Modified: ALL	Programmer: M. A. PORDES
RAM Required: 10 Bytes + Stack	Company: GEC Hirst Research Centre
ROM Required: 1740 Bytes (Intellec) 1751 Bytes (SBC)	Address: East Lane, Wembley, Middx.
Maximum Subroutine Nesting Level: 8	City: London
Assembler/Compiler Used: MDS 8080/8085 Assembler V1.0	~~State~~: ENGLAND

Ref. #F14

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

RIA80

Function

Interpretive Assembler that runs on an Intellec 8/MOD80 or on any SBC system with an SBC-910 system monitor. (See additional sheets).

Required Hardware

Intellec 8/MOD80 or any SBC system with an SBC-910 system monitor, e.g. system 80/10 or SBC-80P.
Teletype.

Required Software

Intellec system monitor vers. 3.0 or SBC-910 system monitor.

Input Parameters

Starting address of the program to be assembled.
Assembly language program.

Output Results

Assembled program in RAM at address specified.
Complete listing of address, machine code, and assembly language mnemonic for each instruction.

Registers Modified:	ALL	Programmer:	M.A. Pordes
RAM Required:	10 Bytes + Stack	Company:	GEC Hirst Research Centre
ROM Required:	1740 Bytes (Intellec) 1751 Bytes (SBC)	Address:	East Lane, Wembley, Middx.
Maximum Subroutine Nesting Level:	8	City:	London
Assembler/Compiler Used:	MDS 8080/8085 Assembler V1.0	[redacted]:	ENGLAND

RIA80

RIA80 is an Interpretive assembler designed to run on the Intellec 8/MOD80 or, with a few modifications on any SBC 80/10 system that utilises the SBC - 910 system monitor. A major advantage of RIA80 is that it occupies less than 2K bytes of memory, thus allowing the assembler to be run on a single SBC 80/10 board without the need for additional memory boards.

RIA80 will assemble all of the standard 8080 instructions as defined in the 8080 Assembly Language Programming Manual, with the following exceptions:-

1) No labels are allowed

2) The only pseudo-instructions allowed are ORG, END, DS, DB, and DW

3) No operators are allowed

4) The assembler will not accept decimal, binary or octal numbers.

All numbers input to the assembler are assumed to be in hexadecimal, typing an H after any hexadecimal number is optional. Also it is not necessary to type a Ø before a number that starts with A - F.

Operating Instructions

On starting the program the assembler will print the pseudo-instruction ORG, at which point the user must type in the address of the area in RAM into which the assembled program will be placed. The user can then proceed to enter his program.

When entering an instruction the code part of the instruction must be separated from the operand (if any), by a single space.

If an illegal instruction is entered it is either thrown out immediately or when the appropriate delimiter is typed in, allowing the user to re-enter his instruction.

The data part of the instructions MVI and CPI, can, as well as being a hex.number be any Ascii character providing the character is enclosed within single quotes (').

DB instructions can consist of a sequence of hex. numbers, a string of Ascii characters (providing the string is enclosed within single quotes), or a combination of the two types. The maximum length of "list" is ten hex. numbers or Ascii characters.

cont. ... / 2

Comments can be added to any instruction by typing a semicolon (;) immediately after entering the instruction. The comment can be of any length and is terminated by typing carriage return (CR).

The pseudo-instruction ORG, followed by an address, can be entered any time during the assembly allowing the user to assemble different parts of his program at different addresses.

When all the instructions in the program have been entered, the pseudo-instruction END is typed to terminate the assembly.

Examples of all the above points are given in the test program.

Modification To Run On An SBC-80/10

To run RIA80 on an SBC-80/10 system with an SBC-910 system monitor the following changes should be made to the assembler:-

1) The statements following the label EX11

```
        MOV   D,A
        CALL  NIBBLE
        JC    EX21
```

should be changed to

```
        MOV   D,A
        CALL  VALDG
        JNC   EX21
        CALL  CNVBN
```

2) An additional subroutine must be added to read characters from the teletype. The subroutine has been labelled TI to allow for already existing references to this label to remain unchanged in the program.

```
TI:     CALL  GETCH
        CALL  CO
        MOV   A,C
        RET
```

3) All references to Intellec 8/MOD80 system monitor routines (apart from (1) and (2) above) should be substituted by the corresponding SBC-910 system monitor routines as follows:-

LBYTE should now be INUST at address 2B2H

MNTR " " " NMOUT " " 202H

cont. ... / 3

CI	should	now	be	CI	at	address	1D5H
CRLF	"	"	"	CROUT	"	"	1F3H
NIBBLE	"	"	"	CNVBN	"	"	1DFH
LADR	"	"	"	ADRD	"	"	1A8H
CO	"	"	"	CO	"	"	1E8H

In addition TI should be removed from the table at the beginning of the program, and GETCH (at address 220H) should be added to the table.

Ref.# F15

insite™ INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/40 ☐8008 ■8080 ☐8048 ☐8085 ☐3000 ☐Other_____ (use additional sheets if necessary)

Program Title

LLL/CHERNACK BASIC INTERPRETER

Function

Modified LLL BASIC to run under ISIS-II, LOAD and LIST programs to ISIS-II filenames.

Required Hardware

ISIS-II hardware configuration.

Required Software

ISIS-II software configuration.

Input Parameters

See attached documentation. Attached DISKETTE contains all sources, object file, and appropriate .CSD files to re-create BASIC or to Run BASIC under ICE-80.

Output Results

Note: Source available on Diskette only for $35.00.

