## SECTION 5
### MATH AND NUMERICAL MANIPULATION PROGRAM

<table>
<thead>
<tr>
<th>REFERENCE NUMBER</th>
<th>PROGRAM Description</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC1</td>
<td>Floating Point Math Package</td>
<td>5-1</td>
</tr>
<tr>
<td>BC2</td>
<td>Floating Point Format Conversion Package</td>
<td>5-5</td>
</tr>
<tr>
<td>BB1</td>
<td>16-Bit Multiply (32-Bit Result)</td>
<td>5-9</td>
</tr>
<tr>
<td>BB2</td>
<td>MYP16 16-Bit Multiply - 16-Bit Result</td>
<td>5-12</td>
</tr>
<tr>
<td>BB3</td>
<td>16-Bit Multiply - 16-Bit Result</td>
<td>5-15</td>
</tr>
<tr>
<td>BB4</td>
<td>SMPY16 16-Bit 2's Complement Signed Multiplication</td>
<td>5-19</td>
</tr>
<tr>
<td>BB5</td>
<td>Binary Multiplication - 24-Bit</td>
<td>5-24</td>
</tr>
<tr>
<td>BB8</td>
<td>16-Bit Division - 16-Bit Results</td>
<td>5-29</td>
</tr>
<tr>
<td>BB9</td>
<td>DIV16 16-Bit Division - 16-Bit Result</td>
<td>5-33</td>
</tr>
<tr>
<td>BB10</td>
<td>BIN to BCD Conversion Routine</td>
<td>5-36</td>
</tr>
<tr>
<td>BB11</td>
<td>BCD to BIN Conversion Routine</td>
<td>5-41</td>
</tr>
<tr>
<td>BB12</td>
<td>BCD to/from Binary Conversion</td>
<td>5-45</td>
</tr>
<tr>
<td>BB14</td>
<td>Gray to Binary Conversion</td>
<td>5-50</td>
</tr>
<tr>
<td>BA1</td>
<td>BCD Sum for 8008</td>
<td>5-54</td>
</tr>
<tr>
<td>BA15</td>
<td>Log Base 2</td>
<td>5-57</td>
</tr>
<tr>
<td>BA16</td>
<td>Digital to Analog Conversion for Eight Outputs</td>
<td>5-62</td>
</tr>
<tr>
<td>BA17</td>
<td>Binary to HEX Routine</td>
<td>5-67</td>
</tr>
<tr>
<td>BA18</td>
<td>Binary to BCD Subroutine</td>
<td>5-72</td>
</tr>
<tr>
<td>BA2</td>
<td>HEX to Decimal Conversion</td>
<td>5-76</td>
</tr>
<tr>
<td>BA19</td>
<td>BCD Input and Direct Conversion to Binary Routine</td>
<td>5-78</td>
</tr>
<tr>
<td>BA4</td>
<td>BINDECBIN - Binary to/from BCD</td>
<td>5-81</td>
</tr>
<tr>
<td>BC3</td>
<td>MATH - Fixed and Floating Point Arithmetic Routines</td>
<td>5-83</td>
</tr>
<tr>
<td>BC4</td>
<td>Elementary Function Package</td>
<td>5-85</td>
</tr>
<tr>
<td>BC5</td>
<td>8080 Floating Point Package with BCD Conversion Routine</td>
<td>5-87</td>
</tr>
<tr>
<td>BC6</td>
<td>8080 Least Squares Quadratic Fitting Routine</td>
<td>5-89</td>
</tr>
<tr>
<td>BC7</td>
<td>Floating Point Procedures</td>
<td>5-91</td>
</tr>
<tr>
<td>BC8</td>
<td>PL/M Floating Point Interface</td>
<td>5-93</td>
</tr>
<tr>
<td>BC9</td>
<td>Floating Point Decimal and HEX Format Conversion</td>
<td>5-95</td>
</tr>
<tr>
<td>BB20</td>
<td>N-BYTE Binary Multiplication and Leading Zero Blanking</td>
<td>5-101</td>
</tr>
<tr>
<td>BA5</td>
<td>Subroutine DMULT (Decimal Multiplication)</td>
<td>5-107</td>
</tr>
<tr>
<td>BA6</td>
<td>BCD Multiplication</td>
<td>5-109</td>
</tr>
<tr>
<td>BB21</td>
<td>MUL/DIV Multi-Precision Pack for 8080</td>
<td>5-115</td>
</tr>
<tr>
<td>BB22</td>
<td>DMPY - Double Precision Multiply</td>
<td>5-120</td>
</tr>
<tr>
<td>BB23</td>
<td>16-Bit Square Root Routine</td>
<td>5-123</td>
</tr>
<tr>
<td>BC10</td>
<td>Floating Point Square Root</td>
<td>5-127</td>
</tr>
<tr>
<td>BB24</td>
<td>SQRTF - Calculates 8-Bit Root of 16-Bit Number</td>
<td>5-133</td>
</tr>
<tr>
<td>BC11</td>
<td>Subroutine SQRT</td>
<td>5-137</td>
</tr>
<tr>
<td>BC12</td>
<td>Fast Floating Point Square Root Routine</td>
<td>5-141</td>
</tr>
<tr>
<td>BB25</td>
<td>Natural Logarithm</td>
<td>5-146</td>
</tr>
<tr>
<td>BC13</td>
<td>Subroutine Log - Common Logarithms</td>
<td>5-148</td>
</tr>
<tr>
<td>BB26</td>
<td>Approximating Routine</td>
<td>5-153</td>
</tr>
<tr>
<td>BB27</td>
<td>SIN X, COS X Subroutine</td>
<td>5-158</td>
</tr>
<tr>
<td>BB28</td>
<td>RMSTF - Integration Routine</td>
<td>5-162</td>
</tr>
<tr>
<td>BB29</td>
<td>32-Bit Binary to BCD Conversion, Leading Zero Blanking</td>
<td>5-167</td>
</tr>
<tr>
<td>BB31</td>
<td>Fixed Point CHEBYSHEV Sine and Cosine for PL/M Users</td>
<td>5-171</td>
</tr>
<tr>
<td>BB32</td>
<td>32-Bit Divide Subroutine</td>
<td>5-175</td>
</tr>
<tr>
<td>BB33</td>
<td>8-Bit Multiply and Divide</td>
<td>5-177</td>
</tr>
<tr>
<td>BB34</td>
<td>Double Precision Integer Arithmetic Package</td>
<td>5-180</td>
</tr>
<tr>
<td>BC14</td>
<td>XMATH - 8080 Floating Point Extended Math Package</td>
<td>5-183</td>
</tr>
</tbody>
</table>
Floating Point Package for Intel 8008 and 8080 Microprocessors 5-187
Random Number Generator 5-189
8-Bit Random Number Generator 5-192
16-Bit Random Number Generator 5-195
PL/M Histogram Procedure and Random Number Generator 5-199
RANDOM$BITS - Random Number Generator 5-204
Factorial of a Decimal Number 5-209
BCD Up/Down Counter 5-212
3-Byte Positive Fractional Multiply 5-213
Algebraic Compare Subroutine 5-214
8080 Double Precision ARC Tangent 5-216
ARC.TAN 2 Subroutine 5-220
Floating Point Interpreter 5-222
12 X 12 Multiply 5-228
MBCD: N1 Bytes X N2 Bytes Decimal Multiply Subroutine 5-232
Histogram 5-238
ASCII to EBCDIC and EBCDIC to ASCII Converters 5-240
Conversion of Scientific to Easily Readable Notation 5-242
DTMHEX 5-249
8048 BCD Multiply 5-252
Hex to ASCII Conversion 5-256
Optimised Ultra Fast Floating Point Package 5-260
ASCII String to Intel Floating Point/Intel Floating Point to ASCII String 5-262
8080 Floating Point Ab 5-274
Floating Point Utility Programs for Use with FPAL.LIB / SBC-310
Floating Point System for Use with SBC 80/20 5-282
Relocatable FMath and XMath, 8085 Floating Point Package 5-286
8048-DIV -- Division Routine 5-288
ARRAY ADDRESSING SUBROUTINE AND CALLING MACRO 5-292
## SECTION 5

### MATH AND NUMERICAL MANIPULATION PROGRAMS

<table>
<thead>
<tr>
<th>REFERENCE NUMBER</th>
<th>PROGRAM</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA1</td>
<td>BCD SUM FOR 8008.</td>
<td>5-54</td>
</tr>
<tr>
<td>BA2</td>
<td>HEX TO DECIMAL CONVERSION</td>
<td>5-76</td>
</tr>
<tr>
<td>BA4</td>
<td>BINDECBIN - BINARY TO/FROM BCD.</td>
<td>5-81</td>
</tr>
<tr>
<td>BA5</td>
<td>SUBROUTINE DMULT (DECIMAL MULTIPLICATION)</td>
<td>5-107</td>
</tr>
<tr>
<td>BA6</td>
<td>BCD MULTIPLICATION.</td>
<td>5-109</td>
</tr>
<tr>
<td>BA11</td>
<td>FACTORIAL OF A DECIMAL NUMBER</td>
<td>5-209</td>
</tr>
<tr>
<td>BA12</td>
<td>BCD UP/DOWN COUNTER</td>
<td>5-212</td>
</tr>
<tr>
<td>BA13</td>
<td>ARC. TAN 2 SUBROUTINE</td>
<td>5-220</td>
</tr>
<tr>
<td>BA14</td>
<td>MBCD: N1 BYTES X N2 BYTES DECIMAL MULTIPLY SUBROUTINE</td>
<td>5-232</td>
</tr>
<tr>
<td>BA15</td>
<td>CONVERSION OF SCIENTIFIC TO EASILY READABLE NOTATION.</td>
<td>5-242</td>
</tr>
<tr>
<td>BA16</td>
<td>8048 BCD MULTIPLY</td>
<td>5-252</td>
</tr>
</tbody>
</table>
16-BIT MULTIPLY (32-BIT RESULT) .......................... 5-9
MPY16 16-BIT MULTIPLY - 16-BIT RESULT .................. 5-12
16-BIT MULTIPLY - 16-BIT RESULT ......................... 5-15
SMPY16 16-BIT 2'S COMPLEMENT SIGNED MULTIPLICATION .. 5-19
BINARY MULTIPLICATION - 24-BIT .......................... 5-24
16-BIT DIVISION - 16-BIT RESULTS ......................... 5-29
DIV16 16-BIT DIVISION - 16-BIT RESULT .................. 5-33
BIN TO BCD CONVERSION ROUTINE ......................... 5-36
BCD TO BIN CONVERSION ROUTINE ......................... 5-41
BCD TO/FROM BINARY CONVERSION ......................... 5-45
GRAY TO BINARY CONVERSION .............................. 5-50
LOG BASE 2. ............................................. 5-57
DIGITAL TO ANALOG CONVERSION FOR EIGHT OUTPUTS ... 5-62
BINARY TO HEX ROUTINE .................................. 5-67
BINARY TO BCD SUBROUTINE ................................ 5-72
BCD INPUT AND DIRECT CONVERSION TO BINARY ROUTINE .. 5-78
N-BYTE BINARY MULTIPLICATION AND LEADING ZERO BLANKING ........................................... 5-101
MUL/DIV MULTI-PRECISION PACK FOR 8080 .................. 5-115
DPY - DOUBLE PRECISION MULTIPLY ...................... 5-120
16-BIT SQUARE ROOT ROUTINE .............................. 5-123
SORTF - CALCULATES 8-BIT ROOT OF 16-BIT NUMBER ...... 5-133
NATURAL LOGARITHM ...................................... 5-146
APPROXIMATING ROUTINE .................................. 5-153
SIN X, COS X SUBROUTINE .................................. 5-158
RMSTF - INTEGRATION ROUTINE ......................... 5-162
32-BIT BINARY TO BCD CONVERSION, LEADING ZERO BLANKING 5-167
FIXED POINT CHEBYSHEV SINE AND COSINE FOR PLM USERS 5-171
32-BIT DIVIDE SUBROUTINE ................................ 5-175
8-BIT MULTIPLY AND DIVIDE ................................ 5-177
DOUBLE PRECISION INTEGER ARITHMETIC PACKAGE ....... 5-180
RANDOM NUMBER GENERATOR .............................. 5-189
8-BIT RANDOM NUMBER GENERATOR ......................... 5-192
16-BIT RANDOM NUMBER GENERATOR ....................... 5-195
PL/M HISTOGRAM PROCEDURE AND RANDOM NUMBER GENERATOR 5-199
RANDOM#BITS - RANDOM NUMBER GENERATOR ............. 5-204
3-BYTE POSITIVE FRACTIONAL MULTIPLY .................. 5-213
ALGEBRAIC COMPARE SUBROUTINE ......................... 5-214
8088 DOUBLE PRECISION ARC TANGENT .................... 5-216
12 X 12 MULTIPLY ....................................... 5-228
HEX TO ASCII CONVERSION .............................. 5-256
8048-DIV -- DIVISION ROUTINE ....................... 5-288
ARRAY ADDRESSING SUBROUTINE AND CALLING MACRO .. 5-292
PL/M MULTIPLE PRECISION ARITHMETIC .................. 5-307
MATH48 - EXTENDED PRECISION ARITHMETIC .............. 5-309
32 X 16 DIVIDE SUBROUTINE - DV3232 .................... 5-321
BINGRY - BINARY TO GRAY CODE CONVERSION SUBROUTINE . 5-327
MULTIPLY/DIVIDE SUBROUTINES:
- MULTIPLY 24 BIT FOR 48 BIT QUOTIENT
- DIVIDE 48 BIT BY 24 BIT .................................. 5-329
SQUARE ROOT FOR MCS-48
BC1  FLOATING POINT MATH PACKAGE ........................................... 5-1
BC2  FLOATING POINT FORMAT CONVERSION PACKAGE .................... 5-5
BC3  MATH—FIXED AND FLOATING POINT ARITHMETIC ROUTINES ........ 5-83
BC4  ELEMENTARY FUNCTION PACKAGE ..................................... 5-85
BC5  8000 FLOATING POINT PACKAGE WITH BCD CONVERSION ............. 5-87
BC6  8000 LEAST SQUARES QUADRATIC FITTING ROUTINE.................. 5-89
BC7  FLOATING POINT PROCEDURES ........................................ 5-91
BC8  PL/M FLOATING POINT INTERFACE .................................... 5-93
BC9  FLOATING POINT DECIMAL AND HEX FORMAT CONVERSION .......... 5-95
BC10 FLOATING POINT SQUARE ROOT ........................................ 5-127
BC11 SUBROUTINE SORT .................................................. 5-137
BC12 FAST FLOATING POINT SQUARE ROOT ROUTINE ...................... 5-141
BC13 SUBROUTINE LOG—COMMON LOGARITHMS ................................ 5-148
BC14 XMATH—8000 FLOATING POINT EXTENDED MATH PACKAGE ........... 5-183
BC15 FLOATING POINT PACKAGE FOR INTEL 8080 AND 8088 MICROPROCESSORS ................................................................. 5-187
BC16 FLOATING POINT INTERPRETER ........................................ 5-222
BC18 OPTIMIZED ULTRA FAST FLOATING POINT PACKAGE ............... 5-260
BC19 ASCII STRING TO INTEL FLOATING POINT/
   INTEL FLOATING POINT TO ASCII STRING ................................ 5-262
BC21 8080 FLOATING POINT A*B ........................................... 5-274
BC22 FLOATING POINT UTILITY PROGRAMS FOR USE WITH
   FPAL.LIB/SBC-310
   FLOATING POINT SYSTEM FOR USE WITH SBC 80/20 .................. 5-282
BC23 RELOCATABLE FMATH AND XMATH,
   8085 FLOATING POINT PACKAGE ...................................... 5-286
BC24 FLOATING POINT CONVERSION ROUTINE ............................. 5-301
BC25 LINEAR SYSTEM (GAUSS ELIMINATION)—LISY ......................... 5-303
BC26 GAMMA FUNCTION SUBROUTINE ...................................... 5-305
BC27 ACQUISITION OF A DECIMAL NUMBER FROM MDS CONSOLE,
   CONVERSION TO FPAL FLOATING POINT NUMBER
   AND VICEVERSA ..................................................... 5-311
BC28 DOUBLE PRECISION FLOATING POINT PACKAGE (DPAL.LIB) ...... 5-313
BC29 DISCRETE FOURIER TRANSFORM ..................................... 5-315
BC30 FLOATING POINT CONSTANT CALCULATOR—FCONST .................. 5-317
BC31 FPAL86.LIB—FLOATING POINT LIBRARY ............................. 5-319
BC32 FPAL—EXTENDED MATH PACKAGE .................................... 5-323
BC33 FLOATING POINT LOAD AND STORE SUBROUTINES .................... 5-333

BD1  HISTOGRAM ......................................................... 5-238
BD2  ASCII TO EBCDIC AND EBCDIC TO ASCII CONVERTERS ............ 5-240
BD3  DTMHEX .......................................................... 5-249
FLOATING POINT MATH PACKAGE

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

None

None

Program offered on diskette only.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>MAC8</td>
</tr>
<tr>
<td>63 bytes</td>
<td>Programmer:</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>C.E. Ohme</td>
</tr>
<tr>
<td>768 bytes</td>
<td>Company:</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address: 44750 Winding Lane Fremont, Ca. 94538</td>
</tr>
</tbody>
</table>
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 AC-
  CUMULATOR.
- 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
-0.000000000001 -1.000000E-13
+1.6E5 160000.0
123456789 1.234568E+08
54321E-10 5.432100E-06
-2718281828E-9 -2.718282
FLOACING POINT FORMAT CONVERSION PACKAGE

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

None.

Floating Point Arithmetic and Utility Package

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Assembler/Compiler Used:</td>
</tr>
<tr>
<td>63 bytes</td>
<td>MAC 8</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>512 bytes</td>
<td>C.E. Ohme</td>
</tr>
<tr>
<td>Company: 44750 Winding Lane</td>
<td>Fremont, Ca. 94538</td>
</tr>
</tbody>
</table>
808 BINARY FLOATING POINT SYSTEM

FORMAT CONVERSION PACKAGE

The format conversion package of the 808 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. Two's 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.

<table>
<thead>
<tr>
<th>Decimal Digits</th>
<th>000B-011B BCD Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>360B</td>
</tr>
<tr>
<td>+</td>
<td>373B Plus</td>
</tr>
<tr>
<td>-</td>
<td>375B Minus</td>
</tr>
<tr>
<td>*</td>
<td>376B Decimal Point</td>
</tr>
<tr>
<td>Exponential Sign</td>
<td>025B Letter E</td>
</tr>
</tbody>
</table>

(These octal representations can be converted to the corresponding ASCII characters by adding 060B to each.)

The output routine generates character strings in two formats, each consisting of 17 characters. The format used in a specific case is dependent upon the magnitude of the value represented.
ZERO AND MAGNITUDES BETWEEN 0.000000 AND 99999999. 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.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.141593</td>
<td>3.141593</td>
</tr>
<tr>
<td>-.0000000000001</td>
<td>-1.000000E-13</td>
</tr>
<tr>
<td>+1.6E5</td>
<td>1600000.0</td>
</tr>
<tr>
<td>123456789</td>
<td>1.234568E+09</td>
</tr>
<tr>
<td>54321E-10</td>
<td>5.432100E-06</td>
</tr>
<tr>
<td>-2718281828</td>
<td>-2.718282E+09</td>
</tr>
</tbody>
</table>

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

16 BIT MULTIPLY - 32 BIT RESULT

MULTIPLICATION OF TWO 16 BIT POSITIVE NUMBERS

Required Hardware

BASIC 8080 SYSTEM

Required Software

23 INSTRUCTIONS

Input Parameters

2 BYTE MULTIPLIER X₁ X₀ IN MEMORY
2 BYTE MULTIPLICAND Y₁ Y₀ IN D&E REGISTERS

Output Results

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>NONE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 BYTES</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>42 BYTES</td>
<td>DAN SOLTZ</td>
</tr>
</tbody>
</table>

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 D21001 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
^17 CA2401 JZ DONE ; AFTER 2ND PASS JUMP TO DONE
  1A E5 PUSH H ; STORE LS RESULT BYTE
0118 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 C30301 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
16-BIT MULTIPLY - 16-BIT RESULT - MPY16

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

NONE

NONE

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

B,C contain result

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A11</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 bytes</td>
<td>MAC 30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>25H bytes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
<th></th>
</tr>
</thead>
</table>
; REF NO. BB2
; PROGRAM TITLE MPY 16

MPY80:

0000 222800   SHLD   TEMP   ; STORE MULTIPLICAND IN TEMPORARY
0001 212A00   LXI    H, BNUM  ; STORE
0006 3611   MVI    M, 11H   ; BIT COUNT
0006 3611   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   ; RIGHT
000F 1F   RAR
0010 5F   MOV    E, A   ; MULTIPLICAND
0011 35   DCR    M     ; DECREMENT BIT COUNT
0012 08   RZ      ; DONE? THEN RETURN
0013 D21F00  JNC    SKIP   ; NO CARRY FROM ROTATE
0100 2A2800  LHL    TEMP   ; OTHERWISE
0109 09   DAD    B
001A 44   MOV    B, H
001B 4D   MOV    C, L
001C 212A00  LXI    H, BNUM

SKIP:

0020 1F   MOV    A, B   ; ROTATE
0021 47   RAR
0022 79   MOV    B, A
0023 1F   RAR
0024 4F   MOV    C, A
0025 C30B00  JMP    LOOP   ; REPEAT LOOP

TEMP:

0028   DS 2

BNUM:

002A   DS 1
0000   END
16-BIT MULTIPLY - 16-BIT RESULT

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

None.

None.