Registers Modified:	Programmer: Charles Chernack
RAM Required:	Company: Consultant
ROM Required:	Address: 1034 East Rose Circle
Maximum Subroutine Nesting Level:	City: Los Altos,
Assembler/Compiler Used:	State: California 94022

2/6/78

NOTES: Chernack/LLL MDS/DOS BASIC Version "C"

This version is a modification of LLL BASIC from the INTEL User's Library. LLL BASIC has been made relocatable, and broken down into modules. Only very minor changes have been made to the Interpreter.

A new command, HELP, has been added, which simply lists all the commands.

A new command, EXIT, has been added, which returns to ISIS-II.

A new command, LOAD, has been added, which must be followed by an ISIS-II filename. This command will load a BASIC program which has been created by BASIC or by the TEXT EDITOR. It effectively 'appends' the program to the BASIC Program Area. The LOAD is terminated by end-of-file or by the first line which starts with a non-numeric character. However, if termination is via a line which starts with a non-numeric character, that line is interpreted as a command. Thus, if the last line of your program is LIST or RUN, the program will be listed or RUN after it is loaded. You should be able to do chain-loads this way as well.

The LIST command has been modified so that you can not specify the range in line numbers for the list, but you can specify an ISIS-II filename. Thus, you can use the LIST command to save programs on the disc, or list them to the line printer. If LIST is not followed by a filename, it will list on the :CO: file.

You can get out of a running program by pressing INTERRUPT 7, which will return with READY. You can also stop a running program which is doing output by hitting the space bar on the CRT.

You can link to absolute or relocatable subroutines. The SUBS table, described in the LLL BASIC write-up, has been moved into a free-standing relocatable module called BASIC7.OBJ. There are 20 blank locations (DS 20) which you can use with an absolute overlay, or you can replace BASIC7.OBJ with a module of your own choosing.

BASIC must be loaded such that the code segment starts at around 3400H. This is because there is still an absolute block between 3200H and 32AAH which is used as "common." On the source disc, there is a file called MAKIT.CSD which will make an absolute version of BASIC from its relocatable modules.

BASIC uses MDS MONITOR subroutine MEMCHK to determine where the end of memory is, so that you can run it on 32K, 48K, and 64K MDS systems. Normally, BASIC will do much of its I/O using ISIS-II system calls; however if location 40H is set to 76H, then BASIC knows that ISIS-II system calls may not be used, and will automatically switch directly to the CO and CI subroutines in the MONITOR. The automatic switch-over allows BASIC to be run under ICE-80. Note that file BASICE.CSD sets up for a 64K ICE-80 session. Value TP1 will have to be changed (See BASIC1.COO) if you use less memory, because of the UPPERLIMIT problem with ICE-80. Remember that ICE-80 must store the BASIC program symbols someplace, and there are quite a few of them.

If you do an absolute overlay for your assembly callable subroutines, you will have to find symbol SUBS and also overlay LXI H,MEMORY to allow yourself some more room. This is near TPØ is BASIC1.COO.

You will note that this is a rather patch-up job, but it may be some time before I get an update out and perhaps some may find this useful.

It is fairly easy to add commands such as DELETE filename, WRITEPROTECT filename, etc. to BASIC the way it is now written. Also ISIS errors are not reported very well, and the ETEST routine is BASIC1 can be enhanced quite a bit.

The next major change I contemplate to BASIC will be an investigation of the substitution of the INTEL Floating Point Package for the rather long floating point package now in BASIC.

BUGS: (1) Sometimes the first LOAD :FN:XXXXXX.XXX will hang up. Re-boot and all will be well.

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☐8080 ☐8048 ☒8085 ☐Other_____ (use additional sheets if necessary)

Program Title

PILOT-80 Ver.2.0 - INTERPRETER

Function

PILOT is a programming system for controlling interactive conversations. It can be used as an author language for computer-assisted instruction. Designed to be simple in its syntax, PILOT allows those without prior computer experience to easily learn to control its features. Dialogue programs can be rapidly constructed and tested.

Required Hardware

ISIS-II Hardware Configuration

Required Software

ISIS-II Software Configuration

Input Parameters

PILOT-80 program (see PILOT-80 User's Guide)

Output Results

Program execution

Source available on diskette only for $35.00. Source listing available for $15.00.

Registers Modified: ALL	Programmer: JOHN STARKWEATHER & RON WILLIAMS
RAM Required: 4K & 2K Editor & Program Requirements	Company:
ROM Required:	Address:
Maximum Subroutine Nesting Level:	City:
Assembler/Compiler Used: 8080/8085 MACRO Assembler V2.0	State:

Ref.# F17

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

LISP INTERPRETER (8080 resident)

Function

Provide input and output of LISP data structures, and interpretation of LISP expressions. Functions supported include DEFINE, CAR, CDR, CONS, ATOM, EQ, LAMBDA, LABEL, QUOTE and COND, CADR, CADDR, NULL, EQUAL, PAIRLIS, ASSOC, EVAL, APPLY.

Required Hardware

Terminal, up to 32K of RAM addressed below 8000H.

Required Software

Terminal driver with read and write character entries; monitor program which CALLs interpreter with stack setup, with entry point for interpreter abort.

Input Parameters

Functions and arguments in LISP to be interpreted: via terminal input. Except for DEFINE, all functions are required to be pure LISP functions.

Output Results

Results of applying functions to arguments are listed on terminal.

Registers Modified: ALL	Programmer: Darrel J. Van Buer
RAM Required: Up to 32K (below address 8000H)	Company: SELF
ROM Required: 1731 bytes for program	Address: 1522 Brockton Ave., Apt 5
Maximum Subroutine Nesting Level: Limited only by expression complexity	City: Los Angeles
Assembler/Compiler Used: Intel compatible assembler	State: California 90025

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

ASM - ON-LINE ASSEMBLER

Function

Allows instructions to be entered by mnemonics rather than absolute binary for experimental or debugging purposes. Especially useful on small machines without much I/O capability. Not useful for production programming.

Required Hardware

1 k bytes of ROM if an INTEL debugging monitor or similar piece of software is avialable, and slightly more if not; enough RAM for a minimal stack and to hold assembled code; terminal interface.

Required Software

Intel monitor would reduce memory requirement but is not necessary.

Input Parameters

8080 mnemonics are typed in from the ternimal. Operands are also accepted (all numbers are in hex, and characters can be assembled by enclosing them in quotes) although labels and expressions are not allowed. The following pseudo-operations are supported; ORG (addr), DB (byte-list), END (addr)

Output Results

Code is assembled directly into memory. No object code is produced.

Available on non-system diskette for $25.00; paper tape for $15.00.

Registers Modified: All	Programmer: Bruce C. Wright
RAM Required: (as desired by user)	Company: Duke Medical Center
ROM Required: 1 K byte	Address: Box 3181
Maximum Subroutine Nesting Level: 4 (8 bytes)	City: Durham
Assembler/Compiler Used: MDS MACRO Assembler	State: North Carolina

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☐ 8080/8085 ☒ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

8086 TINY BASIC

Function

A very small (<1K of Code) Basic interpreter. 26 variables (A-2) and 1 array (@). Statements recognized include GO70, RUN, NEW, FOR, NEXT, LET, IF, STOP

Required Hardware

MDS or EMDS with at least 2 empty slots
SDK-86 or SBC-86 board

Required Software

SDK-86 or SBC-86 Program

Input Parameters

None, Tiny Basic has no input parameters. Once it is running it accepts console input in the form of Basic Commands.

Output Results

Program offered on diskette only for $35.00

Registers Modified:	Programmer: Bob Glossman
RAM Required:	Company: Intel
ROM Required:	Address: SCVI MS6-226 X5169
Maximum Subroutine Nesting Level:	City: Santa Clara
Assembler/Compiler Used: ASM86	State: CA 95051

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 8008 ☐ 8048 ☒ 8080/8085 ☐ 8086 ☐ Other _____ (use additional sheets if necessary)

Program Title

STAGE 2

Function

STAGE 2 is a language independent macro processor developed by prof. William M. Waite of the University of Colorado. It is primarily intended to be used to omplement machine independent software by means of abstract machine modelling, but it is suitable for use as a prepass for any language translator for purposes of providing macro capability.

Required Hardware

MDS 800 Intellec Microcomputer Development System with console device diskette drive(s) and controller, at least 48K RAM memory.

Required Software

ISIS-II Operating System

Input Parameters

Channel assignments equivalencing STAGE 2 channels with ISIS-II files and devices. This implementation of STAGE 2 supports inie non-dummy I/O channels.

Output Results

User defined.

Program offered on diskette only for $35

Registers Modified:	Programmer: Bruce W. Ravenel
RAM Required: 48K minimum	Company: Intel - MCD Software Devl.
ROM Required:	Address: 3065 Bowers Ave.
Maximum Subroutine Nesting Level:	City: Santa Clara
Assembler/Compiler Used:	State: Ca. 95051

SECTION 10

CROSS REFERENCE TO PL/M PROGRAMS

SECTION 11

RMX-80

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐4004/4040 ☐8008 ☒8080 ☐8048 ☐8085 ☐Other_____ (use additional sheets if necessary)

Program Title

CLI - RMX-80 Command Line Interpreter

Function

Provides operator control of RMX tasks.

Required Hardware

SBC 80/20

Required Software

RMX/80 Nucleus, Free Space Manager Terminal Handler

Input Parameters

A command line read from the terminal handler. The first word is tested against a list of keywords CLI received from other tasks. If a match is found, CLI sends the command line to the exchange associated with the keyword.

Output Results

RMX 80 Users Guide $5.00.
Available from Literature.

Note: Source available on diskette only.

Registers Modified: ALL	Programmer: KEN BURGETT
RAM Required: 46	Company: DHARMA SYSTEMS
ROM Required: 988	Address: 21950 McKEAN RD
Maximum Subroutine Nesting Level: 2	City: SAN JOSE
Assembler/Compiler Used: PL/M, V3.0	State: CALIFORNIA 95120

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

WRMIN - RMX Minimal Terminal Output

Function

Provides a terminal handler output task for software operating under RMX/80.

Required Hardware

SBC 80/20

Required Software

RMX/80 Nucleus

Input Parameters

The WRMIN task performs similar functions to the THDINI task of the RMX/80 Terminal Handler. It uses the same output request exchange (RQOUTX) and message format as described in the RMX/80 User's Guide.

Output Results

See application Note AP33

Note: Source available on diskette only.

RMX 80 Users Guide $5.00.
Available from Literature.