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

B,C contain result.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, D, E, H, L</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 bytes</td>
<td>MAC 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>35H bytes</td>
<td></td>
</tr>
</tbody>
</table>

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
0009 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
0029 1F RAR ; RESULT
002A 47 MOV B,A ; RIGHT
002B 79 MOV A,C ; ONE
0020 1F  RAR          ; BIT
0020 4F  MOV C,A      ; REPEAT STEPS IN LOOP
002E C3E00 JMP LOOP    ; SPACE FOR TEMP MULTIPLICAND
0031 BNUM: DS 2       ; SPACE FOR BIT COUNTER
0033 BNUM: DS 1
0000 END
16-BIT 2'S COMPLEMENT SIGNED MULTIPLICATION - SMPY16

16-Bit Binary Multiplication - 32-Bit Result

None

None

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

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

Registers Modified: All

Maximum Subroutine Nesting Level: 1

RAM Required: 4 bytes

Assembler/Compiler Used: MAC8

ROM Required: 68H

Programmer: R.M. Gabrielson
General Electric

Company: Aerospace Controls
Box 5000, Binghamton NY 13902
S 8080 MACRO ASSEMBLER, V1.0

; REF. NO. B64.
; 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  ; E=LO SUM
0403 79 MOV  A,C
0404 2C INR  L
0405 8E ADC  M  ; HI AUG + HI ADD + LO CARRY
0406 8E ADC  M
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
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
0573 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
SETUP:
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:
0593 21E90B  LXI H, TEST
0596 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
05B5 4F      MOV C, A
05B6 78      MOV A, B
05B7 1F      RAR
05BB 4B             MOV    B, A
05BB 7B             MOV    A, E
05BB 1F             RAR
05BC 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
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. BBS.
; PROGRAM NAME BINARY MULTIPLICATION

;**********************************
; BINARY MULTIPLICATION
;**********************************

; ON ENTERING THIS SUBROUTINE
; HL REG. POINTS TO LS BYTES OF MULTIPLIER
; DE REG. POINTS TO LS BYTES OF MULTIPLICAND
; A REG. HAS THE NUMBER OF MULTIPLIER BITS +1 (2 TO "25")

; ON EXIT
; HL REG. POINTS TO LS BYTES OF PRODUCT
; NOT: MEMORY ALLOCATION FOR PRODUCT

; ENTER: XXXXMMM
; EXIT: PPPPPP
; X IS DON'T CARE
; M IS MULTIPLIER
; P IS PRODUCT

0003 BY3 EQU 3 ; 3 BYTES
0010 LSR EQU 10H ; LS PARTIAL PRODUCT POINTER SAVED LOCATION
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

5-26
DCX H
MOV M,A
DCX H
MOV M,A
SHLD MSR ;SAVE MS PRODUCT POINTER
XCHG
SHLD LSMCD ;SAVE LS MULTIPLICAND POINTER

;TEST FOR ZERO
MVI B,0 ;PICK UP LS MULTIPLIER POINTER
POP H
PUSH H
MOV A,M
DCX H
ADC M
DCX H
ADC M
CMP B ;COMPARE WITH B REG. (<0)
JZ EXIT ;EXIT IF MULTIPLIER = 0

;ROTATE MULTIPLIER ROUND AND TEST
LDA NBIT
MOV B,A ;B REG. HAS NUMBER OF MULTIPLIER BITS
XRA A ;CLEAR CARRY BIT
LOOP:
CALL ROT1 ;CALL ROTATE ROUTINE
DCR B ;DECREMENT COUNT FOR MULTIPLIER BITS
JZ FINISH ;IF ZERO -> FINISH
JNC LOOP ;IF CARRY=0 -> JUST ROTATE

;ADDITION ROUTINE
LHLD LSMCD ;PICK UP LS MULTIPLICAND POINTER
XCHG
LHLD LSR ;PICK UP LS PARTIAL PRODUCT POINTER
MVI C, BY3 ;COUNT FOR 3 BYTES
XRA A ;CLEAR CARRY

ADD1:
LDAX D ;PICK UP MULTIPLICAND
ADC M ;ADD IN TO RESULT
MOV M,A
DCX D
DCX H
DCR C
JNZ ADD1 ;IF ZERO -> FINISH ADDING
JMP LOOP

;ALIGN PRODUCT TO WHERE ILS MULTIPLIER WAS
; FINIS:
0149 3A1600 LDA NBIT
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
015E 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
16-BIT DIVISION - 16-BIT RESULT

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

None.

None.

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

D,E contain the result.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 bytes</td>
<td>MAC 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>40H bytes</td>
<td></td>
</tr>
</tbody>
</table>

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
000B 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 D23300 JNC SKIP ; NO BORROW 50 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  MOY B,A

SKIP:
0033 213E00  LXI H,BNUM     ; RESTORE ADDRESS OF BIT COUNTER
0036 9F  SBB A              ; COMPLEMENT
0037 DE80  SBI 80H          ; CARRY
0039 C3E00  JMP LOOP         ; REPEAT LOOP

TEMP:
003C  DS 2                 ; CELL FOR DIVIDEND

BNUM:
003E  DS 1                 ; CELL FOR BIT COUNTER
0000  END
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:

A11

Maximum Subroutine Nesting Level:

0

RAM Required:

3 bytes

Assembler/Compiler Used:

MAC 80

ROM Required:

31H bytes

Programmer:

Company:

5-33
; 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
0023 47         MOV B, A
0024 C5         PUSH B ; SAVE TEMP RESULT ON STACK
0025 D2A00     JNC SKIP ; NO BORROW FROM SUBTRACT
0028 09         DPD B ; ADD DIVIDEND BACK IN
0029 E3        XTHL ; REPLACE TEMP RESULT ON STACK

   TEMP:
002A 213300     LXI H, BNUM ; RESTORE H & L
002D 3F         CMC ; COMPLEMENT CARRY
002E C30C00     JMP LOOP ; REPEAT LOOP STEPS

0031 DS 2

0033 DS 1

END
BINARY TO BCD CONVERSION ROUTINE

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
; EQ. 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 05 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 MOY A.M ; PICK UP LS BYTE
020B 2B DCX H ; ADD IN NEXT HIGHER BYTE
020C 8E ADC M
020D 2B DCX H
020E 8E ADC M ; ADD IN MS BYTE
020F CA7F02 JZ EXIT ; IF BINARY VALUE = 0 -> EXIT
0222 D24002 JNC TEN5 ; NO CARRY => G.T. 2**15
0225 FE0C CPI 12 ; 2**12 = 4,096 * 10**3
0227 D24402 JNC TEN4 ; NO CARRY => G.T. 2**12
0228 FE08 CPI 8
022A D28002 JNC TEN3
022C D26002 JNC TEN2
022E D24002 JNC TEN1
0230 C37402 JMP TEN0 ; REACHED IF NBIT <= 3

; BEGIN CONVERSION

0232 1698 TEN7: MVI D. 98H ; 98H = 10**2
023E 018096 LXI B. 9680H
0244 CD8202 CALL DIGIT
0244 160F TEN6: MVI D. 0FH ; 0FH = 10**1
0246 014042 LXI B. 4240H
0249 CD8202 CALL DIGIT
024C 1601 TEN5: MVI D. 1 ; 01H = 10**0
024E 01A086 LXI B. 86A0H
0251 CD8202 CALL DIGIT
0254 1600 TEN4: MVI D. 0
0256 011027 LXI B. E0000
0259 CD8202 CALL DIGIT
025C 1600 TEN3: MVI D. 0
025E 01E003 LXI B. 10000
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:

027E E1 POP H ; RETURN HL POINTING TO MS BCD
0280 D1 POP D ; RETURN DE POINTING TO LSB OF BINARY
0281 C9 RET
; SUBTRACTION ROUTINE
; DIGIT:
0282 3600  MVI M, 0 ; INITIALIZE DIGIT
0284  E5    PUSH H ; FOR BCD, DIGIT=0
             ; FOR ASCII, DIGIT=30H
0285  2A1000 LHL H SBIN ; PICK UP BIN POINTER
0286  7E    MOV A, M ; PICK UP LS BYTE
0288  91    SUB C
0289  77    MOV M, A ; PUTS IT BACK
028A  2B    DCX H ; POINTS TO 2ND LS BYTE
028B  7E    MOV A, M
028C  98    SBB B
028D  77    MOV M, A
028E  77    DCX H ; POINTS TO MS BYTE
028F  2B    MOV A, M
0290  9A    SBB D
0291  77    MOV M, A
0292  7A    MOV A, D ; SAVE D REG.
0293  DA9E02 JC RSTR ; IF CARRY RESTORE
0294  E1    POP H ; PICK BCD POINTER
0295  34    INR M ; INCREMENT BCD POINTER
0296  E5    PUSH H ; SAVE BCD POINTER
0297  57    MOV D, A ; RESTORE D REG.
0298  C38502 JMP SUB1 ;
; RESTORE ROUTINE
; RSTR:
029E  57    MOV D, A ; RESTORE D REG.
029F  2A1000 LHL H SBIN ; PICK UP BIN POINTER
02A0  7E    MOV A, M
02A2  81    ADD C
02A3  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
**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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A11</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>MAC 80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>38 bytes</td>
<td>Chon Hock Leow</td>
</tr>
<tr>
<td></td>
<td>Tektronix</td>
</tr>
</tbody>
</table>

**Company:**

Box 500, Dept. 50-447
Beaverton, Oregon 97077
S 8080 MACRO ASSEMBLER, V1.0

; 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

.J300

;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 XOR A, 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

;*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
0317 88 ADC B
0318 29 DAD H
0319 8F ADC A ;AHL <- AHL * 1 10

5-43
031A 47       MOV     B, A    ; SAVE A IN B REG.
031B D1       POP      D     ; RESTORE BCD DIGIT POINTER
031C 1A       LDA     D      ; GET NEW DIGIT
031D 85       ADD     L, A    ; ADD IN NEW DIGIT
031E 6F       MOV     L, A    
031F 70       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
BCD TO/FROM BINARY CONVERSION

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

None.

None.

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

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>MAC 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCD to binary 32 bytes</td>
<td></td>
</tr>
<tr>
<td>Binary to BCD 67 bytes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
<th></th>
</tr>
</thead>
</table>

5-45
; 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

BCDIN:

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 77    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
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
;
; BIN2BCD:
0010 011027    LXI B, 10000
0012 01F00     CALL DIGIT
0014 01E02     LXI B, 1000
0016 01F00     CALL DIGIT
0018 01E02     LXI B, 1000
001A 01F00     CALL DIGIT
001C 010A00    LXI B, 10
001E 01F00     CALL DIGIT
0020 010100    LXI B, 1
0022 01F00     CALL DIGIT
0024 C9        RET

; DIGIT:
0026 3600     MOV M, 0

; INITIALIZE DIGIT
; FOR BCD, DIGIT=0,
; FOR ASCII, DIGIT=30H

; SUBTRACT LOOP
0028 7E        MOV A, E
002A 91        SUB C
002C 5F        MOV E, A
002E 7A        MOV A, D
0030 98        SBB B
0032 57        MOV D, A
0034 FA2100    JMP D11
0036 7E        MOV A, M

; INCREMENT ~D DIGIT
0038 7E        MOV A, M
003A C601     ADI 1
003C 77        MOV M, A
003E C2100     JMP D11

; D11:
0040 7E        MOV A, E

; ADJUST ACCUMULATOR FOR NEXT SEQUENCE
0042 81        ADD C
0044 5F        MOV E, A
0046 7A        MOV A, D
0048 88        ADC B
004A 57        MOV D, A
004C CD2B00    CALL INCHL

; HL + 1 -> HL
0050 C9        RET

; INCHL:
0052 2C        INR L
0030 C0          RNZ
0032 24          INR H
0034 C9          RET
0036             END
**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, E₀ = LSB, UNUSED BITS = 0

**Output Results**

BINARY DATA IN D-E REGISTER, E₀ = LSB

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C,</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>ISIS 8080 Macro Assembler, V1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 BYTES</td>
<td>G. Mercola</td>
</tr>
<tr>
<td></td>
<td>Data Works, Inc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>18725 Bryant St.</td>
<td></td>
</tr>
<tr>
<td>Northridge, Ca.</td>
<td>91324</td>
</tr>
</tbody>
</table>
ALGORITHM:

\[ N_B = \left( \left( \frac{N_G}{2} \right) \lor \left( \frac{N_G}{4} \right) \lor \left( \frac{N_G}{8} \right) \right) \]

Until \( N_G/2^n = 0 \)

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

<table>
<thead>
<tr>
<th>Gray ((N_G))</th>
<th>Binary ((N_B))</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 1000</td>
<td>1111</td>
</tr>
<tr>
<td>14 1001</td>
<td>1110</td>
</tr>
<tr>
<td>13 1011</td>
<td>1101</td>
</tr>
<tr>
<td>12 1010</td>
<td>1100</td>
</tr>
<tr>
<td>11 1110</td>
<td>1011</td>
</tr>
<tr>
<td>10 1111</td>
<td>1010</td>
</tr>
<tr>
<td>9 1101</td>
<td>1001</td>
</tr>
<tr>
<td>8 1100</td>
<td>1000</td>
</tr>
<tr>
<td>7 0100</td>
<td>0111</td>
</tr>
<tr>
<td>6 0101</td>
<td>0110</td>
</tr>
<tr>
<td>5 0111</td>
<td>0101</td>
</tr>
<tr>
<td>4 0110</td>
<td>0100</td>
</tr>
<tr>
<td>3 0010</td>
<td>0011</td>
</tr>
<tr>
<td>2 0011</td>
<td>0010</td>
</tr>
<tr>
<td>1 0001</td>
<td>0001</td>
</tr>
<tr>
<td>0 0000</td>
<td>0000</td>
</tr>
</tbody>
</table>
; 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
BCD SUM FOR 8008

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

MCS8 MICROCP.

PLM 8 COMPILER.

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)

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Assembler/Compiler Used:</td>
</tr>
<tr>
<td>8 + 2n (number of decades. These are the positions of the 2 numbers.)</td>
<td>PLM 8</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>208</td>
<td>E. Massetti</td>
</tr>
<tr>
<td></td>
<td>Lab.: Massetti</td>
</tr>
<tr>
<td></td>
<td>Company: 20133 Milano</td>
</tr>
<tr>
<td></td>
<td>via Ronchi 17, Italy</td>
</tr>
</tbody>
</table>
00001 1 /*REF. NO. BA1 */
00002 1 /*PROGRAM TITLE PROCEDURE BCD SUM FOR 8008 */
00003 1 /*
00004 1 *
00005 1 */
00006 1 /* PROCEDURE FOR SUMMING TWO BCD NUMBER, CIFRE DECADERS 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 RAPRESENTS 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 DECADERS 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 3 ADD1 (N)=(ADD1(N)+ADD2(N)+RIPORTO) AND OFH;
00027 3 /* AND NOW BCD CORRECTION ROUTINE*/
00028 3 IF ADD1(N) >9 THEN DO;
00029 4 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
LOG2A

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

None

None

Value of number in A

\[ \log_2(A) = B + \frac{C}{256} \]
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).

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, D, H&amp;L</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>MAC8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 bytes</td>
<td>Bill Baker</td>
</tr>
<tr>
<td></td>
<td>Bell Telephone Labs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
<th>Room 1G625</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Holmdel, N.J. 07733</td>
</tr>
</tbody>
</table>
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:

\[ A = 2^B \times (1 + \frac{I}{256}) \]

where B and I are integers and

\[ 0 \leq B \leq 7 \quad \text{and} \quad 0 \leq I < 255 \]

The expression for \( \text{LOG2}(A) \) in terms of B and I is:

\[ \text{LOG2}(A) = B + \text{LOG2} \left(1 + \frac{I}{256}\right) \]

where B is the characteristic and \( \text{LOG2} \left(1 + \frac{I}{256}\right) \) 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

\[ D(C_0) = 256 \cdot \text{LOG2} \left(1 + \frac{16 \cdot C_0}{256}\right) \]

for \( C_0 \) from 0 to 15 are stored in the program. \( C_0 \) is used to look up the first approximation of

\[ 256 \cdot \text{LOG2} \left(1 + \frac{I}{256}\right) \]

\( C_1 \) 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%.
ORG 44000H

LLL EQU 0610H

; LOG2(A) RESULT IN B, C

L01:

MVI B, 8
ANA A
;SET CARRY TO 0

LOD E60FH
ANI 0170H
ADII C631H
MOV L, A

MVI H, 0110H
; H VALUE OF D TABLE

MVI 7EH
MOV A, M

MVI 2DH
DCR L

MVI 96H
SUB M
;A=D(C0+1)-D(C0)

MVI 57H
MOV D, A
;D=D(C0+1)-D(C0)

MVI 6EH
MOVL, M
;L=D(C0)

MVI 79H
MOVR, C

ANI 3600H
;A=16*C0; CY=0

MVI 2603H
MVI C0, 3
;3 SINCE LSB OF C0 IS 0

L02:

RAR

JNC L03

ADD D

ANA A

L03:

DCR H

JNZ L02

RRC
;A=C0*[D(C0+1)-D(C0)]/2

RRC

RRC

RRC

ANI 0370H
;A=C0*[D(C0+1)-D(C0)]/16

ADD L

MOV C, A

DATA0:

DB 000H, 022H, 044H, 064H

DB 083H, 101H, 118H, 134H
0938  96A5E301  DB   150,165,179,193
093C  CFDE8F4  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   45000
0940  3E00  MVI   A,0
0942  D308  OUT   100
0944  3E08  MVI   A,3
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  FE08  CPI   3500
0963  C26A09 JNZ   ERROR
0966  3EFF  MVI   A,377Q
0968  D308  OUT   10Q
096A  76  ERROR: HLT
0000  END
DIGITAL TO ANALOG CONVERSION FOR EIGHT OUTPUTS

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.

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.

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

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

<table>
<thead>
<tr>
<th>low order value #1</th>
<th>high order value #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>low order remainder #1</td>
<td>high order remainder #1</td>
</tr>
<tr>
<td>low order value #2</td>
<td>high order remainder #2</td>
</tr>
</tbody>
</table>

Eight analog output voltages are produced at the filter outputs.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, D, E, H, L</td>
<td>8080 MACRO ASSEMBLER, VER. 2.2</td>
</tr>
<tr>
<td>SP is used and restored.</td>
<td>Programmer:</td>
</tr>
<tr>
<td></td>
<td>W. M. Hawkins</td>
</tr>
<tr>
<td></td>
<td>Company:</td>
</tr>
<tr>
<td></td>
<td>Hercules Inc., Res. Ctr.</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>34 bytes</td>
<td>Wilmington, De, 19899</td>
</tr>
<tr>
<td>ROM Required:</td>
<td></td>
</tr>
<tr>
<td>32 bytes</td>
<td></td>
</tr>
</tbody>
</table>
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 FFFF produces full output. If the value is 8000, 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 demand on the filter becomes greater as the value moves away from 8000. 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 ; THE 8212 LATCH ADDRESS
0001 DAC EQU 1

0000 F3
0001 210000
0002 7C
0003 19
0004 220010
0005 0608
0006 310210

DAC8: DI ; STOP USING THE SP
LXI H, 0000H ; ZERO H AND L
MOY A, H ; ZERO A
DAD SP ; MOVE SP TO HL
SHLD SAVSP ; SAVE THE POINTER
MVI B, 8 ; PROCESS 8 VALUES
LXI SP, TABLE ; POINT TO DATA

000E D1
000F E1
0010 19
0011 1F
0012 E5
0013 31
0014 31
0015 05
0016 C20E00

LOOP: POP D ; LOAD DE WITH A VALUE
POP H ; LLOAD HL WITH REMAINDER
DAD D ; ADD VALUE TO REMAINDER
RAR ; PUT C BIT INTO A
PUSH H ; NEW REMAINDER
INX SP ; PASS REMAINDER
INX SP ; (SAVE 8 CYCLES)
DCR B ; COUNT PASSES
JNZ LOOP ; GO UNTIL DONE

0019 D301
0018 2A0010
0019 F9
001F F8
0020 76

OUT DAC ; SEND BITS TO LATCH
LHLD SAVSP ; GET POINTER TO HL
SPHL ; RESTORE SP
EI ; ENABLE, IF REQUIRED
HLT ; FOR TESTING ONLY *

0000 ORG 1000H ; OR WHEREVER RAM IS AT
1000 0000 SAVSP: DW 0 ; PLACE TO PUT SP
1002 TABLE: DS 32 ; 8 VALUE, REMAINDER PAIRS
END
Binary to HEX Routine

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

TTY on port 0 and 1
HSPTTR on port 1 and 3

SEE WRITEUP

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Programmer: Sigmund Hjerde</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company: National Institute of Technology Akerson, 24G</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address: OSLO 1, Norway</td>
</tr>
</tbody>
</table>
PROGRAM TITLE: BINARY to HEX Routine.

The program perform loading of paper-tape in binary format (EBCDIC) from HSPTTR (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 exchange 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 accumulator 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 exchange the content of location 1000 with 1001, 1002 with 1003 ....... and so on.

Required hardware:

TTY on port 0 and 1
HSPTTR 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 HS PTR
1E03 0602 MVI B,02 ; CHARACTER LOOP COUNTER
1E05 1E04 MVI E,04 ; BIT LOOP COUNTER
1E07 CDA03E 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
1E18 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 2031E 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 28 DXC 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 ; SUBROUTINE FOR CHECK SIZE IN MEM
1E32 E5 TEST: PUSH H
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
**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

---

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>None</th>
<th>Programmed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>None</td>
<td>Niels S. Gundestrup</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>5210 bytes</td>
<td>Company: Geophysical Isotope Lab.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>N.A.</td>
<td>Address: Haraldsgade 6</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>MAC-80 Vers 3.0</td>
<td>City: DK-2200 Cph N</td>
</tr>
<tr>
<td></td>
<td></td>
<td>State: DENMARK</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
; 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
00 01F0D8 LXI B,-10000
0408 CD2400 CALL DECN0       ; GET MSD
000B 0118FC LXI B,-1000
000E CD2400 CALL DECN0
0011 019CFF LXI B,-100
0014 CD2400 CALL DECN0
0017 01F6FF LXI B,-10
001A CD2400 CALL DECN0
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  DECN0:  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 DECN0+2   ; RESULT NEGATIVE?
002D 3D   DCR A       ; YES, RESTORE DIGIT COUNT
002E 6B   MOV L,E       ; BINARY NUMBER
002F 62   MOV H,D
...30 D1 POP D       ; AND ADDRESS
0031 12  STAX D       ; STORE DIGIT
0032 13  INX D       ; INCREMENT POINTER
0033 C9 RET
0000 END
Hex to Decimal Conversion

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

TTY
H.S. Reader
Intellec 8

Program Object Tape

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

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
BCD Input and Direct Conversion to Binary Routine

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.

An Input Port for BCD parameter.

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

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

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, including A-Register</td>
<td>Intellec 8 Macro Assembler</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>M.H. Gansler</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>18 Bytes</td>
<td>IGM/NTI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P.O.Box 3950</td>
</tr>
<tr>
<td></td>
<td>Bellingham, WA 98225</td>
</tr>
</tbody>
</table>
; 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
0002 4F
0003 E60F
0005 5F
0006 79
0007 E6F0
0008 0F
000A 0F
000C 4F
000D 0F
000E 81
000F 07
0010 83
0011 E9

BCDBN: IN 4 ; READ BCD VALUE
MOV C,A ; STORE IN C.REGISTER
ANI 0FH ; MASK FOR UNIT'S DIGIT
MOV E,A ; STORE IN E.REGISTER
MOV A,C ; GET BCD VALUE BACK
ANI 0F0H ; MASK FOR TEN'S DIGIT
RRC ; ROTATE TWICE RIGHT
RRC
MOV C,A ; RETAIN IN C.REGISTER
RRC ; ROTATE TWICE RIGHT AGAIN
ADD C ; ADD C.REGISTER CONTENTS
RLC ; ROTATE ONE LEFT
ADD E ; ADD UNIT'S DIGIT
PCHL ; AND RETURN WITH BINARY
;
;
END

0000
HEX TO/FROM BCD

Converts hexadecimal numbers input on TTY to decimal numbers and vice versa. Decimal numbers must be ended with D, hexa-
decimals 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.

Intellec 8/MOD 80, TTY

Intellec 8/MOD 80 Monitor, vers. 2.0

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 inter-
preted as error.

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>8080 Macro Assembler, Vers. 3.0</td>
</tr>
</tbody>
</table>

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
FIXED AND FLOATING POINT ARITHMETIC Routines - MATH

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.

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)

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

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.

Program offered on diskette only.

Registers Modified:
See comments on routines

RAM Required: Dynamic allocation, based on stack pointer. Most usage requires less than 100 bytes.

ROM Required: 2000-260E HEX assembled plus monitor console 1/0

Maximum Subroutine Nesting Level:
No intrinsic limit. See comments.

Assembler/Compiler Used:
Intel V3.0 modified

Programmer:
Charles B. Falconer

Company: Yale University

Address: Room 6016 CB, Yale-New Haven Hospital, 789 Howard Ave, New Haven 06504
FLOATING POINT ELEMENTARY FUNCTION PACKAGE

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

NONE

FLOATING POINT PACKAGE, INCLUDING FIX AND FLT ROUTINES - BC1

FURNISHED IN FLOATING POINT ACCUMULATOR -
SEE ATTACHED DOCUMENTATION

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.
          165-79626

Maximum Subroutine Nesting Level:
6 (incl. misc. uses of stack) plus

Address: 4300 E. FIFTH AVENUE
          COLUMBUS, OHIO 43216
FLOATING POINT PACKAGE WITH BCD CONVERSION ROUTINE FOR 8080

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

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

No other software required except the calling program.