Registers Modified: ALL	Programmer: THOMAS ROLANDER
RAM Required: 15	Company: DHARMA SYSTEMS
ROM Required: 74	Address: 21950 McKEAN RD.
Maximum Subroutine Nesting Level:	City: SAN JOSE
Assembler/Compiler Used: 8080/8085 MACRO ASSEMBLER	State: CALIFORNIA 95120

```
ASM80 :F1:WRMIN.M80

ISIS-II 8080/8085 ASSEMBLER, V1.0        WRMIN                                  PAGE

 'OC  OBJ          SEQ          SOURCE STATEMENT

                     0           NAME    WRMIN
                     1           EXTRN   RQELVL,RQOUTX,RQWAIT,RQSEND
                     2           PUBLIC  WRMIN,RQL7EX
00EC                 3 DATOUT    EQU     0ECH    ; USART OUTPUT PORT ADDRESS
                     4           CSEG
                     5 WRMIN:
0000 0E07            6           MVI     C,7
0002 CD0000    E     7           CALL    RQELVL  ; ENABLE INTERRUPT LEVEL 7
                     8 WR0:
0005 110000          9           LXI     D,0
0008 010000    E    10           LXI     B,RQOUTX
000B CD0000    E    11           CALL    RQWAIT  ; WAIT FOR OUTPUT REQUEST
000E E5              12           PUSH    H       ; PUSH MESSAGE ADDRESS
000F 110700          13           LXI     D,7
0012 19              14           DAD     D
0013 4E              15           MOV     C,M     ; GET RESPONSE EXCHANGE
0014 23              16           INX     H
0015 46              17           MOV     B,M
0016 23              18           INX     H
0017 C5              19           PUSH    B       ; PUSH RESPONSE EXCHANGE
0018 3600            20           MVI     M,0     ; STATUS = 0
001A 23              21           INX     H
001B 3600            22           MVI     M,0
001D 23              23           INX     H
 01E 5E              24           MOV     E,M     ; GET BUFFER ADDRESS IN DE
001F 23              25           INX     H
0020 56              26           MOV     D,M
0021 23              27           INX     H
0022 4E              28           MOV     C,M     ; GET COUNT IN BC
0023 23              29           INX     H
0024 46              30           MOV     B,M
0025 23              31           INX     H
0026 71              32           MOV     M,C     ; ACTUAL = COUNT
0027 23              33           INX     H
0028 70              34           MOV     M,B
                     35 WR1:
0029 78              36           MOV     A,B
002A B1              37           ORA     C
002B CA4300    C     38           JZ      WR2     ; EXIT LOOP IF COUNT = 0
002E C5              39           PUSH    B
002F D5              40           PUSH    D
0030 110000          41           LXI     D,0
0033 010000    D     42           LXI     B,RQL7EX
0036 CD0000    E     43           CALL    RQWAIT  ; WAIT FOR TXRDY INTERRUPT
0039 D1              44           POP     D
003A C1              45           POP     B
003B 1A              46           LDAX    D
003C 13              47           INX     D
003D D3EC            48           OUT     DATOUT  ; TRANSMIT NEXT CHARACTER FROM B
^03F 0B              49           DCX     B
 040 C32900    C     50           JMP     WR1
                     51 WR2:
0043 C1              52           POP     B       ; BC = RESPONSE EXCHANGE
0044 D1              53           POP     D       ; DE = MESSAGE ADDRESS
0045 CD0000    E     54           CALL    RQSEND  ; SEND MESSAGE TO RESPONSE EXCHA
0048 C30500    C     55           JMP     WR0
                     56 ;
                     57           DSEG
```

```
                        58 R0L7EX:
ISIS-II 8080/8085 ASSEMBLER, V1.0        WRMIN                                  PAGE


  LOC  OBJ          SEQ          SOURCE STATEMENT

  ^900           C     59          DS      15
                       60 ;
                       61          END


PUBLIC SYMBOLS

R0L7EX D 0000    WRMIN  C 0000

EXTERNAL SYMBOLS
R0ELVL E 0000    R0OUTX E 0000    R0SEND E 0000    R0WAIT E 0000

USER SYMBOLS
DATOUT A 00EC    R0ELVL E 0000    R0L7EX D 0000    R0OUTX E 0000    R0SEND E 000
WR1    C 0029    WR2    C 0043    WRMIN  C 0000

ASSEMBLY COMPLETE, NO ERROR(S)
```

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other_____ (use additional sheets if necessary)

Program Title

MINITH - RMX Minimal Terminal Handler

Function

The function of MINITH is to provide all the basic requirements for a terminal handler. MINITH is described in detail in the application note on RMX/80, AP-33.

Required Hardware

SBC 80/20

Required Software

RMX/80 Nucleus

Input Parameters

The input parameters and output results for MINITH are a subset of those for the THDINI and THDINO tasks of the RMX/80 Terminal Handler described in the RMX/80 User's Guide.

Output Results

See Applications Note AP33.

RMX 80 Users Guide $5.00.
Available from Literature.

Note: Source Available on diskette only.

Registers Modified: ALL	Programmer: THOMAS ROLANDER
RAM Required: 67	Company:
ROM Required: 570	Address: 21950 McKEAN RD
Maximum Subroutine Nesting Level:	City: SAN JOSE
Assembler/Compiler Used: PLM/80	State: CALIFORNIA 95120

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004 ☐ 4040 ☐ 8008 ☒ 8080 ☐ 3000 ☐ Other_____ (use additional sheets if necessary)

Program Title

RMX/80-Based Keyboard Input Handler Subroutine

Function

Accepts ASCII input characters and analyzes for (Carriage Return) (Line Feed) or (Delete) control characters. Dispatches character to a display exchange and/or input exchange using RMX/80 calling procedures.

Required Hardware

80/20 SBC with standard ASCII I/O terminal (with keyboard input port designated as 32H).

Required Software

RMX/80 Operating System Vers. 1.2, using standard configuration procedure to create user object program from RMX/80 library and user-written modules.

Input Parameters

Standard ASCII keyboard entry, using RMX/80 System standard routines/ calling prodedures.

Output Results

ASCII characters to teleprinter or display.

RMX 80 Users Guide $5.00.
Available from Literature.

Note: Source Available on diskette only.