For FADD, FESUB, FDIV, and FMPY, the numbers to be operated upon are loaded into C-D-E and B-H-L. For FIXX, FNEG, and FBBCD, 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).

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.)

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All registers</td>
<td>MACRO (3080)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 Locations</td>
<td>Dr. Keith J. Caserta</td>
</tr>
<tr>
<td>(9 Variables, 12 locs. for Stack)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>767 Locations</td>
<td>The Procter and Gamble Company</td>
</tr>
<tr>
<td>(525 without BCD conversion routine)</td>
<td></td>
</tr>
</tbody>
</table>

Maximum Subroutine Nesting Level: 3
(4 counting the CALL to these routines)
LEAST SQUARES QUADRATIC FITTING ROUTINE FOR 8080

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

\[ aX^2 + bX + c = Y \]

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.

Floating Point Package with BCD Conversion Routine for 8080 - BC5

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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All registers</td>
<td>Dr. Keith J. Caserta</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2359 locations (for a 256 point fit)</td>
<td>The Procter and Gamble Company</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1380 locations (920 without BCD conversion routine)</td>
<td>The Ivorydale Technical Center</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (5 counting call to FIT)</td>
<td>Cincinnati</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO (8080)</td>
<td>Ohio 45217</td>
</tr>
</tbody>
</table>
FLOATING POINT PROCEDURES

DUMMY PL/M INTERFACE PROCEDURES

NONE

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

REVISED 8/8/77

NOTE: BC7 is ordered as one program with BC9.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>PL/VER. 3.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>STOLBERG-ROHR MS E15V22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>124 byte</td>
<td>DANFOSS A/S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE PROGRAM</td>
<td>DK6430 NORDBOURG DENMARK</td>
</tr>
</tbody>
</table>

8/8/77
PLM1 VERS 3.3

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  FCCHS:  PROCEDURE ADDRESS;
00060  2  GO TO 118H;
00061  2  END FCHS;
00062  1
00063  1  /* F.P. LOAD: LOADS SPEC. 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: CONVERTS 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;
/* F.P. FIX: CONVERTS VALUE IN ACC. TO FIX FORMAT
   ( SCALE FACTOR ), RESULT IN TEMP. 2 NEST LEVELS. */

FFIX: PROCEDURE (SCALE) ADDRESS;
DECLARE SCALE BYTE;
GO TO 127H;
END FFIX;

/* OVERFLOW, RETURNS THE VALUE OF THE OVERFLOW FLAG, AND CLEAR THE FLAG. */
OVERFLOW: PROCEDURE BYTE;
GO TO 12AH;
END OVERFLOW;
PL/M - FLOATING POINT INTERFACE

INTERFACES PL/M CONVENTIONS WITH FLOATING POINT ASSEMBLER FORMAT.

NONE

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

AS PR PL/M DEFINITIONS

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler/Compiler Used:</td>
<td>INTEL</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>AS TO FL.PNT. + BYTE</td>
</tr>
<tr>
<td>Programmer:</td>
<td>STOLBERG-ROHR MS E15V22</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>292 BYTE</td>
</tr>
<tr>
<td>Company:</td>
<td>DÅNFOSS A/S</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>SEE PROGRAM</td>
</tr>
<tr>
<td>Address:</td>
<td>DK6430 NORDBORG</td>
</tr>
<tr>
<td>DENMARK</td>
<td></td>
</tr>
</tbody>
</table>
FLOATING POINT DECIMAL & HEX FORMAT CONVERSION

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 TELETYPICALA FORM (SEE EXAMPLE)

TTY
INTELLEC 8

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)

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

REVISED 8/8/77

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>STOLBERG-ROHR, MSE15V22</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>80 BYTE</td>
<td>DANFOSS A/S</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>717 BYTE</td>
<td>DK 6430 NORDBOURG</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td>5 LEVELS</td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
<tr>
<td>PL/M VER. 3.3</td>
<td>DENMARK</td>
</tr>
</tbody>
</table>
00301 1
00302 1
00303 1
7 14 1
00305 1 /* REF. NO. BC9 */
00306 1 /* PROGRAM TITLE FLOATING POINT DECIMAL HEX FORMAT CONVERSION */
00307 1 /* STOLBERG-RÖHR, EL-LAB, E15-V52, 2876, ISA1560. */
00308 1
00309 1
00310 1 /***********************************************************************/
00311 1 /*
00312 1 */
00313 1 /* FLOATING POINT PROCEDURES. A1560 */
00314 1 /*
00315 1 */
00316 1 /***********************************************************************/
00317 1
00318 1 /* INITIALIZE: MOVES A SECTION OF CODE FROM ROM MEMORY TO SCRATCHPAD
00319 1 RAM MEMORY IN PREPARATION FOR EXECUTION OF MULTIPLY AND DIVIDE
00320 1 SUBROUTINES. OVERFLOW FLAG SET TO ZERO. 0 NEST LEVEL
00321 1
00322 1 FINIT: PROCEDURE;
00323 2 GO TO 100H; /* ADDRESS IN PLY-FL.PNT. INTERFACE PROGRAM */
00324 2 END FINIT;
00325 1
00326 1 /* F.P. ADD; ADDS THE SPECIFIED OPERAND TO THE F.P. ACC., RESULT
00327 1 IN TEMP= COPY OF F.P. ACC., RETURN VALUE = ADDR. OF TEMP.
00328 1 3 NEST LEVELS.
00329 1
00330 1 FADD: PROCEDURE (OPERAND) ADDRESS;
00331 2 DECLARE OPERAND ADDRESS;
00332 2 GO TO 103H;
00333 2 END FADD;
00334 1
00335 1 /* F.P. SUBTRACT: 3 NEST LEVELS. */
00336 1
00337 1 FSUB: PROCEDURE (OPERAND) ADDRESS;
00338 2 DECLARE OPERAND ADDRESS;
00339 2 GO TO 106H;
00340 2 END FSUB;
00341 1
00342 1 /* F.P. MULTIPLE. 3 NEST LEVELS. */
00343 1
00344 1 FMUL: PROCEDURE (OPERAND) ADDRESS;
00345 2 DECLARE OPERAND ADDRESS;
00346 2 GO TO 109H;
00347 2 END FMUL;
00348 1
00349 1 /* F.P. DIVIDE. 3 NEST LEVELS. */
00350 1
00351 1 FDIV: PROCEDURE (OPERAND) ADDRESS;
00352 2 DECLARE OPERAND ADDRESS;
00353 2 GO TO 10CH;
00354 2 END FDIV;
00355 1
00356 1 /* F.P. ABSOLUTE: SETS THE SIGN OF F. P. ACC. POSITIVE.
00357 1 1 NEST LEVEL. */
00358 1
00359 1 FABS: PROCEDURE ADDRESS;
00360 2 GO TO 10FH;
00361 2 END FABS;
00362 1
00363 1 /* F.P. ZERO: SETS THE F.P.ACC. = 0. 1 NEST LEVEL */
00364 1
5-97
00061 1 FZRJ: PROCEDURE ADDRESS;
00062 2 GO TO 112H;
00063 2 END FZRJ;
00065 1 /* F.P. TEST: SETS THE CONTROL FLAGS, 1 NEST LEVEL. */
00067 1 FIST: PROCEDURE ADDRESS;
00068 2 GO TO 115H;
00069 2 END FIST;
00070 1 /* F.P. COMPLEMENT: CHANGE THE SIGN OF THE F.P.ACC. 
00071 1 1 NEST LEVEL */
00073 1 FCIS: PROCEDURE ADDRESS;
00075 2 GO TO 118H;
00076 2 END FCIS;
00077 1 /* F.P. LOAD: LOADS SPEC. OPERAND INTO F.P.ACC. 
00079 1 2 NEST LEVELS. */
00080 1 FLDO: PROCEDURE (OPERAND) ADDRESS;
00082 2 DECLARE OPERAND ADDRESS;
00083 2 GO TO 11BH;
00084 2 END FLDO;
00085 1 /* F.P. INPUT: CONVERTS CHAR.STRING (MAX. 32 DIG) IN MEM- 
00087 1 ORY, TO F.P. FORMAT IN F.P.ACC. 4 NEST LEVELS */
00088 1 FINP: PROCEDURE (STRINGBUFFER) ADDRESS;
00089 2 DECLARE STRINGBUFFER ADDRESS;
00091 2 GO TO 11EH;
00092 2 END FINP;
00093 1 /* F.P. OUTPUT: CONVERTS VALUE IN ACC. TO 13 DEC DIG. IN 
00095 1 OUTBUFFER. 4 NEST LEVELS. */
00096 1 FOOUT: PROCEDURE (OUTBUFFER) ADDRESS;
00097 2 DECLARE OUTBUFFER ADDRESS;
00099 2 GO TO 121H;
00100 2 END FOOUT;
00101 1 /* F.P. FLOAT: CONVERTS BINARY VALUE IN 5 BYTE BSTRING 
00103 1 ( 32 BIT + 3 BIT SCALE FACTOR ) TO F.P. FORMAT IN ACC. 
00104 1 3 NEST LEVELS. */
00105 1 FFLT: PROCEDURE (BINSTRING) ADDRESS;
00107 2 DECLARE BINSTRING ADDRESS;
00108 2 GO TO 124H;
00109 2 END FFLT;
00110 1 /* F.P. FIX: CONVERTS VALUE IN ACC. TO FIX FORMAT 
00112 1 ( SCALE FACTOR ), RESULT IN TEMP. 2 NEST LEVELS. */
00113 1 FFIX: PROCEDURE (SCALE) ADDRESS;
00115 2 /* DECLARE SCALE BYTE; 
00116 2 GO TO 127H; 
00117 2 END FFIX; 
00119 1 /* OVERFLOW, RETURNS THE VALUE OF THE OVERFLOW FLAG, 
00120 1 AND CLEAR THE FLAG. */
00121 1 JVERFLOW: PROCEDURE BYTE;
00122 2 GO TO 12AH;
00123 0 '4 2 END JVERFLOW;
00124 1
00125 1
00126 1
00127 1 ITTYOUT: PROCEDURE (X); /* ROUTINES IN VER 2.1 MONITOR */
00128 2 DECLARE X BYTE;
00129 2 GO TO 3809H;
00130 2 END ITTYOUT;
00131 1
00132 1
00133 1 ITTYIN: PROCEDURE BYTE;
00134 2 GO TO 3F44H;
00135 2 END ITTYIN;
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 ITTYOUT(X);
00150 0 30 2 IF Y>9 THEN Y=Y+'A'-10; ELSE
00151 2 Y=Y+'0';
00152 2 CALL ITTYOUT(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 ITTYOUT(' '); END;
00166 1 DO J=0 TO 4; CALL ITTYOUT(HEAD(J)); END;
00167 1 DO J=1 TO 20;CALL ITTYOUT(' '); END;
00168 1 DO J=5 TO LAST(HEAD); CALL ITTYOUT(HEAD(J)); END;
00169 1 CALL CRLF;
00170 1
00171 1 /*
00172 1 MAIN PROGRAM
00173 1 */
00174 1
00175 1 0F '75 1 DO WHILE 1;
00176 04 '76 1 J,X=0;
00177 2 CALL ITTYOUT (3FH); CALL ITTYOUT(' '); 2 CALL ITTYOUT(3FH);
00178 2 /*
00179 2 */
00180 2 KEYBOARD INPUT ROUTINE
*/
00181 2 DC \*WHILE X<>4FH;
00182 2 X,INBUF(J)=TTYIN=30H;
00183 3 IF J=26 THEN DO; X,INBUF(J)=4FH; END; ELSE
00184 3 J=J+1;
00185 3 END;
00186 2 X=J+1;
00187 2 Y=FINP(.INBUF);
00188 2 DO J=1 TO 4; INBUF(J)=ACC(J); END;
00189 2 X=FOUT(.OUTBUF);
00190 2 DO J=X TO 27; CALL TTYOUT( ' '); END;
00191 2 DO J=0 TO 12; CALL TTYOUT(OUTBUF(J)+'0'); END;
00192 2 CALL TTYOUT( ' '); CALL TTYOUT( ' =');
00193 2 DO J=1 TO 4;
00194 2 CALL TTYOUT( ' '); CALL TTYOUT( ' '); CALL TTYOUT( ' ') CALL TTYOUT( ' ');
00195 2 CALL HEX(INBUF(J));
00196 3 CALL TTYOUT( ' ');
00197 3 CALL TTYOUT( ' ');
00198 3 CALL TTYOUT( ' ');
00199 3 END;
00200 2 CALL CRLF;
00201 2 END;
00202 1 EOF
NO PROGRAM ERRORS
BINARY MULTIPLICATION AND LEADING ZERO BLANKING - N-BYTE

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

TTY ON PORT 0 AND 1.

QBMM1E - BINARY MULTIPLICATION,
QLZ11E - REMOVE LEADING ZEROS SUBROUTINE.

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

Registers Modified: ALL
RAM Required: ALL 3 PROGRAMS STACK = 16 BYTES,
PROGRAM = 219 BYTES.
ROM Required: NONE
Maximum Subroutine Nesting Level: TWO

Assembler/Compiler Used: INTELLEG 8/MOD 80 V3.0
Programmer: CHARLES SOOLEY
Company: ENVIRONMENT CANADA
Address: 4905 DUFFERIN STREET, DOWNSVIEW ONTARIO, CANADA.
; REF. NO. B620
; PROGRAM TITLE N-BYTE BINARY MULTIPLICATION & LEADING ZERO BLANK
; ****************************************
; QBMO1S
; ****************************************
; 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          QBMO1S 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 ADDR
; LHLD ADDR   ; 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

5-103
ENDM
ORG 0100H

0100  MULTI: DS 2
0102  MULTC: DS 2
0104  MULTR: DS 2
0106  BEGIN: INX SP
0107  INX SP
0108  E1  POP H
0109  2A0001 SHLD 0100H
010C  E1  POP H
010D  2A0201 SHLD 0102H
0110  E1  POP H
0111  2A0401 SHLD 0104H
0114  3B  DCS SP
0115  3B  DCS SP
0116  3B  DCS SP
0117  3B  DCS SP
0118  3B  DCS SP
0119  3B  DCS SP
011A  3B  DCS SP
011B  3B  DCS SP
011E  OLZ11 EQU 0700H

67 C 2A0001 ZERO: LHLD MULTP ; NUMBER OF BYTES OF THE
011F  7E  MOV A,M ; MULTIPLIER IS ADDED
0120  2A0201 LHLD MULTC ; TO THE NUMBER OF BYTES
0123  3E  ADD M
0124  4F  MOV C,A
0125  2A0401 LHLD MULTR ; OF THE MULTIPLICAND
0128  77  MOV M,A ; CLEAR ACCUMULATOR
0129  AF  XRA A
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  00 86 ; 8 BITS PER BYTE COUNTER
0136  C5  LOOP: PUSH B
0137  F5  PUSH PSW
0138  AF  XRA A ; RESETS CARRY
0139  2A0001 SHIFR MULTP ; NUMBER OF BYTES
013C  7E  LHLD 00100H ; LOCATION OF BYTE TO SHIFT
013D  4F  MOV A,M ; WORD TO SHIFT
013E  7E  RAR
0140  1F  ; SHIFT RIGHT ONE BIT
0141 77   +    MOV M,A       ;RETURN WORD TO MEMORY
0142 0D   +    DCR C       ;DECREASE WORD COUNT
0143 C3E01 +    JNZ ROTATE ;CONTINUE UNTIL ZERO

0146 D24C01 CALL ADDRE  ;CALL ADDR TO ADD MULTC TO MULTR
+PARTB: SHIFR MULTR   ;SHIFT RESULT RIGHT
014C 2A0401 +    LHLD 00104H ;NUMBER OF BYTES
014F 7E   +    MOV A,M     ;LOCATION OF BYTE TO SHIFT
0150 4F   +    MOV C,A     ;WORD TO SHIFT
0151 23 +ROATE: INX H    ;LOCATION OF BYTE TO SHIFT
0152 7E   +    MOV A,M     ;CONTINUE FOR ONE BYTE
0153 1F   +    RAR         ;CONTINUE FOR ONE BYTE
0154 77   +    MOV M,A     ;CONTINUE FOR ONE BYTE
0155 0D   +    DCR C       ;RETURN WORD TO MEMORY
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
~5F 05   DCR B         ;LOOP UNTIL TOTAL
.60 0E08  MVI C,8      ;BYTES ARE DONE
0162 C23601 JNZ LOOP    ;BYTES ARE DONE
0165 2A0401 LHLD MULTR ;ADD THE MULTC TO THE RESULT
0168 CD0007 CALL QLZ11  ;ADD THE MULTC TO THE 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
0175 23   EMUC: INX H  ;BYTE OF MULTC AND MULTR
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   ADDC 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

5-105
**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**

0700 0600  **MVI B,0** ;CLEAR REGISTER B
0702 E5  **PUSH H**
0703 4E  **MOV C,M** ;NUMBER OF BYTES IN RESULT
   704 23  **CHECK: INX H**
   005 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
   0716 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
   25 C22007  **JNZ MOVE** ;WORDS ARE MOVED
   0728 C9  **RET** ;PACK FINISHED.
0000
DECIMAL MULTIPLICATION SUBROUTINE - DMULT

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.)

As required by supporting software

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

Multiplicand to be stored 1F00(LSD) to 1F06(MSD).
Multiplier to be stored 1F0A(LSD) to 1F0C(MSD).
Product delivered to 1F20(LSD) to 1F29(MSD).

Total storage used by routine, including input and output, is from 1F00 to 1F2C inclusive.

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: INTELLE 8 MACRO ASSEMBLER

State: England
BCD Multiplication
Multiplies an up to six digit bcd number by a
4 digit bcd number providing a ten digit bcd result.
All numbers are unsigned.

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

None

<table>
<thead>
<tr>
<th>RAM Contents</th>
<th>Symbol</th>
<th>#bytes</th>
<th>Data format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplicand</td>
<td>HOLD</td>
<td>6</td>
<td>LS-Digit...MS-Digit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>One digit per byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Right justified</td>
</tr>
<tr>
<td>Multiplier</td>
<td>KONST</td>
<td>2</td>
<td>MS-Digit...LS-Digit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Two digits per byte</td>
</tr>
</tbody>
</table>

Register contents - don't care

Product is stored in RAM 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
Program- 225 bytes
Programmer: Daniel S. Coolidge
Company: Barnes Engineerin Co.
80 Commerce Bd - 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 O48H. 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
0FRA   TEMP    EQU      4010D
0FB0   SUM     EQU      4016D
0000   97      SUB      A    ; CLEAR A AND CARRY
0001   32B00F  STA      SUM
0004   32B10F  STA      SUM+1
0007   32B20F  STA      SUM+2
700A   32B30F  STA      SUM+3
D00D   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 MULTIPLR
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   01A00F  LXI      B, TEMP  ; PREPARE FOR SUBROUTINE
002E   21B00F  LXI      H, SUM
0031   1E00   MVI      E, 0
0033   CDC206  CALL     KONST  ; CREATE 'SUM'
0036   3AA00F  LDA      KONST  ; GET NEXT DIGIT OF MULTIPLICAND
0039   E60F   ANI      0FH    ; MASK OUT LOW ORDER DIGIT
003B   67     MOV      H, A
003C   CDA700  CALL     PART   ; CREATE NEXT PARTIAL PRODUCT
003F   01A00F  LXI      B, TEMP
0042   21B10F  LXI      H, SUM+1
9045   1E00   MVI      E, 0
047    CDC200  CALL     KONST
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 21820F 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 21830F LXI H, SUM+3
006F 1E00 MVI E, 0
0071 CDC200 CALL CONST
0074 01800F 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   LDA X   B
007C 57   MOV D, A
007D E60F ANI 0FH   ; PREPARE 1ST DIGIT OF PRODUCT
007F 02   $TAX B   ; REPLACE SUM DIGIT W/PRODUCT DIGIT
0080 7A   LOOP 10V 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 ACCUMULATOR
0089 03   INX B
008A 0A   LDA X   B
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 D29800 JNC MORE   ; IF CARRY RESULT, STORE IN E REG
0099 1E10 MVI E, 10H
009B 57   MORE: MOV D, A   ; STORE RESULT IN D
009C E60F ANI 0FH   ; MASK LOW ORDER DIGIT
009E 02   $TAX B   ; STORE, REPLACING 'SUM' DIGIT
009F 7C   MOV A, H   ; GET LOOP COUNTER
00A0 D001 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 LXI D, HOLD ; MULTIPLICAND ADDRESS IN D
00AA 01A40F 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 9F CONST: SUB A
00C3 1E00 MVI E, 0
00C5 1E06 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 2E50 MVI H, MLTBL
00ED 7E MOV A, M
00EE C9 RET
00F0 END
BINARY MUL/DIV MULTI-PRECISION PACK FOR 8080

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.

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.

A test program may be used.

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

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 ANUL: 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
0513 09 DAD B ; NEXT HIGHER ORDER DIGIT IS 1
0514 C32205 JMP ACOUNT ; ADD MULTIPLICAND TO PARTIAL PRODUCT IF
0517 E601 ANEXT: ANI 1 ; MULTIPLIER DIGIT IS 1 AND NEXT HIGHER 0
051A CA2205 JZ ACOUNT ; DIGIT IS 0
051D 7D MOV A, L ; SUBTRACT MULTIPLICAND FROM PARTIAL PROD
051E 91 SUB C ; IF MULTIPLIER DIGIT IS 0 AND NEXT HIGHER
051F 6F MOV L, A ; DIGIT IS 1
0520 7C MOV A, H
0521 98 SBB B
0522 67 MOV H, A
0525 E3 ACOUNT: XTHL ; CHANGE CONTENTS OF HL WITH ADDRESS TO L
0528 35 DCR M ; DECREASE LOOP COUNTER IN STACK
052A 43 XTHL
052C 9A5 JZ AEND ; JMP OUT OF LOOP IF LOOP COUNTER IS 0
052E 7C MOV A, H ; ARITHMETIC SHIFT RIGHT OF PARTIAL PRODU
052F FE80 CPI 128 ; AND MULTIPLIER IN HLDE
0532 3F CMC
0535 1F RAR
0536 2D MOV H, A
0537 57 MOV A, L
0538 6F MOV L, A
0539 7A MOV A, D
053A 1F RAR
053B 57 MOV D, A
053C 7B MOV A, E
053E 1F RAR

5-117
0536 5F  MOV  E,A
0537 C30B05  JMP  ALOOP
053A 3EFE  AEND:  MVI  A.254
053C 80  ANA  E
053D 5F  MOV  E,A
053E 33  INX  SP
053F 33  INX  SP
0540 C9  RET
0541 014007  ADIV:  LXI  B.1856
0544 C5  PUSH  B
0545 E3  XTHL
0546 360F  MVI  M.15
0547 7C  AAGAIN:  MOV  A.H
0548 AA  XRA  D
0549 37  STC
054C F25005  JP  AALT1
054F 3F  CMC
0550 79  AALT1:  MOV  A,C
0551 17  RAL
0552 4F  MOV  C.A
0553 78  MOV  A.B
0554 17  RAL
0555 47  MOV  E.A
0556 29  DAD  H
0557 29  DAD  H
0558 F25F05  JP  AALT2
055B 19  DAD  D
055C C3E505  JMP  ACONT
055F 7D  AALT2:  MOV  A.L
0560 93  SUB  E
0561 6F  MOV  L.A
0562 7C  MOV  A.H
0563 9A  SBB  D
0564 67  MOV  H.A
0565 E3  ACONT:  XTHL
0566 35  DCR  M
0567 E3  XTHL
0568 C24905  JNZ  AAGAIN
056C 79  MOV  A.C
056D 17  RAL
056E 4F  MOV  C.A
056F 78  MOV  A.B
0570 17  RAL
MOV B, A
MVI A, 128
ADD B
MOV B, A
INX SP ; MAKE STACK POINTER POINT TO RETURN ADDR
INX SP
RET SP ; RETURN TO MAIN PROGRAM
RET
DOUBLE PRECISION MULTIPLY - DMPY

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.

NONE

COMPILE WITH PL/M

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

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>PL/M</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>108 BYTES</td>
<td>WAYNE A. MILLER</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>193 BYTES</td>
<td>CONTEC CONTROLS</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>2 LEVELS</td>
<td>1485 DAVIS RD, ELGIN IL 60120</td>
</tr>
</tbody>
</table>
/* PROGRAM NO. BB22 */
/* PROGRAM TITLE DMPY DOUBLE PRECISION MULTIPLY */

DMPY: PROCEDURE(A,HI,LO) ADDRESS; /*DOUBLE PREC. MULTIPLY*/
DECLARE (A,RES,P) ADDRESS; /*RTN 2 HI-ORD BYTES OF PRODUCT*/
DECLARE (HI,LO)BYTE;
P=LOW(A)*LO ; /* GET LOWEST 2 BYTES OF PRODUCT*/
P=HIGH(P) + HIGH(A)*LO + LOW(A)*HI ; /*=MID 2 BYTES OF PRODUCT */
IF CARRY THEN RES=100H; ELSE RES=0;
IF LOW(P)>7FH THEN RES=RES+1; /*ROUND*/
RETURN HIGH(A)*HI +RES+HIGH(P);
END DMPY;

/* TEST PROGRAM STARTS HERE */
DECLARE (A,B,X,Y,Z,R,S,T,U) ADDRESS, DB DATA (12136);
A=49152; /*.=75 AS LEFT JUSTIFIED FRACTION*/
Z=DMPY(A, DB, DB(1)); /*Z=9102, I HOPE*/
/* TRY WORST CASES-WHERE 3 OR 4 ARG BYTES HAVE MSB SET, */
/* WHICH PRODUCES CARRY IN STATEMENT 6 */
A=61183; /* =EEFF HEX */
B=65518; /* =FEEE HEX */
X=DMPY(A, HIGH(B), LOW(B)); /*X SHOULD=(61183*65518)/2**16 */
Y=DMPY(B, HIGH(A), LOW(A)); /*Y =61166 */
A=65518;
R=DMPY(A, HIGH(B), LOW(B)); /*R=65518**/2/2**16 =65500*/
A=60928; /* =EE00 HEX */
S=DMPY(A, HIGH(B), LOW(B)); /*S=(60928*65518)/2**16 =60911*/
B=61183; /* =EEFF HEX */
T=DMPY(B, HIGH(A), LOW(A)); /*T=(61183*60928)/2**16 =56881*/
A=33333;
U=DMPY(A, HIGH(B), LOW(B)); /*U=31118.97=31119 ROUNDED*/

EDF

NO PROGRAM ERRORS
16 BIT SQUARE ROOT ROUTINE

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.

none

16 bit argument in HL

16 bit result in HL

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D E H L</td>
<td>8080 Macro, version 2.3</td>
</tr>
</tbody>
</table>

| 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) |
8080 MACRO ASSEMBLER, V1.0

; 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 SORT: 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
 .0A0A 7C MOV A, H
 .0A08 65 MOV H, L
 .0A0C 6A MOV L, D
 .0A0D 29 DAD H ; DOUBLE LEFT SHIFT OF
 .0A0E 17 RAL ; REGISTER STRING DEHL
 .0A0F 29 DAD H ; (CONTAINING CURRENT REMAINDER).
 .0A10 17 RAL
 .0A11 EB XCHG
 .0A12 29 DAD H
 .0A13 29 DAD H
 .0A14 5A MOV E, D
 .0A15 57 MOV D, A

 .0A16 F1 POP PSW ; RETRIEVE LOOP COUNT.
 .0A17 E5 PUSH H ; SAVE LOW ORDER 16.
 .0A18 60 MOV H, B ; MOVE PARTIAL RESULT
 .0A19 69 MOV L, C ; TO HL.
 .0A1A 29 DAD H ; SHIFT LEFT, SHIFTING IN A ONE.
 .0A1B 2C INR L
 .0A1C 44 MOV B, H ; SAVE SHIFTED PARTIAL RESULT.
 .0A1D 4D MOV C, L
 .0A1E 29 DAD H ; SHIFT COPY OF PARTIAL RESULT AGAIN.
 .0A1F 2C INR L
 .0A20 19 DAD D ; SUBTRACT SHIFTED PARTIAL RESULT
 .0A21 D22600 JNC NOGO ; FROM HIGH 16 OF CURRENT REMAINDER
 .0A24 0D DCR C ; (BY ADDING COMPLEMENT).

; TEST SUBTRACTION RESULT.
; TACK A ZERO ONTO COMPLEMENTED
0025 EB               XCHG
0026 3D               DCR    A
0027 CA 2F00           JZ    DONE
0028 E1               POP    H
0029 F5               PUSH   PSW
002AC 009000          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
0033 79               MOV    A, C
0034 2F               CMA
0035 6F               MOV    L, A
0036 C9               RET
00

END

; PARTIAL RESULT.
; REPLACE HIGH ORDER 16 OF CURRENT
; REMAINDER WITH SUBTRACTION RESULT.
; DECREMENT AND TEST LOOP COUNT.
; SAVE LOOP COUNT.
; COMPLEMENT RESULT AND STORE IN HL.

; **OPTION 2:
; TO RETURN RESULT IN TWO’S
; COMPLEMENT FROM IN BC,
; SUBSTITUTE THE FOLLOWING:
; DONE: POP    H
; INX    B
; RET
Floating Point Square Root

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.

None

Floating Point Math Package - BC1

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

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All registers affected</td>
<td>Intellec &amp; Macro Assembler</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>13 bytes</td>
<td>Bob Eichenlaub</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>108 bytes</td>
<td>John Fluke Mfg. Co., Inc.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>4</td>
<td>P.O. Box 43210</td>
</tr>
<tr>
<td></td>
<td>Mountlake Terrace, Wa 98043</td>
</tr>
</tbody>
</table>
FLOATING POINT SQUARE ROOT

A FLOATING POINT NUMBER CAN ALWAYS BE WRITTEN IN THE FORM

\[ X = M \times 2^{**2K} \quad \text{IF THE EXPONENT IS EVEN OR} \]
\[ M \times 2 \times 2^{**2K} \quad \text{IF THE EXPONENT IS ODD} \]

WHERE \(0.5 < \text{ABS}(M) < 1.0\) AND \(K = 0,+1,-1,+2,-2, \ldots\)

THUS THE SQUARE ROOT OF \(X\) IS GIVEN BY

\[ (1) \quad \text{SQRT}(X) = \text{SQRT}(M) \times 2^{**K} \quad \text{FOR EVEN EXPONENT OR} \]
\[ = \text{SQRT}(M) \times \text{SQRT}(2) \times 2^{**K} \quad \text{FOR ODD EXPONENT} \]

TO CALCULATE THE SQUARE ROOT, HERON'S ITERATIVE PROCESS IS USED. THIS ENTAILS MAKING AN INITIAL APPROXIMATION, DENOTED \(Y_0\), TO \(\text{SQRT}(M)\) AND COMPUTING THE ITERATIVE VALUES:

\[ Y(i+1) = (Y(i) + M/Y(i))/2 \quad , \quad i=0,1,2, \ldots \]

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

\[ \text{ABS}(Y(n) - \text{SQRT}(M)) < \text{EPSILON} \]

FOR THE FLOATING POINT PACKAGE, \(\text{EPSILON} = 2^{-24}\) OR ABOUT \(6.2 \times 10^{-7}\), 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

\[ Y = \text{SQRT}(X) \]

THE INITIAL APPROXIMATION \(Y_0\) IS COMPUTED AS

\[ Y_0 = 0.41730759 + 0.59016206 \times M \text{ ON THE INTERVAL } [0.5,1.0]. \]

TWO ITERATIONS ARE THEN REQUIRED TO OBTAIN \(\text{SQRT}(M)\). THEN \(\text{SQRT}(X)\) 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 INTELEC 8 / MOD 80 SYSTEM.
; REF. NO. BC10
; PROGRAM TITLE FLOATING POINT SQUARE ROOT
;waresasasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaasaa
SQUARE ROOT SUBROUTINE

ON ENTRY: F.P. ACCUMULATOR HAS SORT ARGUMENT

ON EXIT: F.P. ACCUMULATOR HAS THE SQUARE ROOT

ALL REGISTERS ARE AFFECTED

FSORT:

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. IARADD
1434 CD7D03 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 SORT OF M

143D CD5F14 CALL FSQ1 ;FIRST ITERATION
1440 210120 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 ;UNBIAS THE EXPONENT
144C D680 SUI 80H ;DIVIDE EXponent BY 2 (SORT FNC)
144E 1F RAR ;SAVE CARRY BIT (ODD OR EVEN)
144F F5 PUSH PSW ;RESTORE BIAS TO EXPONENT
1450 C680 ADI 80H
1452 32300B STA ACCE
SUBROUTINE FSQ1

THIS ROUTINE PERFORMS ONE NEWTON ITERATION
TO THE SQUARE ROOT FUNCTION

FSQ1:

145F 210120 LXI H, T astonished
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
1476 C9 RET ; RETURN TO MAIN SORT ROUTINE

END OF SQUARE ROOT ALGORITHM

CONSTANTS NEEDED BY FSORT

1477 801714EB IAMULT: DB 080H, 017H, 014H, 0EBH ; 0.59916206
147B 7F559956 IADD: DB 07FH, 055H, 069H, 056H ; 0.41730759
147F 813504F3 SORT2: DB 081H, 035H, 004H, 0F3H ; 1.41421356

SCRATCH STORAGE

2000 ORG 2000H
2000 TEXP: DS 1
2001 IMANTISSA: DS 4
2005 TMP1: DS 4
2009 ARG: DS 4

2100 ORG 2100H
2100 RESULT: DS 4

END OF SQUARE ROOT FUNCTION PACKAGE

END
SQRGF

GENERATES 8 BIT ROOT OF 16 BIT NUMBER

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

8 BIT ROOT WRITTEN INTO A MEMORY LOCATION

Registers Modified:
A,B,C,D,E,H,L

Assembler/Compiler Used:
INTELIC 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 SQRTrf

0340 ORG 0340H
; B IS MSBYTE.
; C IS LSBYTE.

0421 0340 AF
0421H
0340 1F
0341 67
0342 5A
0343 22104
0346 6F
0347 2C
0348 2C
0349 C612
044C 79
034D 1D
034E C6203
0351 17
0352 4F
0353 78
0355 47
0356 7C
0357 17
0358 67
0359 2D
035A C2403
035D 2C
035E 2C
035F 14
0360 DA6703
0363 92
0364 C36A03
0367 14
0368 14
69 82
036A 67
036B 3A2104
0340 6F
0341 67
0342 5A
0343 22104
0346 6F
0347 2C
0348 2C
0349 C612
044C 79
034D 1D
034E C6203
0351 17
0352 4F
0353 78
0355 47
0356 7C
0357 17
0358 67
0359 2D
035A C2403
035D 2C
035E 2C
035F 14
0360 DA6703
0363 92
0364 C36A03
0367 14
0368 14
69 82
036A 67
036B 3A2104

0421 ROOT: EQU 0421H
0340 SQRTF: XRA A ; CLR A & CARRY
0341 MOV H, A
0342 MOV D, A
0343 STA ROOT
0346 MOV L, A ; SET
0347 INR L ; L
0348 INR L ; TO 2.
0349 ADI 12H ; SET COUNTER.
044C SQRT1: MOV A, C
034D DCR E ; SETS FLAGS 'CEPT CARRY.
034E JZ ENDRT ; EXIT LOOP AT 9TH PASS.
0351 RAL ; GET MSB OF LSBYTE IN CARRY.
0352 MOV C, A ; SAVE REMAINING LSBYTE.
0353 MOV A, B ; GET MSB BYTE.
0354 RAL ; GET MSB IN CARRY & LSBYTE'S MSB TO A.
0355 MOV B, A ; SAVE SHIFTED MSB BYTE.
0356 MOV A, H ;
0357 RAL ; MSB OF B TO CARRY &
0358 MOV H, A ; TO H, RIGHT JUSTIFIED.
0359 DCR L ;
035A JNZ SQRT1 ; REPEAT TIL 2 BITS ARE IN H.
035D INR L ; RESET L
035E INR L ; TO 2.
035F INR D ; GET AN INR IN BEFORE TEST.
0360 JC MINH ;
0363 SUB D ; IF PLUS REMAINDER.
0364 JMP SQRT2 ;
0367 MINH: INR D ;
0368 INR D ;
036A ADD D ; IF PLUS REMAINDER.
036B LDA ROOT ; FLAGS STILL SET
036E FA7703  JM MINH1 ; BY SUB OR ADD.
0371 B7    ORA A  ; CLR CARRY.
0372 17    RAL  ;
0373 3C    INR A ; ROOT CHARACTER
              ; IS 1.
0374 C37903 JMP SORT3 ;
0377 B7    MINH1: ORA A ; CLR CARRY
0378 17    RAL ; ROOT CHARACTER
              ; IS 0.
0379 322104 STA ROOT ; MAKE ROOM IN D
037C 17    RAL ; FOR THE NEXT 2 BIT
037D 17    RAL ; BYTE FROM ALGORITHM.
037E 57    MOV D, A ;
037F C34C03 JMP SORT1 ;
0382 C9    ENDT: RET ;
0000    END
FLOATING POINT SQRT SUBROUTINE

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

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

Location "VAR1" must be the input argument in floating point notation.

Location "XP" becomes the square root of whatever was in "VAR1".

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8008 Assembler 2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jack G. Ganssle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Neotec Corporation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2431 Linden Lane</td>
</tr>
<tr>
<td></td>
<td>Silver Spring, MD 20910</td>
</tr>
</tbody>
</table>
;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
0B48 STR EQU 0B48H
00A9 XPL EQU XP AND 0FFH
0088 CTRL EQU CTR AND 0FFH
0079 VARIL EQU VARI AND 0FFH
2620 ORG BEGIN
2^20 21A814 LXI H, CTR
. 23 360B MVI M, 11 ;# REPEATS=11
2625 2C INR L
2626 3681 MVI M, 81H ;INITIAL GUESS=ABOUT 1
2628 2C	INR	L
2629 3600	MVI	M, 0	; THATS A +1, BOYS
262B 2E79	ITER:	MVI	L, VARIL	; FIGGER (N/\times\times)-5
262D CD6E08	CALL	LOD	; ACC=N
2630 21A914	LXI	H, XP
2633 CD8408	CALL	DIV	; ACC=N/\times
2636 21A914	LXI	H, XP
2639 CDD708	CALL	AD	; ACC=N/\times+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/\times\times)
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
FLOATING POINT SQUARE ROOT ROUTINE - FAST

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

None.

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

Assumes number in floating point accumulator.

Returns square root in floating point accumulator.

<table>
<thead>
<tr>
<th>registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A11</td>
<td>Intellic 8/MOD 80 Ver 3.0</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer: W. Moritz/L. Mace</td>
</tr>
<tr>
<td>16 Bytes</td>
<td>Company: Center for Bioengineering</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: U. of Wash., Seattle, WA, 98195</td>
</tr>
<tr>
<td>161 Bytes</td>
<td>Maximum Subroutine Nesting Level:</td>
</tr>
</tbody>
</table>
; REF. NO. BC12
; PROGRAM TITLE FAST FLOATING POINT SQUARE ROOT ROUTINE

;***********************************************************************
;
; SQUARE ROOT SUBROUTINE ---SORT
; 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 BIOENGINEERING
; W. MORITZ AND L. MACE MAY 15, 1975
;
;***********************************************************************

; SUBROUTINES USED

03D7 AD EQU 03D7H ; FL. PT. ADD SUBROUTINE
03D4 SB EQU 03D4H ; FL. PT. SUBTRACT SUBROUTINE
038C MUL EQU 038CH ; FL. PT. MULTIPLY SUBROUTINE
03B4 DIV EQU 03B4H ; FL. PT. DIVIDE SUBROUTINE
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  EOU  0B49H
0B4C  SCR4  EOU  0B4CH

; SORT ROUTINE  HERON'S METHOD

0800  ORG  0800H

; LOAD ACC AND SCRATCH WITH NUMBER

0800  CD5903  SORT:  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  CD0703  CALL  AD  ; SET CONTROL BITS
080F  DA9608  JC  OFOUT  ; OUT IF OVERFLOW
0812  CA9608  J2  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  21400B  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 SORT 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 ITERATION LOAD NUMBER
0848  21440B  LXI  H,SCR2
084B  CD8A03  CALL  DIV  ; DIVIDE BY LAST GUESS
084E  21440B  LXI  H,SCR2
P951  CD7003  CALL  AD  ; ADD LAST GUESS
J4  21440B  LXI  H,SCR2
0857  CD8A03  CALL  DIV  ; DIVIDE BY 2
085A  21480B  LXI  H,SCR3
085D CD3E03 CALL STR ;STORE NEW GUESS
0860 21440E LXI H, SCR2
0863 CD0403 CALL SB ;SUBTRACT LAST FROM NEW GUESS
0866 CA6C08 JZ SROUT ;FINISHED IF FINAL ITERATION
0869 C33608 JMP LOOP ;JUMP TO NEXT ITERATION

;FIND SORT 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 E001 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 21A008 LXI H, SORT2 ;ADDR OF SORT2 CONST.
0886 CD8C03 CALL MUL ;SORT2*SORTX
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 OROUT: CALL ZRO ;ZERO OUT ACC FOR RETURN
0899 C9 RET
089A 813504F3 SORT2: DB 81H, 35H, 04H, 0F3H
089E 82000000 TW0: DB 82H, 00H, 00H, 00H
0000 END
NATURAL LOGARITHM

Computes the natural logarithm of a number between 1 and 65535

Basic 8080 System

70 Instructions
(+12 instructions for MULTIplication Subroutine)

Number to be converted in H and L

Mantissa of LN Z in A

Characteristic of LNZ in H and L, expressed in 1/65536
(Fixed point with format (A). (H) (L)

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSW, H, L</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Bytes</td>
<td>MAC 80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company: Battelle Institute</th>
</tr>
</thead>
<tbody>
<tr>
<td>128 Bytes for LN</td>
<td>B. Hauert</td>
</tr>
<tr>
<td>20 Bytes for MUL.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7 Route de Drize</td>
</tr>
<tr>
<td></td>
<td>1277 Carouge</td>
</tr>
<tr>
<td></td>
<td>Geneva, SWITZERLAND</td>
</tr>
</tbody>
</table>
FLOATING POINT LOG SUBROUTINE - COMMON LOGARITHMS

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

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.)

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

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

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>8008 Assembler 2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jack G. Ganssle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Neotec Corporation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2431 Linden Lane</td>
</tr>
<tr>
<td></td>
<td>Silver Spring, MD. 20910</td>
</tr>
</tbody>
</table>
; 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
; SO 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
; FLOATING POINT REPRESENTATION
; OF 2.302585093.

; CODE MAY 5, 1975  YFNED

0100  BEGIN  EQU  100H
09D7  ADD  EQU  8D7H ; FP ADD
(8C  MUL  EQU  88CH ; FP MULTIPLY
083E  STR  EQU  83EH ; STORE FP ACCUM
086E  LOD  EQU  86EH ; FP LOAD

5-150
08B4  DIV     EQU     8B4H ; FP DIVIDE
08B0  FLT     EQU     0800H ; 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     D, M ; A=X EXPONENT
0106  D671    SUI     71H ; MAKE 2**N BINARY POINT
0108  5F      MOV     E, A ; E=BINARY POINT
010B  AF      XRA     A
010D  4F      MOV     C, A
010F  57      MOV     D, A
0111  0601    MVl     B, 1 ; ABCD=UNJUSTIFIED 1
011C  CD0008   CALL    FLT ; ACCUM=2**N
011F  210011   LXI     H, BUF
0122  215011   LXI     H, LOG
0125  1D18    CALL    STR ; SAVE 2**N
0128  CD6808   CALL    LOD ; ACCUM=X
012B  210011   LXI     H, BUF
012D  CD8408   CALL    DIV ; ACCUM=X/2**N=M
0130  21A801   LXI     H, FP1
0133  CDD708   CALL    AD ; ACCUM=M+1
0136  210011   LXI     H, BUF
0139  CD3E08   CALL    STR ; SAVE M+1
013C  CD408    CALL    SB ; ACCUM=M-1
013F  210011   LXI     H, BUF
0142  CD408    CALL    DIV ; ACCUM=Z=(M-1)/M+1
0145  210011   LXI     H, BUF
0148  CD3E08   CALL    STR ; SAVE Z
014C  CDB580   CALL    MUL ; ACCUM=Z**2
0150  2E00    MVl     L, BUF AND 0FFH
0153  210411   LXI     H, BUF+4
0156  CD3E08   CALL    STR ; SAVE Z**2
0159  21B401   LXI     H, FP15
015C  CD8008   CALL    MUL ; (Z**2)/15
015F  21B801   LXI     H, FP23
0162  CD408    CALL    DIV ; (15+Z**2)/23
0165  21A801   LXI     H, FP1
0168  CDD708   CALL    AD ; 1+15*Z**2/23
016B  210411   LXI     H, BUF+4
016E  CD8008   CALL    MUL ; (1+15*Z**2/3)*Z**2
0171  21B001   LXI     H, FP3
0174  CD408    CALL    DIV ; T=(1+15*Z**2/3)*Z**2/3
0168 21A01 LXI H, FP1
0169 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 CD708 CALL AD ; ACCUM=LN(X)
019B 21EC01 LXI H, BASE
019E CD4008 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 85300000 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)
0008 END
Approximating Routine

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.

None

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

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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, C, D, H and L</td>
<td>MAC 8 VER 2.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>W.E. BAKER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>59 bytes</td>
<td>BELL TELEPHONE LABORATORIES 16-625</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE</td>
<td>HOLMDEL, NEW JERSEY 07733</td>
</tr>
</tbody>
</table>
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 \( F(16 \cdot N) \) for \( N = 0, 1, 2, \ldots, 16 \). For any given input, \( X \), the routine calculates an \( N \) and a \( D \) such that \( X = 16 \cdot N + D \). It then approximates the value of \( F(X) \) as

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

**ANT LOG 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 \( N = 0, 1, 2, \ldots, 16 \). 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 \( \pm 1 \).
ORG 4000H

0800 2E28 ALOG: MVI L, 053H ; POINT TO
0802 2608 AP1: MVI H, 010H ; F(1)
0804 07 AP2: RCL
0805 07 RCL
0806 07 RCL
0807 07 RCL
0808 4F MOV C, A ; C = 16 * DEL + N
0809 E60F ANI 360H
080B 57 MOV D, A ; D = 16 * DEL
080C 79 MOV A, C
080D E60F ANI 017H ; 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
081A 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 037H
0828 85 ADD L
0829 C9 RET
082A 000B1723 DB 0, 11, 23, 35
082E 303E4C5B DB 48, 62, 76, 91
0832 6A7A889C DB 106, 122, 139, 156
0836 AFC2D6EA DB 175, 194, 214, 234
083A 00 DB 0
0000 END
SIN X, COS X SUBROUTINE

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.

Machine line and configuration for cross products.

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.