Registers Modified: All	Programmer: M. H. Gansler
RAM Required: 37 Bytes + Stack (this module only)	Company: NTI
ROM Required: 85 Bytes (this module only)	Address: 4041 Home Rd
Maximum Subroutine Nesting Level: 2 (minimum)	City: Bellingham
Assembler/Compiler Used: ISIS-II 8080/8085 Macro Assem. V2.0	State: Washington 98225

THIS IS A DEMONSTRATION OF OUTPUT RESULTS
OF ASCII CHARACTERS RECEIVED FROM THE RMX/80 - COMPATIBLE
KEYBOARD INPUT HANDLER SUBROUTINE (KIHS) AND POSTED TO
THIS TELEPRINTER DEVICE.

END. 11/07/77

```
ASM80 :F1:KIHS.SRC

ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        KIHS     PAGE    1

 LOC  OBJ          SEQ          SOURCE STATEMENT

                     1     ;
                     2     ;       RMX/80 - BASED GENERAL KEYBOARD
                     3     ;              INPUT HANDLER SUBROUTINE (KIHS)
                     4     ;
                     5     ;       NOTE - 1) ALL "RQ" - NAMED PARAMETERS ARE
                     6     ;                 RMX/80 DESIGNATED SYSTEM DIRECTIVES
                     7     ;              2) MESSAGES & EXCHANGES ARE IMPLEMENTE
                     8     ;                 ACC. TO STD. RMX/80 NOMENCLATURE
                     9     ;              3) DISPLAY-TYPE CODE (DTYPE) IS SET:
                    10     ;                    65 = NORMAL CHARACTER
                    11     ;                    75 = LINE FEED CHARACTER
                    12     ;                    76 = CARRIAGE RETURN CHARACTER
                    13     ;                    77 = DELETE OR RUBOUT CHARACTER
                    14     ;              4) INPUT-TYPE CODE (ITYPE) IS SET:
                    15     ;                    80H = NOT SET
                    16     ;                    81H = CARRIAGE RETURN
                    17
                    18                    NAME    KIHS
                    19
                    20                    PUBLIC  KIHS, INPEXC, RQL5EX, CHAR
                    21
                    22                    EXTRN   RQWAIT, RQSEND, RQELVL, DISEXC
                    23
000A                24 LF       EQU     10      ; LINE FEED
000D                25 CR       EQU     13      ; CARRIAGE RTN
007F                26 DEL      EQU     7FH     ; DELETE
0032                27 KBI      EQU     32H     ; KEYBOARD IN PORT#
                    28
                    29
                    30          CSEG
                    31
                    32 KIHS:
0000 0E05           33          MVI     C,5     ; ENABLE LEVEL 5
0002 CD0000    E    34          CALL    RQELVL  ; INTERRUPT
0005 010000    D    35 B1:      LXI     B,RQL5EX        ; WAIT FOR A
0008 110000         36          LXI     D,0     ; CHARACTER FROM
000B CD0000    E    37          CALL    RQWAIT  ; THE KEYBOARD
000E,210500    C    38          LXI     H,B1    ; SET RETURN
0011 E5             39          PUSH    H       ; ON THE STACK
0012 3E80           40          MVI     A,80H   ; INITIALIZE
0014 321D00    D    41          STA     ITYPE   ; INPUT CHARACTER TYPE
                    42
0017 DB32           43          IN      KBI     ; GET THE CHARACTER
0019 322300    D    44          STA     CHAR    ; SAVE CHARACTER
001C FE20           45          CPI     20H     ; IS IT A CONTROL
001E DA2B00    C    46          JC      CNTRL   ; YES GO DECODE
0021 FE7F           47          CPI     DEL     ; IS IT DELETE ?
0023 CA5600    C    48          JZ      C3      ; YES, OTHERWISE
0026 3E41           49          MVI     A,65    ; SET FOR SUBSEQ.
0028 C33200    C    50          JMP     DISOT   ; CHARACTER DISPLAY
                    51
                    52 CNTRL:
```

```
LOC  OBJ           SEQ          SOURCE STATEMENT

002B FE0A            53          CPI     LF      ; IS CHAR. A LINE FEED?
002D C23F00    C     54          JNZ     C2      ; NO
0030 3E4B            55          MVI     A,75    ; YES, SET CODE & OUTPUT
0032 322200    D     56 DISOT:   STA     DTYPE
0035 010000    E     57          LXI     B,DISEXC        ; SEND MSG TO THE
0038 111E00    D     58          LXI     D,DISMSG        ; DISPLAY EXCHANGE
003B CD0000    E     59          CALL    RQSEND
003E C9              60          RET
                     61
003F FE0D            62 C2:      CPI     CR      ; IS IT CARRIAGE RTN?
0041 C0              63          RNZ             ; NO, IGNORE CHAR.
0042 3E4C            64          MVI     A,76    ; YES, SET
0044 CD3200    C     65          CALL    DISOT   ; CODE FOR CR & ALSO
0047 3E81            66          MVI     A,81H   ; SET INPUT TERMINATE
0049 321D00    D     67          STA     ITYPE   ; CODE
004C 010F00    D     68          LXI     B,INPEXC        ; SEND MSG TO THE
004F 111900    D     69          LXI     D,INPMSG        ; INPUT EXCHANGE
0052 CD0000    E     70          CALL    RQSEND
0055 C9              71          RET
                     72
                     73 C3:
0056 3E4D            74          MVI     A,77    ; SET DEL CODE
0058 C33200    C     75          JMP     DISOT   ; & OUTPUT
                     76
                     77          DSEG
                     78
000F                 79 RQL5EX:  DS      15      ; LEVEL 5 EXCHANGE
000A                 80 INPEXC:  DS      10      ; INPUT EXCHANGE
                     81
0004                 82 INPMSG:  DS      4       ; INPUT MESSAGE
0001                 83 ITYPE:   DS      1       ; INPUT TYPE
                     84
0004                 85 DISMSG:  DS      4       ; DISPLAY MESSAGE
0001                 86 DTYPE:   DS      1       ; DISPLAY TYPE
0002                 87 CHAR:    DS      2       ; ASCII CHARACTER
                     88
                     89          END

PUBLIC SYMBOLS
CHAR   D 0023    INPEXC D 000F    KIHS   C 0000    RQL5EX D 0000

EXTERNAL SYMBOLS
DISEXC E 0000    RQELVL E 0000    RQSEND E 0000    RQWAIT E 0000

USER SYMBOLS
B1     C 0005    C2     C 003F    C3     C 0056    CHAR   D 0023    CNTRL  C 002
D*SEXC E 0000    DISMSG D 001E    DISOT  C 0032    DTYPE  D 0022    INPEXC D 000
       A 0032    KIHS   C 0000    LF     A 000A    RQELVL E 0000    RQL5EX D 000

ASSEMBLY COMPLETE,  NO ERRORS
```