Subroutine is entered with call CPS X or call SINX with X in register A. For 0° < Angle < 90°
Then 0FH < X < FFH

Subroutine is exited with the Cos of X (Sine of X) in register A. For 0 ≤ Cos ≤ 1.00

0H ≤ Cos X ≤ FFH

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All register modified</td>
<td>8080 Macro Assembler Version 2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum (256 Bytes)</td>
<td>Billy R. Slater</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>25H</td>
<td>Forney Engineering</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two</td>
<td>P.O. Box 189</td>
</tr>
<tr>
<td></td>
<td>Addison, Texas 75001</td>
</tr>
</tbody>
</table>
; 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
1001 SINF: CMA ;FORM SINF=COS(90'-X)
1004 COSX: LXI D, 0 ;CLEAR D&E
1005 CD2510 CALL MULT ;SQUARE X
1008 5C MOV E, A
1009 1600 MOV E, H
100B 05 PUS 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 MOV D, 0
1013 3E3C MOV 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
1028 0608    MVI B, 8
102A 29     LOOP: DAD H
102B 17     RAL
102C D23210  JNC DEC
102F 19     DAD D
1030 CE00    ACI 0
1032 05     DEC: DCR B
1033 C22A10  JNZ LOOP
1036 C9     RET
0000      END
INTEGRATION ROUTINE - RMSTF

To calculate the integration \( \int_a^b F(x) \, dx \) of any continuous function \( F(x) \) between two limits \( a \) and \( b \).

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

None

PL/M Compiler

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 subdivisions.

\( N \) 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.
Explanation of RMSTF

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

\[ T(L) = (1/2) (F_a + F_b) + \sum_{i} F_i / NL \]  \[ ..... (1) \]

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 \( \sum_{i=1}^{m} H_i = (b - a)/N \) 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:

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

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:

**RMSTF:**

PROCEDURE (PTR,N) ADDRESS;

DECLARE PTR ADDRESS, (F BASED PTR) (129) ADDRESS;
DECLARE (TEMP,MAX,C) ADDRESS;
DECLARE POS LITERALLY '0', NEG LITERALLY 'OFFH',
CONSTANT LITERALLY 'MAX';
DECLARE (T,D) (8) ADDRESS;
DECLARE S (8) BYTE;
DECLARE (N,NL,M,I,L,K,P,H,HIGH) BYTE;

Difference:

PROCEDURE;

S(L) = POS;
D(L) = T(L) - T(L-1);
IF CARRY THEN
   DO:
      D(L) = -D(L);
      S(L) = NEG;
   END;
   D(L) = D(L) / (CONSTANT - 1);
RETURN;

END DIFFERENCE;

H,NL = N;
M = 0;
DO WHILE NL>1;
   M = M + 1;
   NL = SHR(NL,1);
END;
MAX = 0;
DO I = 0 TO N;
   IF F(I)>MAX THEN MAX = F(I); /* 0<=F(I)<=32787 */
END;

IF MAX=0 THEN RETURN MAX;
P = P + 1; /* RESET CARRY */
DO WHILE CARRY=0;
P = P + 1;
MAX = SHR(MAX,1);
END;

/* ALL VALUES OF F(I) WILL BE SHIFTED LEFT P TIMES TO INCREASE THE ACCURACY */

TEMP = F(0) +F(N);
IF P>1 THEN TEMP = SHR(TEMP,P=1);
T(0) = TEMP; /* FIRST APPROX. USING SINGLE TRAPEZOID */
HIGH = 0;
CONSTANT = 4;
/* NOW NL = 2 */
DO L = 1 TO M; /* FIND BETTER APPROXIMATIONS T(L) */
   /* SUM UP ALL NEW INTERMEDIATE ORDINATES */
DO WHILE I<N;
    C = C + SHL(F(I),P);
    IF CARRY THEN HIGH = HIGH + 1;
    I = I + H;
END;
C = 66
H = SHR(H,1);

/* 3-BYTE SUM OF TERMS IN EQUATION (1) */
TEMP = TEMP + C;
IF CARRY THEN HIGH = HIGH + 1;
/* DIVIDE BY NL USING RIGHThifts L TIMES */
C = TEMP;
K = HIGH;
DU 1 = 1 TO L;
    K = SHR(K,1);
    C = SCR(C,1);
END;
T(L) = C;

CALL DIFFERENCE;
NL = NL + NL; /* USE TWICE AS MANY TRAPIZIOIDES FOR NEXT CALCULATION */
END;

/* START CORRECTION FOR THE EFFECT OF ALL ERRORS INTRODUCED BY UP TO THE (2M)TH POWER OF THE SMALLEST SUBDIVISION */
DU K = 1 TO M;
    CNSINT = SHR(CNSINT,2);
DU L = K TO M;
    IF S(L) THEN T(L) = T(L) - D(L);
    ELSE T(L) = T(L) + D(L);
    IF L>K THEN CALL DIFFERENCE;
END;

RETURN SHR(T(M),P);
END RMSTF; /* RMSTF INVOCATION */

DECLARE Y(17) ADDRESS INITIAL
(10000,9412,8889,8421,8000,7619,7273,6956,6667,
  6400,6154,5926,5714,5517,5333,5161,5000);
DECLARE (X1,X2) ADDRESS INITIAL (1,2); /* X2 > X1 */

AERA = (X2-X1) * RMSTF(Y,LAST(Y));
EOF
NO PROGRAM ERRORS
BINARY TO BCD CONVERSION, LEADING ZERO BLANKING - 32-BIT

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.

NO HARDWARE REQUIRED

16 BIT X 16 BIT DIVIDE ROUTINE - INCLUDED

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

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL REGISTERS MODIFIED</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP TO 10 LOCATIONS FOR RESULT</td>
<td>8080 MACRO ASSEMBLER, VER 2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>6DH -109 WORDS</td>
<td>Dr. Ilan Tal</td>
</tr>
<tr>
<td></td>
<td>Elscint Ltd.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Box 5258</td>
<td></td>
</tr>
<tr>
<td>Haifa 31-051, ISRAEL</td>
<td></td>
</tr>
</tbody>
</table>
; 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
0013 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 C3100  JMP BCD ; BCD CONVERSION

; BINARY TO BCD (ASCII 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 M, B ; STORE 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 C2100  JNZ BCD1
0028 21000  BCD2:  LXI H, 0 ; NO REMAINDER
2B 01F6FF  LXI B, NOT 9 ; 2'S COMPLEMENT 10
; E CD4000  CALL DIVIDE ; MSB/10
0031 E3   XTHL
0032 EB   XCHG
CALL  DIVIDE:LSB/10
MVI A,30H ASCII "0"
ADD E ADD REMAINDER
MOV B,A TEMP STORE IN B
POP D MSB
MOV A,L
ORA H
ORA D
ORA E ;WORD=0?
JZ BCD3
XTXL HL=ADDR OF RESULT
MOV M.B WRITE RESULT
DCX H B2C ADDR
XTXL
PUSH H ;STORE IN STACK ADDR,LSB
JMP BCD2
POP H HL=ADDR OF RESULT
MOV M.B WRITE RESULT
RET
DIVIDE: MOV A,3CH COUNTER
DIV1: INX SP ADJUST STACK POINTER FOR
XCHG SP NO UNDERFLOW
DIV2: RAR SAVE CARRY BIT
RAL RESTORE CARRY,SAVE DE OVERFLOW
JNC $+4
RM
INX H WRITE CARRY IN DE
ADI 4 INCREMENT COUNTER
XCHG
DAD H
RAR RESTORE DE OVERFLOW
JNC $+4 WRITE DE OVERFLOW IN HL
INX H WRITE DE OVERFLOW IN HL
ADD A CORRECT A SHIFT
PUSH H SAVE HL FOR UNDERFLOW
DAD B 2'S COMPLEMENT ADD
JC DIV1 NO UNDERFLOW
DAD H UNDERFLOW, RESTORE HL
JMP DIV2
END
Fixed point Chebyshev sine and cosine for PL/M users

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 213 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:

\[
\begin{align*}
\text{COS}(x) &= \text{COSINE}(x) \\
\text{SIN}(x) &= \text{COSINE}\left(\frac{\pi}{2} - x\right) \quad 0 \leq x \leq \frac{\pi}{2} \\
\text{COS}(x) &= -\text{COSINE}(\pi - x) \quad \frac{\pi}{2} < x \leq \pi \\
\text{SIN}(x) &= -\text{COSINE}\left(\frac{3\pi}{2} - x\right) \quad \pi < x \leq \frac{3\pi}{2} \\
\text{COS}(x) &= \text{COSINE}\left(2\pi - x\right) \quad \frac{3\pi}{2} < x \leq 2\pi \\
\text{SIN}(x) &= -\text{COSINE}\left(\frac{3\pi}{2} - x\right) \quad \frac{3\pi}{2} < x \leq 2\pi
\end{align*}
\]

COSINE, the heart of the package, finds the cosine of an angle between 0 and \(\frac{\pi}{2}\) radians by evaluating the Chebyshev polynomial
\[
\cos = 1 + x^2 (a_2 + x^2 (a_4 + x^2 (a_6 + x^2 (a_8 + a_{10} x^2))))
\]

\[
\begin{align*}
a_2 &= -0.49999 \quad 99963 \\
a_4 &= 0.04166 \quad 66418 \\
a_6 &= -0.00138 \quad 88397 \\
a_8 &= 0.00002 \quad 47609 \\
a_{10} &= -0.00000 \quad 02605
\end{align*}
\]

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:

\[
\begin{align*}
\text{TSTCOS}(0) &= 2D \quad 43 \\
\text{TSTCOS}(1) &= D2 \quad BE \\
\text{TSTCOS}(2) &= D2 \quad BE \\
\text{TSTCOS}(3) &= 20 \quad 02 \\
\text{TSTCOS}(0) &= 2D \quad 43 \\
\text{TSTCOS}(1) &= D2 \quad BE \\
\text{TSTCOS}(2) &= D2 \quad BE \\
\text{TSTCOS}(3) &= C8 \quad 93
\end{align*}
\]

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>PL/M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>42 words</td>
<td>Bradley G. Stewart</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>711 words</td>
<td>Aeronutronic Ford Corp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3939 Fabian Way</td>
</tr>
<tr>
<td></td>
<td>Palo Alto, CA 94303</td>
</tr>
</tbody>
</table>
DECLARE (TSTCJS,TSTSIN) (4) ADDRESS;
DECLARE VALUES(4) ADDRESS INITIAL (6433,19301,32169,42893);
DECLARE I BYTE;

/* THE MULTI ROUTINE MULTIPLIES TWO BIT NUMBERS 
AND RETURNS THE HIGH ORDER 16 BITS OF THE RESULT */

MULT: PROCEDURE (MULX,MULY) ADDRESS;
DECLARE (MULX,MULY,MULHO,MULLO) ADDRESS;
DECLARE MUL BYTE;
MULHO = MULX; MULLO = 0;
DO MULI = 1 TO 16;
   MULLO = SHL(MULLO,1);
   MULHO = SCL(MULHO,1);
   IF CARRY THEN DO;
       MULLO = MULLO + MULY;
       MULHO = MULHO PLUS 0;
   END;
END;
RETURN MULHO;
END MULT;

/* THIS ROUTINE FINDS THE COSINE OF AN ANGLE BETWEEN 0 AND PI/2 THE ARGUMENT IS SCALED BY 2**15 */

COSINE: PROCEDURE (RADS) ADDRESS;
DECLARE (RADS,RADT) ADDRESS;
RADS = MULT (RADS,RADS);
RADT = 10923 - MULT(RADS,1456 - MULT(RADS,
104 - MULT(5,RADS)));
RETURN 16384 - MULT(RADS,32768 - MULT(RADS,RADT));
END COSINE;

/* THE NEXT TWO FUNCTIONS FIND THE SINE AND COSINE OF ANGLES BETWEEN 0 AND 2*PI THE ARGUMENTS ARE IN RADIANS AND ARE SCALED BY 2**13 */

SIN: PROCEDURE (RADS) ADDRESS;
DECLARE RADS ADDRESS;
IF RADS>51471 THEN RADS = RADS - 51471;
IF RADS>18603 THEN
   RETURN -COSINE(SHL(RADS - 38603,2));
IF RADS>25735 THEN
   RETURN -COSINE(SHL(38603 - RADS,2));
IF RADS>12867 THEN
   RETURN COSINE(SHL(RADS - 12857,2));
RETURN COSINE(SHL(12867 - RADS,2));
END SIN;

CJS: PROCEDURE (RADS) ADDRESS;
DECLARE RADS ADDRESS;
IF RADS>51471 THEN RADS = RADS - 51471;
IF RADS>18603 THEN
   RETURN CJSINE(SHL(51471 - RADS,2));
IF RADS>25735 THEN
RETURN -COSINE(SHL(RADS - 25735, 2));
IF RADS>12867 THEN
RETURN -COSINE(SHL(25735 - RADS, 2));
RETURN COSINE(SHL(RADS, 2));
END COS;

/*
THE MAIN PROGRAM CALCULATES THE SINE AND COSINE OF
45, 135, 225, AND 300 DEGREES */

DO I = 0 TO 3:
ISTSIN(I) = SIN(VALUES(I));
ISTCOS(I) = COS(VALUES(I));
END;

ENDF
32 BIT DIVIDE SUBROUTINE

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.

Minimum: SDK-80

None

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

\[
\begin{align*}
((H)(L)) + 0 & \text{ MS BYTE DIVIDEND} \\
1 & \\
2 & \\
3 & \text{ LS BYTE DIVIDEND}
\end{align*}
\]

\[
\begin{align*}
((H)(L)) -6 & \text{ MS BYTE DIVIDEND} \\
-5 & \\
-4 & \\
-3 & \text{ LS BYTE DIVIDEND}
\end{align*}
\]

\[
\begin{align*}
((H)(L)) - 2 & \text{ MS BYTE DIVISOR} \\
-1 & \text{ LS BYTE DIVISOR} \\
0 & \text{ MS BYTE QUOTIENT} \\
1 & \text{ LS BYTE QUOTIENT}
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,B,C,D,E,H,L, F/F's</td>
<td>MAC80 on TYMSHARE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Routine &amp; Temp</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>202 (CA) _{16}+8+6</td>
<td>Storage &amp; Stack</td>
<td>Albert J. Gibbons</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Westinghouse Industry Systems Div.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non reentrant</td>
<td>200 Beta Drive Pittsburgh, Pa. 15238</td>
</tr>
</tbody>
</table>
8 BIT MULT AND DIV.

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

DIV : TO DIV 15 BITS BY 8

8080 PROCESSOR SYSTEM

NONE

SEE NOTES ON ATTACHED DOCUMENTATION

 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 FORM 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 D2A00 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 BIT IDEND
; 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

5-179
Double-Precision Integer Arithmetic Package

- Compute sine and cosine of angle defined by an orthogonal 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.

Intel 8080 Processor

None

See next page

See next page

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>ISIS 8080 Macro Assembler V1.0</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer: N. Anderson/G. Woodley</td>
</tr>
<tr>
<td>approximately 30 bytes</td>
<td>Company: Woodley Associates</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: 604 Indian Home Road</td>
</tr>
<tr>
<td>a total of 581 bytes</td>
<td>Danville, CA 94526</td>
</tr>
</tbody>
</table>
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 7FFF16.
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 0
and 16-bit 2's complement result in HL if division is successful. If the
division is not successful (overflow or division by 0) exit with carry
equals 1 and result equals 0.

Square Root
Enter SQRT with a 30-bit positive operand in BCDE. On exit, the 15-bit
result is in HL. Maximum result is 7FFE16.
FLOATING POINT EXTENDED MATH PACKAGE FOR 8080 - XMATH

FLOATING POINT ROUTINES
INTEGER/FRACTIONAL PART
REAL BASE TO REAL EXponent A\times X
SQUARE ROOT
TRIG SIN, COS, AND TAN
LOG BASE E
ARCSIN, ARCCOS, AND ARCTAN
EXPONENTIAL, E^X
POLYNOMIAL EXPANDER
LOG BASE 10
DEGREES \leftrightarrow RADIANS CONVERSIONS
10^X

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

SEE ATTACHED DOCUMENTATION

SEE ATTACHED DOCUMENTATION

Program offered on diskette only.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEE DOCUMENTATION</td>
<td>RESIDENT MACRO-ASSEMBLER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 BYTES</td>
<td>RICHARD C. ALLEN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1700 bytes</td>
<td>TEXAS MICROSYSTEMS, INC.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6610 HARWIN, SUITE 125</td>
</tr>
<tr>
<td></td>
<td>HOUSTON, TEXAS 77036</td>
</tr>
</tbody>
</table>
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
           FSIN - 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 DC1 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 BOUNDARIES! ***

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, ROMX, AND RAM, ARE USED IN THE FOLLOWING MANNER. SINCE THE BASIC MATH PACK WAS WRITTEN FOR THE 8080, ALL FLOATING POINT NUMBERS USED BY BOTH SYSTEMS MAY NOT CROSS PAGE BOUNDARIES. 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 ROMX 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 'ROMX' 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 BY 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.
Floating Point Package for Intel 8008 and 8080 Microprocessors

Add, subtract, multiply, divide, square root

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

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

Address of both operands
Address of result

Output to TTY routines included
Input from TTY routines included

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

Registers Modified:
All

Assembler/Compiler Used:
Intel assembler

RAM Required:
4 words per operator
4 words per result +17 word

Programmer:
M.D. Maples/H. Brand

ROM Required:
scratch

Company:
Lawrence Livermore Lab

Maximum Subroutine Nesting Level:

Address: Box 808, M/S L403
Livermore, Ca. 94556
RNGEN RANDOM NUMBER GENERATOR

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.

8080 MICROPROCESSOR

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

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

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEE ATTACHED SHEET</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEE ATTACHED SHEET</td>
<td>PL/M 80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>K. CHRISTIAN KNUDSEN</td>
</tr>
</tbody>
</table>

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/ARG1;
00014 2 LOC2=LOC1=LOC2*ARG1;
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(D)=NUM;
00026 1 GO TO LOOP; /* ENDLESS LOOP */
00027 1 EOH
00028 1 NO PROGRAM ERRORS
RANDOM NUMBER GENERATOR - 8-BIT

The program reads data from page 0, address FF and generates a random number. The new number is written back in the same location. All numbers except zero are generated. Zero is a disallowed state and is corrected in the program.

TTY

TTY Routine

The program may be called with any number in memory page 0, address FF

Program exits with the pseudo random number in register A and memory, page 0, address FF

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, H, L</td>
<td>MAC80, VER 2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Bytes</td>
<td>Joe J. Gentle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>26 Bytes</td>
<td>Collins Radio Company</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Mail Station 106-171</td>
</tr>
<tr>
<td></td>
<td>Cedar Rapids, Iowa 52402</td>
</tr>
</tbody>
</table>
; REF. NO. BB37
; PROGRAM TITLE 8 BIT PSEUDO RANDOM NUMBER GENERATOR
; 
; 8 BIT PSEUDO RANDOM NUMBER GENERATOR
; 
; THE PROGRAM READS DATA FROM ADDRESS FF
; GENERATES A RANDOM NUMBER WHICH WHICH IS WRITTEN
; IN THE SAME LOCATION. ALL NUMBERS EXCEPT ZERO
; ARE GENERATED. 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
0009 7D MOV A, L ; SET TO ALL ONES IF SO
000B 47 SKIP: MOV B, A ; PIT IT IN B
000D E61D ANI 1DH ; SAVE BITS 0, 2, 3, AND 4
000F 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
0015 0F RRC ; ROTATE IT RIGHT
0017 E67F ANI 7FH ; SET MSB TO ZERO
0019 81 ADD C ; ADD C TO IT
001B 77 MOV M, A ; STORE NEW NUMBER
001D C9 RET ; RETURN
;
0000 END

; JOE J GENTLE
; COLLINS RADIO CO
; JUNE 17, 1975

5-194
**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.

\[ X(N+1) = (2058^4X(N) + 13849) \mod 2^{**16} \]

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

**Required Hardware**

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

** Required Software**

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).

**Input Parameters**

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

**Output Results**

---

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>INTELLEC 8 MACRO ASSEMBLER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>42H for the subroutine; 0AH for the stack; 2 for random no.</td>
<td>Vito A. Trujillo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Zot Manufacturing Co.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack size greater than 0AH</td>
<td>1619 Reed St.</td>
</tr>
<tr>
<td></td>
<td>Lakewood, Colorado 80215</td>
</tr>
</tbody>
</table>
; 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  DCH     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  DCH     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
CALL SBR1
CALL SBR2
MVI D, 2
CALL SBR1
CALL SBR2
RET

; FORMS (B AND C)*2^D

SUB A
MOV A, C
RAL
MOV C, A
MOV A, B
RAL
MOV B, A
DCR D
RZ
JMP SBR1

; 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

SUB A
MOV A, M
ADD C
MOV C, A
INX H
MOV A, M
ADC B
MOV B, A
DCX H
RET

END
PL/M HISTOGRAM PROCEDURE AND RANDOM NUMBER GENERATOR

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.