Ref. #G5

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☐ Other _____ (use additional sheets if necessary)

Program Title

HND534 - RMX/80 Driver for iSBC/534

Function

RMX/80 Driver for the iSBC 534 Communications Expansion Board.

Required Hardware

iSBC 80/10, 80/10A, 80/20 or 80/20-4 with iSBC 534

Required Software

RMX/80 (any version)

Input Parameters

none

Output Results

none

Available on diskette only for $35.00.

Registers Modified: ALL	Programmer: Joe Barthmaier & Steve Verleye
RAM Required: 256	Company: Intel Corp.
ROM Required: 1555	Address:
Maximum Subroutine Nesting Level: 20	City:
Assembler/Compiler Used: PL/M 80 Ver. 3.1	State:

Ref.#G6

insite T.M. INTEL® USER'S LIBRARY SUBMITTAL FORM

☐ 4004/4040 ☐ 8008 ☒ 8080 ☐ 8048 ☐ 8085 ☒ Other RMX/80 (use additional sheets if necessary)

Program Title

RMX/80 DEMONSTRATION PROGRAM WITH TIME

Function

This program provides an excellent demonstration of the timing accuracy of the RMX/80 SYSTEM. It consists of the RMX/80 demonstration program with an added task, RMXCLK. RMXCLK keeps and displays the time on the output device at user command. The clock cycle is 24 hours.

Required Hardware

System 8020 connected to MDS 800 via ICE 80 as per Appendix E, RMX/80 User's Guide.

Required Software

RMX/80 package for SBC 8020 V.1.2.

Input Parameters

The user types 'TIME' on the input device to obtain the current time, or 'TIME XX XX XX' to set the time to a particular value.

Output Results

The out put device types: 'THE TIME IS XX:XX:XX'

The source tape includes:

1) CLK module
2) A modified version of RMX/80 DEMONSTRATION CONFIGURATION MODULE
3) A submit file to change DEMO.LIB to DEMO2.LIB
4) A submit file to load the demonstration program

Registers Modified:	Programmer: Michael Sussman
RAM Required:	Company: Amscor
ROM Required:	Address: 2512 N. Velasco
Maximum Subroutine Nesting Level:	City: Angleton
Assembler/Compiler Used: ISIS; PL/M 80, ASM 80	State: Texas 77515

SECTION 12

CROSS REFERENCE TO 8048 PROGRAMS

HIGHLIGHTS

NEW PROGRAMS IN THE AUGUST UPDATE

IBM BI-SYNC CRC16 GENERATION SUBROUTINE

Generates IBM CRC 16 Check Bytes using polynomial
$X^{16} + X^{15} + X^2 + 1$

Ref.#AB165 p.4-751

MODIFIED SDK-80 RESTART ROUTINE

The original SDK-80 monitor restart routine destroys the carry bit because of the use of the DAD SP instruction before the flags are saved; the DAD SP instruction affects the carry bit. This routine does not destroy the carry bit.

Ref.#AB168 p.4-759

MDS SERIES II - DUMB TERMINAL

Allows MDS Series II keyboard and CRT to be used as a "dumb" terminal using Serial Port 2 on back panel.

Ref.#AB169 p.4-761

FILE GENERATOR

To create and load a source file from an off-line terminal into an ISIS file.

Ref.#AB170 p.4-763

PROGRAMMABLE SOFTWARE TIMERS

This program allows a user to set a software timer by specifying the time (# of counts) and vector (Address of subroutine) to be executed when the timer expires. This program allows for 24 timers.

Ref.#AD7 p.4-765

MTL DIV - SUBROUTINES

Multiplication of two 24 bit binary numbers to give a 48 bit result. Integer division of a 48 bit binary number by a 24 bit binary number.

Ref.#BB53 p-5-329

SQUARE ROOT FOR MCS-48

This routine generates an 8 bits root of a 16 bits number.

Ref.#BB54 p.5-331

FLOATING POINT LOAD AND STORE SUBROUTINES

Six routines to assist in storing and loading floating point numbers as defined in the INSITE floating point packages (Ref.#BC5).

Ref.#BC33 p.5-333

RTCOPY

Copies first file from a PDP-11 diskette (RT-11) to an MDS ISIS-II diskette file.

Ref.#C35 p.6-78

FILES

This module consists of a group of utility procedures which ease file oriented I/O under ISIS-II.

Ref.#D64 p.7-201

PROGRAM HIGHLIGHT

PILOT-80 ISIS-II Version 2.0

PILOT is a programming system for controlling interactive conversation. PILOT stands for Programmed Inquiry, Learning or Teaching. It has most commonly been used as an author language for Computer-Assisted Instruction. It was first developed at the University of California in San Francisco. It is an interpreter written in assembly language that requires less than 4K bytes of memory for the interpreter code. Total memory requirements depend on the space required for PILOT program text and will often be no more than 8K bytes.

PILOT is designed to be simple in its syntax so that those without prior computer experience can easily learn to control its features. Dialogue programs can be rapidly constructed and tested.

It is possible to imitate the operation of PILOT in a general purpose programming language such as BASIC, APL, FORTRAN, or PL/I. The reverse is certainly not true, for PILOT is a specialized language oriented toward dialogs, drills, texts, etc. If a BASIC program is to be made interactive for the handling of free-response dialog, the programmer must expend a large effort in processing input and arranging for comparisons with possible words or portions of words that might be important to recognize.

PILOT makes this kind of processing easy and keeps the program sufficiently readable that it can be reviewed by someone with a primary interest in the logic of how language content is to be presented. PILOT is relatively poor at kinds of computation handled well by general purpose languages. For many instructional uses, that doesn't matter, but some versions of PILOT allow a mix of programming with a concurrently available general language.

Ref. #F16 p.9-28

HIGHLIGHTS

Program Highlight

GRAPH

The subroutine GRAPH can be used to generate a simple one-quadrant graph of one variable on a console device such as a teletype or CRT. Along with the function, plotted as X's, a y-axis, an x-asis and a 5 x 5 coordinate grid will be plotted.

The subroutine produces one line of graph per call. The user can therefore number the axes as he pleases through the calling program. Provided the data to be plotted is scaled such that a zero byte corresponds to the 'value' of the left-most column on the output device, a 2-quadrant graph can be produced though the plotted x-axis will not correspond to the line y = 0.

The graph produced has its y (dependant) axis increasing to the right, and its x (independent) axis increasing downwards. Therefore although the y-coordinates are limited to between 0 and 128 or the width of the output device, the x-coordinate is unlimited.

The fact that the subroutine produces only one line of graph per call allows some kluging. For instance, two functions could be plotted on the same graph by setting up two data blocks in memory, then alternately adding and subtracting to the pointer registers (H and L) between call to give a graph which alternates between the two functions.

Or, if a graph is to be plotted for negative as well as positive values of x, make use of the fact that the E - register holding OFFH signifies to the subroutine that the current call is the first, therefore the y-axis is to be plotted, and that subsequently the subroutine maintains E as a counter for the grid, ranging from 0 to 4. Put the appropriate value of 0 to 4 into the E register before the first call. Then when the graph has plotted up to the line x = -1 (determined by the calling program), put the value OFFH into the E register (which should have held the value 4 in order that the grid be aligned properly) and the next call to GRAPH will plot the y-axis.

Program Highlight

INTELLEC MDS Diagnostic Confidence Test

The Intellec Microcomputer Development System Diagnostic Confidence Test is a simple verification test that exercises standard Intellec Modules and input/output (I/O) devices returning a pass or fail indication to the operator. The confidence test will verify the functionality of:

- Intellec Central Processor Unit
- RAM memory (minimum of 16K or RAM located at 0-16K, plus any additional contigious or non-contigious blocks of RAM).
- Teletypewriter (TTY)
- CRT
- Diskette Drives
- Line Printer
- High Speed Paper Tape Reader (PTR)
- High Speed Paper Tape Punch (PTP)

The program will reside on diskette or on paper tape. The diskette version is in object format and execution is started automatically after completing the start-up procedures for the diskette.

The paper tape version is in a Hex format and can be read in through the PTR or the paper tape reader on the TTY. Both the diskette and paper tape versions require parts of the monitor to be functional to load the program into the system.

The program is structured to avoid a hang up on any operator input. All operator inputs are subject to a software time out that will allow the program to continue executing if no inputs are supplied by the operator within the allotted time. This feature will allow the program to run to completion unsupervised, while testing as much of the system as possible. The program will accept the ALT MODE key from control consoles not having an ESC key.

Total execution time varies with the amount of operator intervention from a minimum execution time on the order of 5 minutes.

HARDWARE REQUIREMENTS

The confidence test hardware requirements include:

- Standard Intellec Microcomputer Development System: which includes the 8080 CPU module, a 16K RAM memory module, a
- Control Console Device: which can be a TTY or a CRT.
- Program Input Device: the program input device is used to transfer the program to RAM memory and is a diskette drive for the diskette versions of a PTR or TTY reader for the paper tape versions of the confidence test.

Beyond this minimum requirement, the program also checks for and exercises:

- Other blocks of RAM memory
- CRT
- TTY
- Diskette Drives
- Line Printer
- High Speed Paper Tape Reader
- High Speed Paper Tape Punch

P R O G R A M H I G H L I G H T

OPTIMISED ULTRA-FAST FLOATING-POINT PACKAGE

The OPTIMISED ULTRA-FAST FLOATING-POINT PACKAGE was designed for an application in which floating-point operation was required, but where speed of operation was more important than great precision. The Package handles numbers as a one byte exponent, and a two byte mantissa. The exponent is in 7 bits in excess-64 form, and the 8th bit is a guard bit for distinguishing between overflow and underflow. The mantissa is in two's complement form and its magnitude is normalised within the range 0.5 to 1.0-2**(-15). Thus 1.0 is represented as 0.5 * 2**1, and is stored in three bytes:

```
41      ; Exponent (2**1 plus excess-64)
00      ; Mantissa, low byte
40      ;    "      high byte
```

Zero is represented by three bytes of 0. The dynamic range of numbers is 2.71 * 10**(-20) to 9.22 * 10**18, and the accuracy of representation is about 4 1/2 decimal figures. Again for speed of operation, where least significant bits are discarded, truncation, rather than rounding, is done.