TTY on ports 0 and 1

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

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)

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/M program</td>
<td>PL/M VER 2.0</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>800H locations</td>
<td>Rex Tracy</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>0</td>
<td>Colorado State University</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>stack size = 8 bytes</td>
<td>Electrical Engineering Dept.</td>
</tr>
</tbody>
</table>
DECLARE CR LITERALLY '0DH', IF LITERALLY '0AH', TRUE LITERALLY '1',
FALSE LITERALLY '0';

** PRINT VIA MONITOR **

PRINTCHAR : PROCEDURE (CHAR);
  DECLARE CHAR BYTE;
  DECLARE IOCO LITERALLY '3809H';
  GOTO IOCO;
END PRINTCHAR;

PRINTSTRING : PROCEDURE (NAME, LENGTH);
  DECLARE NAME ADDRESS,
    (LENGTH, 1) CHAR BASED NAME) BYTE;
  DO I = 0 TO LENGTH-1;
    CALL PRINTCHAR(CHAR(I));
  END;
END PRINTSTRING;

PRINTNUMBER : PROCEDURE (NUMBER, BASE, CHAR, ZEROSUPPRESS);
  DECLARE NUMBER ADDRESS,
    (BASE, CHAR, ZEROSUPPRESS, I, J) BYTE;
  DECLARE TEMP (16) BYTE;
  IF CHARS > LAST(TEMP) THEN CHARS = LAST(TEMP);
  DO I = 1 TO CHARPS;
    J = NUMBER MOD BASE + '0';
    IF J > '9' THEN J = J + 7;
    IF ZEROSUPPRESS AND I <= 1 AND NUMBER = 0 THEN
      J = '0';
    TEMP(LENGTH(TEMP)-1) = J;
  END;
  NUMBER = NUMBER / BASE;
  END;
END PRINTNUMBER;

/*
/* PL/M PROCEDURE TO PRODUCE A HISTOGRAM */

/*
/*
/*

HISTOGRAM : PROCEDURE (MIN, MAX, DTAA, NBINS, CONTRL);

// MIN = MINIMUM VALUE TO BE HIStographed
// MAX = MAXIMUM VALUE TO BE HIStographed
// DTAA = DATA TO BE PLACED INTO HISTOGRAM
// NBINS = NUMBER OF BINS FOR HISTOGRAM
// CONTRL = 0 THEN ZERO HISTOGRAM OUTPUT ARRAY
// 1 THEN PRINT CURRENT HISTOGRAM ARRAY
// 2 THEN INSERT DTAA INTO HISTOGRAM ARRAY

DECLARE (MIN, MAX, DTAA, NBINS, CONTRL) BYTE;
DECLARE ARRAY (255) ADDRESS;
DECLARE (I, J, K) BYTE;
DECLARE (RANGE, PERBIN) ADDRESS;
DECLARE (V, V1) ADDRESS;
DECLARE MXN ADDRESS INITIAL (0);
DECLARE HDR1 DATA ("*IN*=");
DECLARE HDR2 DATA ("* MAX=");
DECLARE HDR3 DATA ("* NBINS=");
DECLARE HDR4 DATA ("* EACH * =");
DECLARE HDR5 DATA ("* UNITS*");
DECLARE BDR1 DATA ("");
DECLARE BDR2 DATA ("");
DECLARE STAR DATA ("*");

/*
IF CONT=0 THEN
/* ZERO OUTPUT ARRAY */
DO I=0 TO LAST(ARRAY);
ARRAY(I)=0;
MXN=0;
END;
ELSE DO;
IF CONT=1 THEN
/* PRINT THE HISTOGRAM */
DO;

/* SCALE OUTPUT TO 72 COLUMNS */ (72-8=64)
V1=SHR(MXN,6)+1;
CALL PRINTSTRING (.HDR1,LENGTH(HDR1));
CALL PRINTNUMBER (MIN,10,6,1);
CALL PRINTSTRING (.HDR2,LENGTH(HDR2));
CALL PRINTNUMBER (MAX,10,6,1);
CALL PRINTSTRING (.HDR3,LENGTH(HDR3));
CALL PRINTNUMBER (NBINS,10,6,1);
CALL PRINTSTRING (.HDR4,LENGTH(HDR4));
CALL PRINTNUMBER (V1,10,6,1);
CALL PRINTSTRING (.HDR5,LENGTH(HDR5));
CALL PRINTSTRING (.CR,LF,LF,3);
DO I=1 TO 36;
CALL PRINTSTRING (.BDR2,LENGTH(BDR2));
END;
CALL PRINTSTRING (.CR,LF,2);
RANGE = MAX-MIN;
PERBIN=RANGE/NBINS;
DO J=0 TO NBINS;
IF I MOD 5 = 0 THEN
DO;
J=MIN+I*PERBIN;
CALL PRINTNUMBER (J,10,6,1);
CALL PRINTSTRING (.BDR2,LENGTH(BDR2));
END;
ELSE CALL PRINTSTRING (.BDR1,LENGTH(BDR1));
V=ARRAY(I);
DO WHILE V>=V1;
CALL PRINTSTRING (.STAR,1);
V=V-V1;
END;
IF V<0 THEN CALL PRINTNUMBER (V,16,1,1);
END;
END;
END;
END;
00121 3 ELSE
00122 3 /* INSERT DTAA INTO CURRENT ARRAY */
00123 3 IF CONTROLL=2 THEN
00124 3 DO;
00125 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)>MNX THEN MNX=ARRAY(J);
00133 5 END;
00134 5 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 PRODUCED 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:
00149 1 CALL HISTOGRAM (0,0,0,0,0);
00150 1 XX=1;
00151 1 DO I = 1 TO 1000;
00152 1 Y=XX;
00153 2 Z=XX AND 1B;
00154 2 W=(Z+ROL(XX,1)) AND 1B;
00155 2 XX=SHL(XX,1) OR W;
00156 2 CALL PRINT$NUMBER(XX,10,6,1);
00157 2 CALL PRINT$STRING (, (CR,LF),2);
00158 2 CALL HISTOGRAM (0,0FFH,XX,50,2);
00159 2 END;
00160 1 CALL HISTOGRAM (0,0FFH,0,50,1);
00161 1 /* RETURN TO THE MONITOR */
00162 1 GO TO MONIT;
00163 1 END
RANDOM NUMBER GENERATOR - RANDOM$BITS

A non-multiplicative pseudo-random number generator.

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

PL/M compiler

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.

Registers Modified:
none

Assembler/Compiler Used:
PL/M VER 3.0

Programmer:
KARL AUERBACH

Company: SYSTEM DEVELOPMENT CORPORATION

RAM Required:
about 300 bytes, but may be decreased.

Address: 2500 Colorado Avenue
Santa Monica, CA, 90406

ROM Required:
about 300 bytes, for program storage

Maximum Subroutine Nesting Level:
PROCEDURE RANDOMSBITS

THIS PROCEDURE PROVIDES THE CALLER WITH SOME NUMBER OF
16 BIT RANDOM NUMBERS.

CALLING SEQUENCE

CALL RANDOMSBITS(POINTER,COUNT);

POINTER IS THE ADDRESS OF THE AREA WHERE THE
RANDOM NUMBERS ARE TO BE PLACED.
COUNT IS A BYTE VALUE CONTAINING THE NUMBER OF
16 BIT RANDOM NUMBERS TO BE PLACED, ONE AFTER
ANOTHER, IN THE AREA DESIGNATED BY POINTER.

VALUES RETURNED
A RANDOM BIT STRING.

SUBPROCEDURES CALLED
NONE

GLOBAL VARIABLES REFERENCED
XTAB
YTAB
XINDEX
YINDEX
VTAB

REMARKS
THE SEQUENCE OF RANDOM NUMBERS GENERATED BY THIS
PROCEDURE IS REPEATABLE, E.G. GIVEN THE SAME
VALUES IN THE GLOBAL VARIABLES XTAB, YTAB, INDEX,
YINDEX, AND VIAND, THE SAME RANDOM NUMBER WILL RESULT.

IT IS EXPECTED THAT THESE GLOBAL VARIABLES WILL BE
INITIALIZED BY SOME OTHER PROCEDURE. THE CONTENTS OF
THE GLOBAL VARIABLES MAY BE ALTERED EXTERNALLY AT ANY TIME
EXCEPT WHEN THIS PROCEDURE IS IN CONTROL.

THE GENERAL TECHNIQUE USED IS TAKEN FROM
KNUUTH, THE ART OF COMPUTER PROGRAMMING VOL. 2
"SEMINUMERICAL ALGORITHMS".

ALGORITHM M PAGE 30.
The method uses two sub-random number generators, X AND Y.
Neither X nor Y need be very good.
A table V is indexed by Y AND THE VALUE THEREIN IS OUTPUT.
X IS USED TO REPLACE THE USED CONTENTS OF V.

HERE WE USE AN ADDITIVE RANDOM GENERATOR FOR BOTH
X AND Y. THE METHOD IS DESCRIBED IN "A BETTER
ADDITIVE CONGRUENTIAL RANDOM NUMBER GENERATOR"
BY ROGER BURFORD AS PUBLISHED IN DECISION SCIENCE
VOLUME 4 NUMBER 2 APRIL '73.

THIS METHOD USES A FIFO LIST.
A RANDOM NUMBER IS GENERATED BY SUMMING THE CONTENTS
OF THE ELEMENTS OF A FIXED-LENGTH FIFO LIST.
The list is then updated by removing the oldest
element from the head of the list and adding the
newly generated random number to the tail of the
list.
The FIFO list is implemented using a table (XTAB
AND YTAB) AND AN INDEX POINTER (XINDEX AND YINDEX).
The index shows the current head and next available
tail. The index is advanced as each random number
is generated and is wrapped around whenever
necessary.

IT IS IMPORTANT THAT THESE TABLES BE INITIALIZED SO THAT
there is at least one odd value in them.
Otherwise it is possible for the tables to reach a
degenerate state where zero is always generated.

*/
DECLARE POINTER ADDRESS;
DECLARE RANDOM BASED POINTER (0) ADDRESS;
DECLARE TADDR ADDRESS;
DECLARE COUNT ADDRESS;
DECLARE (I,J,TBYTE) BYTE;
DO J = 0 TO COUNT-1;
    TADDR,TBYTE = 0;
    DC I = 0 TO LAST(YTAB); /*SUM THE Y FIFO LIST */
    TBYTE = TBYTE + YTAB(I);
END;
YTAB(YINDEX) = TBYTE; /*DELETE OLD HEAD AND ADD NEW TAIL*/
YINDEX = (YINDEX + 1) AND 07H; /*INCREMENT INDEX MODULO 8*/
DO I = 0 TO LAST(XTAB);
    TADDR = TADDR + XTAB(I);
END;
XTAB(XINDEX) = TADDR;
XINDEX = (XINDEX + 1) AND 07H; /*TRYTE MODULO 128*/
BYTE = TBYTE AND 07FH;
00121  3       RANDBN(J) = VTAB(TBYTE); /*GET NEXT 16 BITS OF OUTPUT*/
00122  3       VIAR(TBYTE) = TADDR; /*REPLACE THE USED VALUE*/
00123   3      END;
00124  2      END RANDBNBSITS;
   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 RANDBNBSITS(.TVAL(T1),1);
00132   2      END;
00133   1      DO T1 = 128 TO 200 BY 2;
00134   1      CALL RANDBNBSITS(.TVAL(T1),2);
00135   2      END;
00136   1      CALL RANDBNBSITS(.TVAL(202),54);
00137   1      EOF
NO PROGRAM ERRORS
Factorial of a Decimal Number

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.

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

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).

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, C, D, E, H, L, and Carry</td>
<td>8080 MDS Macro Assembler 1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 bytes</td>
<td>J.L. Marcel Lalonde</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Agriculture Canada</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Engineering Research Service, Ottawa, Ontario, Canada, K1A 0C6</td>
</tr>
</tbody>
</table>
;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 #
000D CD1300 CALL DSUB
0011 C20800 JNZ AGAIN ;FINISHED WHEN A=00
0014 214100 LXI H, SUM ;REG PAIR H= ADDR OF RESULT
0017 C9    RET

DSUB:
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
0025 86    ADD M ;&SUBTRACT 1 FROM IT
0026 27    DAA ;CONVERT TO DECIMAL
0027 77    MOV M, A ;STORE RESULT
0028 C9    RET

DADD:
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: LDA X 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

5-210
003A 033200  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
**Program Title**

BCD UP/DOWN COUNTER

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

**Function**

1 output port and 1 input port.

**Required Hardware**

None

**Required Software**

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

**Input Parameters**

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,B,C,D</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (for COUNT and SLOCT)</td>
<td>Assembler Ver. 4.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>197</td>
<td>Richard Young</td>
</tr>
</tbody>
</table>

**Company:**

Naval Air Rework Facility Code 323

Alameda, Calif 94501

(415) 869-4472
FRACTIONAL MULTIPLY - POSITIVE - 3-BYTE

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

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

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

Registers Modified:
All

RAM Required:
10 bytes

ROM Required:
333 bytes

Maximum Subroutine Nesting Level:

Assembler/Compiler Used:
370/Fortran IV

Programmer:
Wm. James - Dept. 75

Company:
Airesearch Mfg. Co.

Address:
Torrence, CA. 90509
ALGEBRAIC COMPARE SUBROUTINE

ALGEBRAIC COMPARE BETWEEN TWO 16 BIT NUMBERS (A & B) WHICH ARE REPRESENTED IN SIGNED 2'S COMPLEMENT NOTATION.

NONE

NONE

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

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.

Registers Modified:

PSW (A-REG & FLAGS)

RAM Required:
NONE

ROM Required:
31 BYTES

Maximum Subroutine Nesting Level:
NONE WITHIN CPR

Assembler/Compiler Used:
INTELLEC 8/MOD 80 MACRO ASSMB. VERSION 1.0

Programmer:
C. Messerle

Company:
INTECO

Address:
HARRISON, OHIO
; REF. NO. BB43
; PROGRAM TITLE ALGEBRAIC COMPARE SUBROUTINE

; SUBROUTINE CPR --- 08/04/76
;
; ALGEBRAIC 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 F20003 JP  LIKE
0305 7C   MOV  A,H    ; NO. AFFECT CARRY AND
0306 C680  ADI  80H    ; MINUS FLAGS WITH B-VALUE.
0308 031A03 JMP  BELOW
030B 7E   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 9C   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
031E C9
0000

ADI     81H
RET
END
ARC TANGENT (DOUBLE PRECISION) FOR 8080 - DPATH

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.

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

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.

For DPATH, 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.

Results from DPATH, DPADD, DPMPY and DPDIV are all returned in register pairs B-C.
METHOD: The rational approximation below is used:

\[
\tan^{-1} x = \frac{a_0 + a_1 x^2 + a_2 x^4}{b_0 + b_1 x^2 + b_2 x^4}, \quad 0 < x < 1
\]

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

\[
\tan^{-1} x = \frac{a_0 + x^2(a_1 + a_2 x^2)}{b_0 + x^2(b_1 + b_2 x^2)}
\]

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

\[
\begin{align*}
a_0 &= b_0 = b_1 = 75 \text{ AH} \\
a_1 &= 4E \text{ GEH} \\
a_2 &= 04 \text{ FEH} \\
b_2 &= 14 \text{ B8H}
\end{align*}
\]

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 \( a_0 = b_0 = b_1 = 0.459534 \), \( a_1 = 0.306366 \), \( a_2 = 0.019501 \), \( b_2 = 0.080933 \).
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 \( -1_x \)
0009   C30900   S: JMP S   ;Stop
```

The various test cases are tabulated below:

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>8F</td>
<td>66</td>
<td>FF</td>
</tr>
<tr>
<td>80</td>
<td>02</td>
<td>E6</td>
<td>FF</td>
<td>FF</td>
</tr>
<tr>
<td>Output</td>
<td>BD</td>
<td>8E</td>
<td>99</td>
<td>OF</td>
</tr>
<tr>
<td></td>
<td>76</td>
<td>02</td>
<td>BB</td>
<td>CF</td>
</tr>
</tbody>
</table>

\( X_L \) \( X_h \) \( \theta_L \) \( \theta_h \)
ARC.TAN 2 Subroutine

The function is to generate the angle \( \theta = \arctan \frac{y}{x} \) in any of the four quadrants.

Intel 8080 and 400 bytes of memory chips

none

\( x, \) a fractional number, in DE
\( y, \) a fractional number, in HL

\( \theta, \) a fractional number scaled by \( 180^\circ (\pi) \), 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. Otherwise, \( \theta = \frac{\tan^{-1} y}{x} \cdot 180^\circ \)

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Linn Zien</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bytes</td>
<td>Naval Weapons Center</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>371 bytes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>China Lake</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>16K ISIS 8080 Assembler, V 1.0</td>
<td>California 93555</td>
</tr>
</tbody>
</table>
Floating Point Interpreter (FPIP)

Interprets floating point pseudo-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.

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

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

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

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.

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

5-224
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:

<table>
<thead>
<tr>
<th>Floating Point Pseudo-Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEXIT</td>
</tr>
<tr>
<td>FSUB arg</td>
</tr>
<tr>
<td>FDIV arg</td>
</tr>
<tr>
<td>FSTR arg</td>
</tr>
<tr>
<td>FOUT</td>
</tr>
<tr>
<td>FIDIV</td>
</tr>
<tr>
<td>FCOS</td>
</tr>
<tr>
<td>FCOSH</td>
</tr>
</tbody>
</table>

Extended Set:

| FYX arg       | FEX10          | FLG10          |
| PDEC           | PRAD           | FSQU           |
| FISUB arg      | FNEG           | FTAN           |
| FNOP           |                |                |
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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, R0, 2, 3, 4, 5</td>
<td>H. Serindat</td>
<td>Societe ECA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Z.I. Toulon-EST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>47 Bytes</td>
<td>83087 Toulon-Cedex</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>France</td>
</tr>
</tbody>
</table>
; 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
000F CD1300           CALL MULT
0012 C9               RET

0013 29               MULT: DAD H      ; SHIFT PARTIAL RESULT LEFT ONE PLACE.
0014 17               RAL        ; ROTATE MULTIPLIER LEFT TO CARRY.
0015 D21800           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 B          ; DECREMENT BIT COUNTER.
001C C21300           JNZ MULT
001F C9               RET
0000                END MLT12
DECIMAL MULTIPLY SUBROUTINE (N1 x N2 BYTES) - MBCD

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

None

None

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

- Memory RES (N1 + N1 bytes) contains product

\[
\begin{array}{|c|c|c|}
\hline
\text{MRT} & \text{low.ad} \\
\text{MTD} & \text{N2 + 1} \\
\text{KMTD} & \text{N2 + 1} \\
\text{RES} & \text{N1 + N2} \\
\text{LR} & 1 \\
\text{IND} & 2 \\
\hline
\end{array}
\]

fig. 1

Registers Modified: A, B, C, D, E, H, L

Programmer: Renault C.

RAM Required: (2N1 + 3N1 + 5) bytes

Company: Latfeoere

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
ORG 20H

; MBCD: N1 BYTES * N2 BYTES DECIMAL MULTIPLY SUBROUTINE

0020 110702 MBCD:  LXI D, MTD ; MOVE MTD INTO KMTD
0021 210A02 LXI H, KMTD
0022 0E02 MVI B, N2
0023 CDB400 CALL TDBH
0024 8F XRA A
0025 77 MOV M, A
0026 12 STAX D ; CLEAR MSB IN MTD AND KMTD
0027 210002 LXI H, RES ; CLEAR RES AND LR
0028 0E06 MVI B, (N1+N2)+1
0029 77 RAZ: MOV M, A
002A 23 INX H
002B 05 DCR B
002C C23300 JNZ RAZ
002D E01 UN: MVI C, 1
002E 210000 DEUX: LXI H, 0 ; CLEAR INDEX
002F 221302 SHLD IND
0030 2A1302 TROIS: LHLDA IND
0031 110402 LXI D, MTR
0032 19 DAD D
0033 3A1202 LDA LR
0034 B7 ORA A ; FLAG Z AFFECTED
0035 7E MOV A, M
0036 CA5400 JZ QUAT
0037 0F RRC
0038 0F RRC
0039 0F RRC
003A 0F RRC
003B E60F QUAT: ANI 0FH
003C B9 CMP C
003D C27500 JNZ CINO
003E 2A1302 LHLD IND
003F 110002 LXI D, RES
0040 19 DAD D
0041 110A02 LXI D, KMTD
0042 0E03 MVI B, N2+1
0043 CDA800 CALL ABDHD
0044 D27500 JNC CINO
0045 3E00 JERN:  MVI A, 0
0046 8E ADC M
0047 F7 DAA
0048 77 MOV M, A
0049 23 INX H
0072 DA6000          JC   JEAN
0075 211302          CINC: LXI   H, IND
0078 34               INR   M
0079 3E03             MVI   A, N1
007B EE               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 CD9800           CALL   ABDHD
008B 3E0A             MVI   A, 0AH
008D E9               CMP   C
008E C23B00           JNZ   DEUX
0091 3A1202           LDA   LR
0094 B7               ORA   A
0095 C0               RNZ
0096 3C               INR   A
0097 221202           STA   LR   ;lr=1 left
0099 110402           LXI   D, KMTD
009D 210702           LXI   H, MTD
1000 0603             MVI   B, N2+1
1002 CDB400           CALL   TBDH
1005 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
0089 C2B400 JNZ TBDH
008C 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
HISTOGRAM

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.

Teletype of other printer.

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 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.

Graph of incidence of input numeric data.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Robert A. Mikkelson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>389</td>
<td>System Services</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12120 Rochester Avenue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>West Los Angeles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microkit assembler ver 1.0</td>
<td>California 90025</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>W. R. Ott</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>411 Bytes (Decimal)</td>
<td>Applied Data Communications</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1509 East McFadden Avenue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Santa Ana</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEL</td>
<td>California 92705</td>
</tr>
</tbody>
</table>
Conversion of Scientific to easily readable notation.

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.

None

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

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 FBBCD subroutine in program BC-5 directly.

A buffer is created with up to eleven ASCII 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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B. A. Robinson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 bytes for output buffer</td>
<td>Du Pont of Canada Ltd.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1570 bytes</td>
<td>P.O. Box 5000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level: 2 (3 including the CALL to this subroutine).</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kingston</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS Macro Assembler</td>
<td>Ontario, Canada K7L 5A5</td>
</tr>
</tbody>
</table>
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 'EOT'

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
E     X2 X2 X2 X2 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
1234000
-12340
12340
-123.4
12.34
-1.234
.1234
0
-.000001234

NUMBERS LESS THAN 10 EXP. -07 ARE LOADED AS 0.000.
1000 79 CERN: MOV A.C ;FETCH EXP.
1001 E680 ANI 80H ;MASK
1003 CA0B10 JZ SPACE
1006 362D MVI M.2- ;SIGN LOADED IN BUFFER
1008 C30D10 JMP CERN1
100A 3620 SPACE: MVI M.2- ;SPACE LOADED
100E 79 CERN1: INX H
1010 E9C9 EXNEG: MOV A.C ;IS EXPONENT -VE?
1014 E640 ANI 40H
1018 CA5A10 JZ EXP ;2 IMPLIES EXP. POSITIVE
101C 79 MOV A.C ;IS EXP. < -6?
101F E63F ANI 3FH
101F FED7 CPI 07H
1019 D25310 JNC SMALL ;NUMBER TOO SMALL. SET TO 0
101D 362E MVI M.3- ;LOAD A DECIMAL PT.
101F 23 INX H
1021 79 MOV A.C ;MASK SIGNS
1023 E60F ANI 0FH
1025 4F MOV C.A
1027 0D CERN2: DCR C ;STORE (C) ZEROS IN BUFFER
1029 CA2D10 JZ STDE1
102D 3630 MVI M.0- ;
102F 23 INX H
1031 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.
1033 24310 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 C3 RET

1043 1F SHSTR: PRA ;CONVERT (A) TO ASCII & STORE
1044 1F PRA
1045 1F PRA
1046 1F PRA
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 RN2 ;NZ IMPLIES NO D.P. WANTED
104F 362E MVI M.3- ;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 C0E10 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 C0E10 JMP STDE2 ; STORE (DE) WITH D.P.

1068 FE07 LARGE: CPI 07H ; EXP. >3
1069 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 CNRN: DCR C
1074 C0E10 JMP EOTSH ; Z IMPLIES EOT SHOULD BE RE-INSERTED
1077 3630 MVI M,'0' ; STORE (EXP. -3) ZEROS
1079 23 INX H
107A C27310 JMP CNRN

107D 05 AWFUL: PUSH B ; NUMBER TOO LARGE
107E 119410 LXI D,MSG1 ; TRANSFER MSG TO BUFFER
1081 080A MVI C,100 ; SET (C)= NUMB. OF CHAR.
1083 EB CNRN: 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 CNRN
108D C1 POP B
108E 59 MOV E,C ; STORE EXP. ON BUFFER
108F 06FF MVI B,0FFH
1091 C23710 JMP STE

1094 3E20310 MSG1: DB >10 EXP
1098 20455850
109C 2E20
DTMF TO HEX CODE

CONVERTS 2-OF-7 DTMF CODE TO HEXADECIMAL.

NONE

2-OF-7 DTMF IN THE ACCUMULATOR.

<table>
<thead>
<tr>
<th>BIT</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>Ø</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

COLUMN 3 2 1

RETURNS HEXADECIMAL EQUIVALENT IN THE ACCUMULATOR FOR ZERO THROUGH NINE.
RETURNS ØAH FOR *.
RETURNS ØBH FOR #.
RETURNS 80H IF INPUT IS INVALID.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>A, R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler/Compiler Used:</td>
<td>ISIS - 11 8048, V1.2</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>NONE</td>
</tr>
<tr>
<td>Programmer:</td>
<td>FRANK FAFF</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>40H BYTES</td>
</tr>
<tr>
<td>Company:</td>
<td>ATLANTIC RESEARCH CORP.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>ZERO</td>
</tr>
<tr>
<td>Address:</td>
<td>5390 CHEROKEE AVE. ALEXANDRIA, VA. 22314</td>
</tr>
</tbody>
</table>
LOC OBJ SE0 SOURCE STATEMENT

1 :REF. NO. BD3
2 :PROGRAM TITLE DTMHEX
3 :
4 :
5 :
6 :******************************************************************************
7 :
8 : DTMHEX
9 :
10 :******************************************************************************
11 :
12 : CONVERTS 2-0F-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 :
21 : BIT 7 6 5 4 3 2 1 0
22 : ROW 4 3 2 1
23 :
24 DTMHEX: MOV R7,#0 ; CLEAR HEX VALUE
0000 BF00
25 JB0 ROW1 ; JUMP IF ROW 1 TONE PRESENT
0002 1213
26 INC R7 ; OTHERWISE ADD 3 TO HEX VALUE
0004 1F
27 INC R7
0005 1F
28 INC R7
0006 1F
29 JB1 ROW2 ; JUMP IF ROW 2 TONE
0007 3215
30 INC R7
0009 1F
31 INC R7
000A 1F
32 INC R7
000B 1F
33 JB2 ROW3
000C 5217
34 JB3 ROW4 ; ROW 4 IS SPECIAL
000E 722A
35 ERROR: MOV A,#80H ; ERROR VALUE IF NO ROW
0010 2300
36 RET
0012 83
37 ROW1: JB1 ERROR ; ERROR IF MORE THAN ONE ROW
0013 3210
38 ROW2: JB2 ERROR
0015 5210
39 ROW3: JB3 ERROR
0017 7210
40 INC R7 ; ADD ONE TO HEX VALUE
0019 1F
41 JB4 COL1 ; FOR EACH COLUMN
001A 9224
42 INC R7
001C 1F
43 JB5 COL2
001D B226
44 INC R7
001F 1F
45 JB6 COL3
0020 D228
46 JMB ERROR ; ERROR IF NO COLUMN
0022 0418
47 COL1: JB5 ERROR ; ERROR IF MORE THAN ONE COLUMN
0024 B210
48 COL2: JB6 ERROR
0026 D210
49 COL3: MOV A,R7 ; GET HEX VALUE
0028 FF
50 RET ; AND RETURN
002A 9232
51 JB4 STAB ; TONE = *
002C B239
52 JB5 ZERO ; ZERO
**Source Statement**

<table>
<thead>
<tr>
<th>JC</th>
<th>OBJ</th>
<th>53</th>
<th>JBS</th>
<th>NUMB</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>002E D23D</td>
<td>0030 0410</td>
<td>0032 B210</td>
<td>D210</td>
<td>0036 230A</td>
<td>0038 83</td>
</tr>
<tr>
<td>0030 0410</td>
<td>0032 B210</td>
<td>0034 D210</td>
<td>0036 230A</td>
<td>0038 83</td>
<td>0039 D210</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0036 230A</td>
<td>0038 83</td>
<td>0039 D210</td>
</tr>
<tr>
<td>0030 0410</td>
<td>0032 B210</td>
<td>0034 D210</td>
<td>0036 230A</td>
<td>0038 83</td>
<td>0039 D210</td>
</tr>
<tr>
<td>0030 0410</td>
<td>0032 B210</td>
<td>0034 D210</td>
<td>0036 230A</td>
<td>0038 83</td>
<td>0039 D210</td>
</tr>
</tbody>
</table>

**User Symbols**

<table>
<thead>
<tr>
<th>COL1</th>
<th>0024</th>
<th>COL2</th>
<th>0026</th>
<th>COL3</th>
<th>0028</th>
<th>DTAHEX</th>
<th>0000</th>
<th>ERROR</th>
<th>0010</th>
<th>NUMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW3</td>
<td>0017</td>
<td>ROW4</td>
<td>002A</td>
<td>STAR</td>
<td>0032</td>
<td>ZERO</td>
<td>0009</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ASSEMBLY COMPLETE, NO ERRORS
8048 BCD MULTIPLY
(Decimal Multiply 6 digit x 4 digit or less)

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.

Prompt 48

In a Prompt 48 you can use system calls and by this you get a calculator-like operation.

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.

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 to R7 and R12 to R16</td>
<td>ASM 48</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>None</td>
<td>Karl-Magnus Heinrichs</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>61 bytes</td>
<td>Vaaka-Nyholm</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>None</td>
<td>Finland</td>
</tr>
<tr>
<td></td>
<td>Oskelantie 1, 00320 Helsinki</td>
</tr>
</tbody>
</table>
**LISTING**

**LOC**  **OBJ**  **SEQ**  **SOURCE STATEMENT**

```
0060 0060 A5
0061 FD
0062 37
0063 0301
0065 E668
0067 B5
0068 57
0069 37
006A 0300
006C 57
006D AD
006E 7683
0070 97
0071 BF03
0073 B002
0075 B912
0077 F1
0078 70
0079 57
007A 18
007B A1
007C 19
007D EF77
007F F691
0081 0460
0083 FE
```

```
ORG 60H

SUB:
CLR   F1   ; CLEAR FLAG 1
MOV   A, R5   ; LOAD ACCUMULATOR WITH R
CPL   A   ; COMPLEMENT ACCUMULATOR
ADD   A, #1   ; SUBTRACT ONE FROM ACCUM
JNC   NOBO   ; JUMP ON NO BORROW TO "N"
CPL   F1   ; SET FLAG 1
NOBO:
DA   A   ; DECIMAL ADJUST OF ACCUM
CPL   A   ; COMPLEMENT ACCUMULATOR
ADD   A, #0   ; PREPARE ACCUMULATOR FOR
DA   A   ; DECIMAL ADJUST OF ACCUM
MOV   R5, A   ; LOAD BACK ACCUMULATOR T
JF1   BORROW   ; JUMP IF FLAG 1 IS SET T
MUL:
CLR   C   ; CLEAR CARRY BIT
MOV   R7, #3   ; LOAD R7 WITH 3 FOR ADDR
MOV   R0, #2   ; LOAD R0 WITH 2 FOR ADDR
MOV   R1, #18   ; LOAD R1 WITH 18 FOR ADD
ADI:
MOV   A, @R1   ; LOAD ACCUMULATOR WITH O
ADDR   A, @R0   ; ADD ACCUMULATOR TO MULT
ADDC   A, #0   ; ADD ACCUMULATOR TO MULT
INC   R0   ; INCREMENT REGISTER 0 FOR
MOV   @R1, A   ; LOAD ACCUMULATOR INTO R
INC   R1   ; INCREMENT R1 FOR NEW RE
DJNZ   R7, ADI   ; DECREMENT R7 AND IF NO
JC   CARRY   ; JUMP IF CARRY BIT TO "C"
JMP   SUB   ; JUMP TO "SUB" FOR NEXT
BORROW:
MOV   A, R6   ; LOAD ACCUMULATOR WITH R
CPL   A   ; COMPLEMENT ACCUMULATOR
ADD   A, #1   ; SUBTRACT ONE FROM ACCUM
JC   WAIT   ; JUMP ON BORROW TO "WAIT"
DA   A   ; DECIMAL ADJUST OF ACCUM
CPL   A   ; COMPLEMENT ACCUMULATOR
ADD   A, #0   ; PREPARE ACCUMULATOR FOR
DA   A   ; DECIMAL ADJUST OF ACCUM
MOV   R6, A   ; LOAD BACK ACCUMULATOR T
```

**5-254**
<table>
<thead>
<tr>
<th>JC</th>
<th>OBJ</th>
<th>SEQ</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>008F</td>
<td>0470</td>
<td>53</td>
<td>JMP MUL ; JUMP TO &quot;MUL&quot;</td>
</tr>
<tr>
<td>0091</td>
<td>27</td>
<td>54</td>
<td>CLR A ; CLEAR ACCUMULATOR</td>
</tr>
<tr>
<td>0092</td>
<td>F7</td>
<td>55</td>
<td>RLC A ; ROTATE ACCUMULATOR LEFT</td>
</tr>
<tr>
<td>0093</td>
<td>61</td>
<td>56</td>
<td>ADD A, @R1 ; ADD ACCUMULATOR TO PC</td>
</tr>
<tr>
<td>0094</td>
<td>57</td>
<td>57</td>
<td>DA A ; DECIMAL ADJUST OF ACCUM</td>
</tr>
<tr>
<td>0095</td>
<td>A1</td>
<td>58</td>
<td>MOV @R1, A ; LOAD BACK ACCUMULATOR</td>
</tr>
<tr>
<td>0096</td>
<td>19</td>
<td>59</td>
<td>INC R1 ; INCREMENT R1 FOR NEXT R</td>
</tr>
<tr>
<td>0097</td>
<td>F691</td>
<td>60</td>
<td>JC CARRY ; JUMP ON &quot;CARRY&quot; TO &quot;CAR</td>
</tr>
<tr>
<td>0099</td>
<td>0460</td>
<td>61</td>
<td>JMP SUB ; JUMP TO &quot;SUB&quot; FOR NEXT</td>
</tr>
<tr>
<td>009B</td>
<td>049B</td>
<td>62</td>
<td>JMP WAIT ; RESULT IN REG. R12, R13</td>
</tr>
</tbody>
</table>

USER SYMBOLS
ADI 0077 BORROW 0083 CARRY 0091 MUL 0070 NOBO 0068 SUB

ASSEMBLY COMPLETE, NO ERRORS
Hex to ASCII Conversion

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.

None

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

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)

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

Company: Fluke Trendar

Address: 630 Clyde Ave.

City: Mt. View

State: Calif. 94043

RAM Required: Depends on input string length

ROM Required: 49 bytes

Maximum Subroutine Nesting Level: 2

Assembler/Compiler Used: ASM80 V1.0
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 CHARACT

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 ;CONVER SEAST 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

5-258
RESULT, AND OR WITH 40H TO FORM
NOTE: CONV PRODUCES ODD PARITY AS SHOWN, FOR EVEN PARITY CHANG

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 C3200 JMP PAR ; GO TO SET PARITY
002A F630 ORI 30H ; OR WITH 30H TO COMPLETE ASCII CHAR
002C E20000 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
FLOATING POINT PACKAGE - OPTIMISED ULTRA FAST

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

SBC 80/10 or similar

SBC 80/10 P Monitor or similar

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

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

Program offered on diskette only.

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

© Intel Corporation, 1976

Ref. #BC18
"AS2FL" - ASCII STRING TO INTEL FLOATING POINT NUMBER

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

MDS 800 with ISIS II Linking Loader

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

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

String Format:

\[ \pm xxxxx.xxx \quad E \pm xx \# \]

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

Revised 2/78

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSUME ALL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>BART EVANS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLES: 24</td>
</tr>
<tr>
<td>STACK: 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DURRUM INSTRUMENT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE: 814</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1228 Titan Way</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Level after entry</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunnyvale</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/M-80 V3.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976

98-034D

R/78

5-262
$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: PROCEEDURE (FPNADDR, STRNADDR) PUBLIC;
21 2     DECLARE (FPNADDR, STRNADDR) ADDRESS;
22 2     DECLARE FPR BASED FPNADDR (18) BYTE;
23 2     DECLARE STRING BASED STRNADDR (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:
' 1000
' -1 0000
' 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.

2/6/78
(3) NO SPACES ALLOWED BETWEEN DIGITS.
(4) EXPONENT IS ASSUMED TO BE POSITIVE UNLESS A MINUS SIGN EXISTS.

/*

24 2 DECLARE

    TRUE LITERALLY '1111111B',
    FALSE LITERALLY '0000000B',

    DIG (4) BYTE,
    NUM (4) BYTE,
    FRAC (4) BYTE,
    DIV (4) BYTE,
    TEN (4) BYTE DATA (0, 0, 2OH, 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 */

42 3 IF STRING(I) = '"' THEN DO:
43 3     I = I+1;
44 3     GO TO CHAR1;
45 3     END;
47 3 ELSE IF STRING(I) = '"' THEN DO:
48 3     POS#NUM = FALSE;
49 3     I = I+1;
50 3     END;
52 3 ELSE IF STRING(I) = '"' THEN DO:
53 3     I = I+1;
55 3 END;

CHAR2:  /* BRANCH ON '"' OR '"'; ELSE COMPUTE MANTISSA */
IF STRING(I) = '. ' THEN DO;
   I = I+1;
   GO TO POINT;
   END;
ELSE IF STRING(I) = ' ' OR STRING(I) = 'E' THEN DO;
   I = I+1;
   GO TO EXP0;
   END;
ELSE IF STRING(I) < '0' OR STRING(I) > '9' THEN DO;
   GO TO FINAL;
   END;
ELSE DO;
   CALL DIGIT;
   CALL LOAD (FPR, NUM);
   CALL FMUL (FPR, TEN);
   CALL FADD (FPR, DIG);
   CALL FSTOR (FPR, NUM);
   I = I+1;
   GO TO CHAR2;
   END;

POINT: /* COMPUTE FRACTIONAL PART */

IF STRING(I) >= '0' AND STRING(I) <= '9' THEN DO;
   CALL DIGIT;
   CALL LOAD (FPR, DIV);
   CALL FMUL (FPR, TEN);
   CALL FSTOR (FPR, DIV);
   CALL LOAD (FPR, DIG);
   CALL FADD (FPR, DIG);
   CALL FSTOR (FPR, FRAC);
   I = I+1;
   GO TO POINT;
   END;
ELSE IF STRING(I) = ' ' OR STRING(I) = 'E' THEN DO;
   I = I+1;
   GO TO EXP0;
   END;
ELSE GO TO FINAL;

EXPO: /* COMPUTE EXPONENT */

IF STRING(I) = ' ' OR STRING(I) = 'E' THEN DO;
   I = I+1;
   GO TO EXP0;
   END;
ELSE IF STRING(I) = '-' THEN DO;
   POS#EXP = FALSE;
   I = I+1;
   END;
ELSE IF STRING(I) = '+' THEN DO;
   I = I+1;
   END;

DO WHILE STRING(I) >= '0' AND STRING(I) <= '9';
EXP = (10*EXP) + (STRING(I) - '0');
I = I+1;
END;

FINAL: /* COMBINE AND PERFORM NORMALIZATION */

CALL FLOAD (FPR, NUM);
CALL FADD (FPR, FRAC);

IF NOT POS#NUM THEN FPR(17) = 10000000;

IF POS#EXP THEN DO I = 1 TO EXP;
    CALL FMUL (FPR, TEN);
END;
ELSE DO I = 1 TO EXP;
    CALL FDIV (FPR, TEN);
END;

END AS2FL;

END AS2FL;

MODULE INFORMATION:

CODE AREA SIZE = 032EH  814D
VARIABLE AREA SIZE = 0013H   24D
MAXIMUM STACK SIZE = 0004H   4D
186 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
"FL2AS" - INTEL FLOATING POINT NUMBER TO ASCII STRING

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

Intel MDS 800 with ISIS II Linking Loader

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

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)"

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.

Revised 2/78

BC19B is offered as one program with BC19A.

Registers Modified:
ASSUME ALL

RAM Required:
VARIABLE: 28 STACK: 4

ROM Required:
CODE: 828

Maximum Subroutine Nesting Level:
1 LEVEL AFTER ENTRY

Assembler/Compiler Used:
PL/M-80 Ver.3.0

Programmer: Bart Evans

Company: Durrum Instrument

Address: 1220 Titan Way

City: Sunnyvale,
# Title

```
FILE = 'F1.F2AS LST
PL/M-88 Compiler

*** FLOATING POINT TO ASCII STRING ***

ISIS-II PL/M-88 V3.0 Compilation of Module F2AS
Object Module Placed in .F1.F2AS.OBJ
Compiler Invoked By: plm88 .F1.F2AS PLM DATE(0931JAN9)

Title(' FLOATING POINT TO ASCII STRING ')