The Package consists of several small utility routines for shifting numbers, etc., and routines to perform negation, addition, multiplication, division and square-root. All routines within the Package expect appropriately normalised operands, and they return normalised results. The routines will return a result very quickly if one operand is zero. The routines will not produce meaningful results with unnormalised operands.

The Package is organised to greatly simplify and speed chained arithmetic operations. All the routines expect their operand(s) to have been already loaded into the appropriate registers, and leave their result in the appropriate registers for the first operand of the next operation. The first (or only) operand and the result are transferred in the registers B (exponent) DE (mantissa), and the second operand (if any) in A (exponent) HL (mantissa). Chained arithmetic is most efficiently organised in Reverse Polish Notation (RPN). Intermediate results can be left on the stack (by PUSH B , PUSH D), and reloaded as future operands (by POP D , POP B for a first operand, or by POP H , POP PSW for a second operand).

Subtraction is performed by negation and then addition. The division routine has two entry points, one for normal division, and one for reversed division. The normalise section of the addition routine is used for converting two's complement integers into floating point format.

Conditions of overflow and underflow are detected and the result set to +/- the largest possible number, and zero respectively. A floating-point error flag byte is maintained in which bits are set to indicate the occurence of overflow, underflow, division by zero, and the square-root of a negative number. The flag bits are set but never cleared by the routines, and so the user may clear this byte before starting an operation or series of operations, and then test the flag byte and take such action as is appropriate.

With the exception of the error flag byte, all working space is allocated on the stack, and so the routines are fully reentrant. Typical times for execution of the routines on an INTEL 8080A system running at 2Mhz, and with no wait states are:

	typical (ms.)	maximum
Add	0.25	0.53
Subtract	0.27	0.55
Multiply	0.75	0.85
Divide	0.75	0.85
Square-root	1.50	<2.00

S.N.Cope & S.E.Evans
Dept. Engineering Science
Oxford University
Oxford, England

Program Highlight

STAGE2

The STAGE2 macro processor is a very versatile software tool, developed by W. M. Waite. Its uses range from simple filtering to language translation, as well as most conventional macro processing applications.

The syntax of STAGE2 inputs may vary widely as STAGE2 employs a generalized pattern matching facility to select the appropriate macro. For example, suppose one wishes to manipulate every line of input which begins with APPLE and ends with a question mark. The STAGE2 macro template,

APPLE'?

expresses this (the apostrophe indicates that part of the template which may vary). The string of characters which actually appears between APPLE and ? in the input are made available to the macro body as a parameter. STAGE2 allows nine such parameters in each macro template.

The STAGE2 processor provides a dynamically allocated memory so that information may be saved by one macro and later used by another macro. One use of this memory is the normal symbol definition facility provided by most assemblers. More sophisticated uses are possible, including a simple form of associative memory.

Conditional macro expansion facilities are provided so that fully general, input directed transformations are possible. STAGE2 provides nine-channel input/output support, including a simple formatted output facility. Number conversion and arithmetic expression evaluation are also provided.

Because STAGE2 is so general, it can in theory perform any computational process. However, this generality also makes STAGE2 a relatively slow, (compute-bound) processor, and hence unsuitable for some applications. Its power lies in its generalized pattern matching facility. Applications which require such a facility and perform straight forward transformations on input data, will typically be an order of magnitude simpler to implement with STAGE2 than with a normal algorithmic programming language.

Program Highlight

String Manipulation Package

The string manipulation package is a set of four mutually independent subroutines which perform commonly used functions in non-numerical data processing. These functions are the following:

- Substring Access and Assignment

  Procedure SUBST extracts a substring from the source variable and stores it into a substring of the destination variable. It performs range checking, ensuring that no characters will be written outside of the destination variable's bounds.

- String Assignment

  Procedure ASSGN transfers the contents of the source variable to the destination variable. It also performs range checking.

- String Comparison

  Procedure CMPST compares the contents of one string to another, indicating whether they are identically equal, whether one string is a prefix of the other, or whether they don't match.

- String Delimiting

  Procedure DELMT is the most powerful part of the package. The user gives it three strings. They are:

  (a) The object set, which contains string which will be scanned.

  (b) The initial set, which contains the characters to bypass.

  (c) The delimiting set, which contains the possible characters in the symbol the user wants to delimit.

DELMT starts scanning the object string at the specified position, ignoring characters in the initial set until it finds a character not in the initial set. It marks that position, then

continues scanning, making sure that the characters it sees are in the delimiting set. As soon as it finds a character not in the delimiting set, it stops. To the user, it returns the following information:

- Whether the delimiting was possible
- The length of the delimited string
- Its starting position in the object string
- The index to the character immediately following the delimited string

The clearest use for this procedure is in lexical analysis. One can, for example, scan a line input to an assembler, ignore blanks, and extract an alphanumeric symbol.

Included in the package are two utility routines needed by all of the programs. One passes the parameters. The other bypasses the parameters upon return.

The strings on which the subroutines operate consist of up to 255 contiguous bytes preceeded by two control bytes, the first specifying the maximum length, and the second specifying the current length of the string.

There are no specific hardware requirements for this package.

HIGHLIGHTS

INSITE USER'S LIBRARY

8 BIT PROGRAM INDEX

INTEL CORPORATION, 3065 Bowers Avenue, Santa Clara, California 95051 • (408) 987-8080

Printed in U.S.A./B-24/0378/1.3K NCG