1  F2AS: DO;
2    FLOAD: PROCEDURE (A,B) EXTERNAL;
3      DECLARE (A,B) ADDRESS;
4      END FLOAD;

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

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

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

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

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

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

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

26   FCMFR: PROCEDURE (A,B) EXTERNAL;
27      DECLARE (A,B) ADDRESS;
28     END FCMFR;

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

29    F2AS: PROCEDURE (FP#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 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
```

2/6/78
WHERE 'N' 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#ADDR, STR#ADDR) ADDRESS;
31 2 DECLARE FPR BASED FPR#ADDR (18) BYTE;
32 2 DECLARE STR BASED STR#ADDR (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, 06H, 4CH),
   P10E6 (4) BYTE DATA (7FH, 96H, 19H, 48H),
   TEN (4) BYTE DATA (0, 0, 20H, 41H),

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

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

34 2 MAIN: /* CHECK FOR ZERO. IF SO, LOAD '0 000000' AND RETURN */

   DO I = 0 TO 13;
   STR (I) = '/';
   END;
35 3
36 3
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 3
MAINTISSSIGN: /* IF NEGATIVE, LOAD '−' AND NEGATE */

IF FPR (17) = 10000000 THEN DO;
   FPR (17) = 0;
   STR (0) = '−';
   END;

CALL FSTOR (FPR, NUM);

EXP = 100;

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);
CALL FCMPR (FPR, P100E6);
IF (FPR (0) AND 110000000) 0 0 THEN DO; /* X = 100,000,000 */
   EXP = EXP+1;
   CALL FLOAD (FPR, NUM);
   CALL FDIV (FPR, TEN);
   CALL FSTOR (FPR, NUM);
   GO TO NORMALIZE1;
   END;

NORMALIZE2:

CALL FLOAD (FPR, NUM);
CALL FCMPR (FPR, P100E6);
IF (FPR (0) AND 001000000) 0 0 THEN DO; /* X < 100,000,000 */
   EXP = EXP-1;
   CALL FLOAD (FPR, NUM);
   CALL FMUL (FPR, TEN);
   CALL FSTOR (FPR, NUM);
   GO TO NORMALIZE2;
   END;

EXTRACTDIGITS: /* REMOVE DIGITS FROM RIGHT TO LEFT —
   DIGIT = N − 10 * INT(N/10)
   N' = INT(N/10) */

DO I = 0 TO 7;
   CALL FLOAD (FPR, NUM);
   CALL FDIV (FPR, TEN);
   CALL FIXSD (FPR, INT);
   CALL FLTDS (FPR, INT);
   CALL FMUL (FPR, TEN);
   CALL FSTOR (FPR, REM);
   CALL FLDS (FPR, .INT);
   CALL FSUB (FPR, .REM);
   CALL FIXSD (FPR, .REM);
   TMPSTR (7-I) = REM (0) + 90H;
   CALL FLTDS (FPR, .INT);
   CALL FSTOR (FPR, NUM);
   END;

2/6/78
ROUNDOFF: /* ROUND TO NEAREST 7 DIGITS */

TMP*STR (7) = DEC (TMP*STR (7) + 5);
TMP*STR (6) = DEC (TMP*STR (6) PLUS 0);
TMP*STR (5) = DEC (TMP*STR (5) PLUS 0);
TMP*STR (4) = DEC (TMP*STR (4) PLUS 0);
TMP*STR (3) = DEC (TMP*STR (3) PLUS 0);
TMP*STR (2) = DEC (TMP*STR (2) PLUS 0);
TMP*STR (1) = DEC (TMP*STR (1) PLUS 0);
TMP*STR (0) = DEC (TMP*STR (0) PLUS 0);

RESOLVE: /* IF .1 <= N <= 99999999 THEN LOAD WITH SPACES
INSTEAD OF 'E XX' ELSE COMPUTE EXPONENT AND LOAD
IN 'E' FORMAT */

J = 1;

IF EXP >= 92 AND EXP <= 99 THEN DO;
   DO I = 0 TO 6;
      IF EXP-92 = 1 THEN DO;
         STR (J) = '.';
         J = J+1;
      END;
      STR (J) = (TMP*STR (1) AND 0FH) + '0';
      J = J+1;
   END;
   RETURN;
END;
ELSE DO:
   POS*EXP = TRUE;
   IF EXP < 93 THEN DO;
      EXP = 93 - EXP;
      POS*EXP = FALSE;
   END;
   ELSE EXP = EXP - 93;

   DO I = 0 TO 6;
      IF I = 1 THEN DO;
         STR (J) = '.';
         J = J+1;
      END;
      STR (J) = (TMP*STR (1) AND 0FH) + '0';
      J = J+1;
   END;
   STR (10) = 'E';

   IF NOT POS*EXP THEN STR (11) = '-';
   I = EXP/10;
   STR (12) = I + '0';
   STR (13) = (EXP - 10*I) + '0';
   END;
   END FL2AS;
   END FL2AS;
PL/M-80 COMPILER

***** FLOATING POINT TO ASCII STRING *****

MODULE INFORMATION:

CODE AREA SIZE = 033CH 828D
VARIABLE AREA SIZE = 001CH 28D
MAXIMUM STACK SIZE = 0004H 40
222 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
8080 FLOATING POINT A\(^b\), EXP AND NATURAL LOG FUNCTIONS FOR USE WITH FLOATING POINT MATH PKG. BC1 and BC2

\[ A^b \] is evaluated by \( X = \text{EXP} (B \times \text{Ln}(A)) \), and has been checked to 9.999 (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.

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

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

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

\[ \text{EXP} = 1 + X + X^2/2 + \ldots \text{ which is very accurate for } X<0.5 \]

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

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

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

The result is then added to \( N \times \text{Ln}(2) \) 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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Edward C. Perkins</td>
<td></td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
<td>Adams-Smith, Inc.</td>
</tr>
<tr>
<td>18-Bytes + FP math RAM</td>
<td>Address:</td>
<td>Summer Road</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>City:</td>
<td>Boxboro</td>
</tr>
<tr>
<td>161 H Bytes</td>
<td>State:</td>
<td>MASSACHUSETTS 01719</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 + FP pkg.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAC68 V2.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ORG 800H ; TEMP LOC

TOEX: LXI D, FLOOR + 4 ; EXPONENT
LXI H, FLOOR + VALUE
CALL FPWR ; VALUE ** EXPONENT
IN 1 ; PAUSE W/RESULT IN ACCS

TOEX: LXI H, FLOOR + X
CALL FLOT ; LOAD
CALL FEXP ; EXP(X)
LXI H, FLOOR + 4 ; STORE
CALL FSTR
IN 1 ; PAUSE
TOEX: LXI H, FLOOR + 4 ; EXP(X)
CALL FLOT ; LOAD IT
CALL FNLOG ; LN(EXP(X))
IN 1 ; PAUSE W/RESULT IN ACCS
HLT ; STOP RUNNING OFF END

FLOAT EQU 0018H ; VALUE

FLTAC - FLOAT VALUE IN A REG
MVI E, B ; CLEAR DATA IN A
MVI B, 0
MOV C, B
MOV D, B
CALL FLT ; FLOAT
RET ; DONE

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

FPWR: PUSH D ; SAVE B
CALL FLOT ; LOAD A
CNZ FNLOG ; A NOT 0, SO TAKE LN(A)
POP H ; B
CALL FMUL ; B * LN(A)
JMP FEXP ; EXP TO GET A**B

FEXP - EXP(FP ACCUM), USING SERIES EVALUATION
EXP = 1 + X + X**2 / 2! + .......
ENTRY WHEN HL = ADDR OF X
0040 CD6F12 FEXPH: CALL FLOD ; LOAD X
0043 AF FEXP: XOR A ; RESET SCALE COUNT
0044 325118 STA TXSCL
0047 CD6A12 CALL FTST ; CHECK SIGN OF X
004A FE6008 JP FEXPP ; X IS +
004D CD5212 CALL FABS ; MAKE X +
0050 CD6008 CALL FEXPP ; EXP(+X)

0053 21DC08 LXI H,FPONE ; EXP(-X) = 1/EXP(X)
0056 CD6F12 CALL FLOD ; 1.0
0059 214418 LXI H,TEXP ; EXP(+X)
005C CD412 CALL FDIV ; EXP(-X)
005F C9 RET ; DONE

0060 FEB0 FEXPP: CPI 00H ; SEE IF X > 0.5
0062 DA6C08 JC FEXOK ; X < 0.5

; FOR ACCURACY, WITH FEW TERMS, X IS SCALED /2**N
; SO 0 < X < 0.5, THEN EXP(X) IS SQUARED N TIMES
0065 D67F SUI 7FH ; GET N FOR /2**N
0067 325118 STA TXSCL ; SAVE SCALE
006A 3E7F MVI A.7FH ; X > 0.5
006C 214018 FEXOK: LXI H,TMPX ; SAVE X
006F E5 PUSH H ; SAVE
0070 CD3E12 CALL FSTR
0073 E1 POP H
0074 CD6F12 CALL FLOD ; LOAD SCALED X, REPLACING ORIG X
0077 214818 LXI H,TERMX ; INIT TERM
007A CD3E12 CALL FSTR
007D 215018 LXI H,TDNOM ; DENOMINATOR
0080 3601 MVI M,1 ; SET AT 1
0082 21DC08 LXI H,FPONE ; 1.0
0085 CD0012 FXSUM: CALL FADD ; 1+X
0088 214418 LXI H,TEXP ; SUM
008B CD3E12 CALL FSTR
008E 3A5018 LDA TDNOM ; INC DENOM
0091 3C INR A
0094 232518 STA TDNOM
0097 FE08 CPI MXTRM ; MAX # TERMS?
009A CA1608 JZ FEXPX ; YES-EXIT SCALING EXP(X)

; NOW CALC NEXT TERM
009D CD2808 CALL FLTAC ; FLOAT DENOM
009F 214818 LXI H,TMPD ; SAVE IT
00A2 CD3E12 CALL FSTR
00A5 214818 LXI H,TERMX ; TERM
00AA CD6F12 CALL FLOD
00AF 214018 LXI H,TMPX ; X
00B3 CD3C12 CALL FMUL ; TERM*X
00B6 214C18 LXI H,TMPD ; DENOM
00BE CD412 CALL FDIV ; TERM/DENOM

2/6/78
0B5 21481B  LXI H,TERMX
0BB  CD3E12  CALL FSTR ;STORE NEW TERM
0BB  21441B  LXI H,TEPX ;SUM
0BBE  C38508  JMP FXSUM ;SUM IT

;EXP(X) CALCULATED, SCALE IF NEEDED
0C1  3A511B  FEXPX:  LDA TXSCL ;SCALE COUNT
0C4  FE00   CP1  0
0C6  CA5A12  JZ  FTST ;NO SCALING, X WAS < .5

;SCALE EXP BY SQUAREING
0C9  67  MOV H,A ;SET COUNT
0CA  E5  FEXSQ:  PUSH H ;SAVE COUNT
0CB  21441B  LXI H,TEPX ;EXP
0CE  E5  PUSH H
0CF  C9BC12  CALL FMUL ;SQUARE IT
0D2  E1  POP H
0D3  C9BE12  CALL FSTR ;SAVE
0D6  E1  POP H
0D7  25  DCR H ;COUNT
0D8  C2CA08  JNZ FEXSQ ;MORE
0DB  C9  RET ;DONE

0DBB MXTRM EQU 8 ;GO TO X**7, GIVES 7 PLACES FOR X,.5
                  ;FOR ACCURACY 1.E-7 ERROR

0DC  B1000000 FPONE:  DB 81H,0,0,0 ;1.0

1840  TMPX  EQU 1840H ;X HOLD
1844  TEPX  EQU TMPX+4 ;SERIES SUM
1848  TERMX EQU TEPX+4 ;TERM
184C  TMPD  EQU TERMX+4 ;DENOM
1850  TDNUM EQU TMPD+4 ;8 BIT DENOM INTEGER
1851  TXSCL EQU TDNUM+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+1)**3+1/5(X-1/X+1)**5+...)
;ACCURATE TO 6 DECIMALS OVER INTERVAL 1 <= 2
;RETURNS LN IN FP ACCUM

0E0  CD6F12 ;ENTRY WHEN HL=ADDR OF X
          FNLOG:  CALL FLOD ;LOAD X

0E3  5F  FNLOG:  MOV E,A ;GET EXP
0E4  3E81  MVI A,01H ;SCALE X, 1<=X<2
0E6  21401B  LXI H,TMPX ;STORE X
0E9  CD3E12  CALL FSTR
0EC  7B  MOV A,E ;ORIG EXP
0ED  D081  SUI 81H ;ADJ
0EF  CD2808  CALL FLTAC ;POWER OF 2
0F2  216209  LXI H,LOC2 ;LN(2)
08F5 CD8C12  CALL FMUL ;INIT LN
08F6 214418  LXI H.TMPY ;Y
08FB CD3E12  CALL FSTR ;STORE
08FE 3E01  MVI A, 1 ;K+1
0900 323018  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 CD8D12  CALL FADD ;X+1
0910 E1  POP H ;SAVE
0911 E5  PUSH H
0912 CD3E12  CALL FSTR
0915 21B609  LXI H.FPTWO ;NOW X-1
0918 CD3D12  CALL FSUB
091B E1  POP H ;X+1
091C E5  PUSH H
091D CD4412  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 214018  LXI H.TMPZ
092C CD3E12  CALL FSTR
092F 214018  LXI H.TMPX ;LOAD INIT TERM
0932 CD6F12  CALL FLOD
0935 C349FF  JMP FNLSM ;START SUMMING
  |NEW TERM = TERM x Z (INIT TERM**2)
0938 214018  FNLC2: LXI H.TMPX ;TERM
093B E5  PUSH H ;SAVE
093C CD6F12  CALL FLOD ;LOAD IT
093F 214018  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
  LDA TFNLK ;GET K
0951 3A5018  CALL FLTAC ;FLOAT IT
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
095E E3           XTHL  :SWAP W/2*TERM
0960 CD6F12       CALL FLOAD  :2*TERM
0963 E1           POP H ;K
0964 CDB412       CALL FDIV  :SET FOR SUM
0967 21441B       LX1 H,TMPY ;SUM
096A E5           PUSH H
096E CDD812       CALL FADD
096F E1           POP H
0970 CD3E12       CALL FSTR  :NEW SUM
0972 3A501B       LDA TFNK  ;NEXT K+K+2
0973 C602         ADI 2 ;K=1,3,5,...
0977 325018       STA TFNK
097A FE0C         CPI 12 ;6 TERMS?
097C DA3B09       JC FNLC2 ;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 TEKP ;Y
1848   TMPZ EQU TERMX ;Z
1850   TFNK EQU TDNOM ;K
1851   TMPN EQU TFNK+1 ;TEMP STORE

;EXTERNALS
1500   FLT EQU 1500H ;FLOAT ROUTINE
1208   FADD EQU 1208H ;ADD
1203   FSUB EQU 1203H ;SUBTRACT
120C   FMUL EQU 120CH ;MULTIPLY
1244   FDIV EQU 1244H ;DIVIDE
126F   FLOAD 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
Floating Point Utility Programs for Use with FPAL.LIB

FFBCD - Converts floating point number in FAC to BCD in buffer memory. Fixed point for values .000000 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.

FPAL.LIB

No special hardware required

BC22A is offered as one program with BC22B.

Available on diskette only for $35.00.

<table>
<thead>
<tr>
<th>Registers Modified: All registers restored unless result in that register</th>
<th>Programmer: James C. Follansbee</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required: Varies with function</td>
<td>Company: J F Microsystems</td>
</tr>
<tr>
<td>ROM Required: Varies with function</td>
<td>Address: 5617 W Argent Rd</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level: Varies with function</td>
<td>City: Pasco</td>
</tr>
<tr>
<td>Assembler/Compiler Used: ISIS-II MACRO V2.0</td>
<td>State: Washington 99301</td>
</tr>
</tbody>
</table>
SBC-310 FLOATING POINT SYSTEM FOR USE WITH SINGLE OR MULTIPLE SBC-80/20 Processors
INTERFACES SBC-80/20 PROGRAMS WITH HIGH SPEED MATH BOARD, SBC-310

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.)

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)

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 *
FFLDS CONVERT FIXED MEMORY TO FLOATING ACCUMULATOR *
FFCMPR COMPARE ACCUMULATOR TO MEMORY VALUE *
FFZIST 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 ACCUMULATOR INTO TEMPORARY REGISTER
FFRSTR RESTORE TEMPORARY REGISTER TO ACCUMULATOR

* OPERATOR OR MEMORY POINTED TO BY D & E REGISTERS

Note: BC22B is offered as one program with BC22A.

<table>
<thead>
<tr>
<th>Registers Modified: ALL REGISTERS/FAC RESTORED UNLESS RESULT THEREIN</th>
<th>Programmer: JAMES C. FOLLANSBEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required: VARIES WITH FUNCTION</td>
<td>Company: J F MICROSYSTEMS</td>
</tr>
<tr>
<td>ROM Required: VARIES WITH FUNCTION</td>
<td>Address: 5617 W ARGENT RD</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level: VARIES WITH FUNCTION</td>
<td>City: PASCO</td>
</tr>
<tr>
<td>Assembler/Compiler Used: ISIS-II MACRO V2.0</td>
<td>State: WASHINGTON 99301</td>
</tr>
</tbody>
</table>
RELOCATABLE FMATH AND XMATH, 8085 FLOATING POINT PACKAGE

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

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

8080

ISIS II for system generation

Source available on diskette for $35.00.
Source Listing available for $15.00.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>See documentation</td>
<td>Richard Allen</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td></td>
<td>Texas Microsystems, Inc.</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td></td>
<td>6610 Harwin, Suite 125</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td></td>
<td>Houston</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
<tr>
<td>ISIS II MACRO ASSEMBLER</td>
<td>Texas 77036</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, R0, 2, 3, 4, 5</td>
<td>H. Serindat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Societte ECA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>47 Bytes</td>
<td>Z.I. Toulon-EST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>83087 Toulon-Cedex</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>France</td>
</tr>
</tbody>
</table>
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
ARRAY ADDRESSING SUBROUTINE AND CALLING MACRO

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>H, L</th>
<th>Programmer: Roy G. Witton</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>10 bytes dec.</td>
<td>Company: GKN Sankey Limited</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>ARRAY subroutine uses 139 bytes dec.</td>
<td>Address: Tweedale Industrial Site, Madeley,</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>1 level used</td>
<td>City: Telford,</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>8080 MDS Macro Assembler Ver. 1.0</td>
<td>State: Salop TF7 4JR, England. Tel: Telford 586261</td>
</tr>
</tbody>
</table>
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, 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 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 transferred 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 = ZERO + (I x IW)\* + (J x IW x ISIZE)\*\* + (K x IW x ISIZE x JSIZE)
```

ADDR represents the final indexed address returned by the subroutine.
ZERO 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.
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.
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>SEQ</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>ARRAY ADDRESSING VIA SUBSCRIPTS</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>MDS ADDRESSES USED</td>
</tr>
<tr>
<td>7</td>
<td>C0</td>
<td></td>
<td>EQU 0F809H ; CONSOLE OUTPUT</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>RAM LOCATIONS USED</td>
</tr>
<tr>
<td>13</td>
<td>I</td>
<td></td>
<td>EQU 1000H ; I SUBSCRIPT (INDEX)</td>
</tr>
<tr>
<td>14</td>
<td>J</td>
<td></td>
<td>EQU I+2 ; J SUBSCRIPT (INDEX)</td>
</tr>
<tr>
<td>15</td>
<td>K</td>
<td></td>
<td>EQU J+2 ; K SUBSCRIPT (INDEX)</td>
</tr>
<tr>
<td>16</td>
<td>PLIST</td>
<td></td>
<td>EQU K+2 ; SUBROUTIN ARGUMENTS</td>
</tr>
<tr>
<td>17</td>
<td>TABLE</td>
<td></td>
<td>EQU 2000H ; ARRAY BEING ADDRESSED</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td>MACRO TO INITIALISE &amp; CALL ARRAY SUBRBTN</td>
</tr>
<tr>
<td>22</td>
<td>ARSET</td>
<td></td>
<td>MACRO ZERO, IW, ISIZE, JSIZE, KSIZE</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>LXI H.(IW SHL 8 XOR ISIZE) AND 0FFFFH</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td>SHLD PLIST ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td>LXI H.(JSIZE SHL 8 XOR KSIZE) AND 0FFFFH</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td>SHLD PLIST+2 ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td>LXI H.ZERO ; TO ADDR ZERO LOCN OF ARRAY</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td>CALL ARRAY ; CALL ARRAY SUBRBTN</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td>ENDM</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td>TEST PROGRAM</td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td>ORG 0100H</td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td>LXI SP, 3000H ; SET STACK POINTER</td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td>JMP TEST ; BEGIN TEST PROGRAM</td>
</tr>
<tr>
<td>39</td>
<td></td>
<td></td>
<td>ORG 0120H</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td>TEST: ARSET TABLE, 4, 4, 4, 4</td>
</tr>
<tr>
<td>41+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42+</td>
<td></td>
<td></td>
<td>LXI H.(4 SHL 8 XOR 4) AND 0FFFFH</td>
</tr>
<tr>
<td>43+</td>
<td></td>
<td></td>
<td>SHLD PLIST ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>44+</td>
<td></td>
<td></td>
<td>LXI H.(4 SHL 8 XOR 4) AND 0FFFFH</td>
</tr>
<tr>
<td>45+</td>
<td></td>
<td></td>
<td>SHLD PLIST+2 ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>46+</td>
<td></td>
<td></td>
<td>LXI H.TABLE ; TO ADDR ZERO LOCN OF ARRAY</td>
</tr>
<tr>
<td>47+</td>
<td></td>
<td></td>
<td>CALL ARRAY ; CALL ARRAY SUBRBTN</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>HLT</td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
<td>ORG 0140H</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td>ARSET TABLE, 4, 4, 4, 0</td>
</tr>
<tr>
<td>51+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52+</td>
<td></td>
<td></td>
<td>LXI H.(4 SHL 8 XOR 4) AND 0FFFFH</td>
</tr>
<tr>
<td>LOC</td>
<td>OBJ</td>
<td>SEQ</td>
<td>SOURCE STATEMENT</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>------------------</td>
</tr>
<tr>
<td>0143</td>
<td>220610</td>
<td>53+</td>
<td>SHLD PLIST ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>0146</td>
<td>210004</td>
<td>54+</td>
<td>LXI H. (4 SHL 8 XOR 0) AND 0FFFFH</td>
</tr>
<tr>
<td>0149</td>
<td>220810</td>
<td>55+</td>
<td>SHLD PLIST+2 ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>014C</td>
<td>210020</td>
<td>56+</td>
<td>LXI H.TABLE ; TO ADDR ZERO LOCN OF ARRAY</td>
</tr>
<tr>
<td>014F</td>
<td>CD0002</td>
<td>57+</td>
<td>CALL ARRAY ; CALL ARRAY SUBRTN</td>
</tr>
<tr>
<td>0152</td>
<td>76</td>
<td>58</td>
<td>HLT</td>
</tr>
<tr>
<td>0160</td>
<td></td>
<td>59</td>
<td>ORG 0160H</td>
</tr>
<tr>
<td>0160</td>
<td></td>
<td>60</td>
<td>ARSET TABLE, 4, 4, 0, 0</td>
</tr>
<tr>
<td></td>
<td>61+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0160</td>
<td>210404</td>
<td>62+</td>
<td>LXI H. (4 SHL 8 XOR 4) AND 0FFFFH</td>
</tr>
<tr>
<td>0163</td>
<td>220610</td>
<td>63+</td>
<td>SHLD PLIST ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>0166</td>
<td>210000</td>
<td>64+</td>
<td>LXI H. (0 SHL 8 XOR 0) AND 0FFFFH</td>
</tr>
<tr>
<td>0169</td>
<td>220810</td>
<td>65+</td>
<td>SHLD PLIST+2 ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>016C</td>
<td>210020</td>
<td>66+</td>
<td>LXI H.TABLE ; TO ADDR ZERO LOCN OF ARRAY</td>
</tr>
<tr>
<td>016F</td>
<td>CD0002</td>
<td>67+</td>
<td>CALL ARRAY ; CALL ARRAY SUBRTN</td>
</tr>
<tr>
<td>0172</td>
<td>76</td>
<td>68</td>
<td>HLT</td>
</tr>
<tr>
<td>0180</td>
<td></td>
<td>69</td>
<td>ORG 0180H</td>
</tr>
<tr>
<td>0190</td>
<td></td>
<td>70</td>
<td>ARSET TABLE, 4, 4, 1, 4</td>
</tr>
<tr>
<td></td>
<td>71+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0180</td>
<td>210404</td>
<td>72+</td>
<td>LXI H. (4 SHL 8 XOR 4) AND 0FFFFH</td>
</tr>
<tr>
<td>0183</td>
<td>220610</td>
<td>73+</td>
<td>SHLD PLIST ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>0186</td>
<td>210401</td>
<td>74+</td>
<td>LXI H. (1 SHL 8 XOR 4) AND 0FFFFH</td>
</tr>
<tr>
<td>0189</td>
<td>220810</td>
<td>75+</td>
<td>SHLD PLIST+2 ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>018C</td>
<td>210020</td>
<td>76+</td>
<td>LXI H.TABLE ; TO ADDR ZERO LOCN OF ARRAY</td>
</tr>
<tr>
<td>018F</td>
<td>CD0002</td>
<td>77+</td>
<td>CALL ARRAY ; CALL ARRAY SUBRTN</td>
</tr>
<tr>
<td>0192</td>
<td>76</td>
<td>78</td>
<td>HLT</td>
</tr>
<tr>
<td>01A0</td>
<td></td>
<td>79</td>
<td>ORG 01A0H</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td></td>
<td>ARSET TABLE, 4, 1, 4, 4</td>
</tr>
<tr>
<td></td>
<td>81+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01A0</td>
<td>210104</td>
<td>82+</td>
<td>LXI H. (4 SHL 8 XOR 1) AND 0FFFFH</td>
</tr>
<tr>
<td>01A3</td>
<td>220610</td>
<td>83+</td>
<td>SHLD PLIST ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>01A6</td>
<td>210404</td>
<td>84+</td>
<td>LXI H. (4 SHL 8 XOR 4) AND 0FFFFH</td>
</tr>
<tr>
<td>01A9</td>
<td>220810</td>
<td>85+</td>
<td>SHLD PLIST+2 ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>01AC</td>
<td>210020</td>
<td>86+</td>
<td>LXI H.TABLE ; TO ADDR ZERO LOCN OF ARRAY</td>
</tr>
<tr>
<td>01AF</td>
<td>CD0002</td>
<td>87+</td>
<td>CALL ARRAY ; CALL ARRAY SUBRTN</td>
</tr>
<tr>
<td>01B2</td>
<td>76</td>
<td>88</td>
<td>HLT</td>
</tr>
<tr>
<td>01C0</td>
<td></td>
<td>89</td>
<td>ORG 01C0H</td>
</tr>
<tr>
<td></td>
<td>90</td>
<td></td>
<td>ARSET TABLE, 4, 1, 4, 0</td>
</tr>
<tr>
<td></td>
<td>91+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01C0</td>
<td>210104</td>
<td>92+</td>
<td>LXI H. (4 SHL 8 XOR 1) AND 0FFFFH</td>
</tr>
<tr>
<td>01C3</td>
<td>220610</td>
<td>93+</td>
<td>SHLD PLIST ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>01C6</td>
<td>210004</td>
<td>94+</td>
<td>LXI H. (4 SHL 8 XOR 0) AND 0FFFFH</td>
</tr>
<tr>
<td>01C9</td>
<td>220810</td>
<td>95+</td>
<td>SHLD PLIST+2 ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>01CC</td>
<td>210020</td>
<td>96+</td>
<td>LXI H.TABLE ; TO ADDR ZERO LOCN OF ARRAY</td>
</tr>
<tr>
<td>01CF</td>
<td>CD0002</td>
<td>97+</td>
<td>CALL ARRAY ; CALL ARRAY SUBRTN</td>
</tr>
<tr>
<td>01D2</td>
<td>76</td>
<td>98</td>
<td>HLT</td>
</tr>
<tr>
<td>01E0</td>
<td></td>
<td>99</td>
<td>ORG 01E0H</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td>ARSET TABLE, 4, 1, 1, 4</td>
</tr>
<tr>
<td></td>
<td>101+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01E0</td>
<td>210104</td>
<td>102+</td>
<td>LXI H. (4 SHL 8 XOR 1) AND 0FFFFH</td>
</tr>
<tr>
<td>01E3</td>
<td>220610</td>
<td>103+</td>
<td>SHLD PLIST ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>01E6</td>
<td>210401</td>
<td>104+</td>
<td>LXI H. (1 SHL 8 XOR 4) AND 0FFFFH</td>
</tr>
<tr>
<td>01E9</td>
<td>220810</td>
<td>105+</td>
<td>SHLD PLIST+2 ; STORE ARGS IN PARAMETER LIST</td>
</tr>
<tr>
<td>01EC</td>
<td>210020</td>
<td>106+</td>
<td>LXI H.TABLE ; TO ADDR ZERO LOCN OF ARRAY</td>
</tr>
<tr>
<td>01EF</td>
<td>CD0002</td>
<td>107+</td>
<td>CALL ARRAY ; CALL ARRAY SUBRTN</td>
</tr>
</tbody>
</table>
LOC OBJ  SEQ SOURCE STATEMENT
01F2 76  108  HLT
109 ;
110 ;
111 ;
112 ;
113 ;
114 ;
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 ERRXX ; 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 ERRXX ; INDEX >= SIZE
0214 2A0010 126 LHLDD 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+W>
0220 E1  130 POP H ; RESTORE ZERO ADDR
022F 19  131 DAD D ; INDEX ZERO ADDR
0222 E5  132 PUSH H ; SAVE CURRENT ADDR
0221 3A0910 133 LDA PLIST+3 ; J SIZE TO A REG
022C FE00 134 CPI 0 ; J SIZE = ZERO?
0226 C2E02 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 ERRXX ; INDEX >= SIZE
0236 2A0210 145 LHLDD 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+W>
0240 3A0610 149 LDA PLIST ; I SIZE <ISZ> TO A REG
0243 CD8102 150 CALL INDX ; INDEX TO <J+I+W>*<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 ERRXX ; INDEX >= SIZE
0259 2A0410 161 LHLDD K ; K INDEX TO H.L REGS
025C EB  162 XCHG ; K INDEX TO D.E REGS 5-298
LOC   OBJ  SEQ  SOURCE STATEMENT

025D  3A0710  163  LDA   PLIST+1 ; WORD LENGTH TO A REG
0260  CD8102  164  CALL  INDX ; INDEX TO (K*I#W)
0263  3A0610  165  LDA   PLIST ; I SIZE TO A REG
0266  CD8102  166  CALL  INDX ; INDEX TO ((K*I#W)*ISZ)
0269  3A0910  167  LDA   PLIST+3 ; J SIZE (JSZ) TO A REG
026C  CD8102  168  CALL  INDX ; INDEX TO (((K*I#W)*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 ;
            181 ;
0281  210000  182  INDX:  LXI   H,0 ; ZERO BASE
0284  19    183  INDX1:  DAD   D ; ADD INDEX TO BASE,
0285  3D    184  DCR   A ; NO. OF TIMES SHOWN,
0286  C28402  185  JNZ   INDX1 ; IN A REG
0289  EB    186  XCHG  ; RESULTANT INDEX TO D,E REGS
028A  C9    187  RET   ; RETURN TO CALLER
028B  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
FLOATING POINT CONVERSION ROUTINE (for use with FPAL.LIB)

CONVERT FREE FORM ASCII STRING TO FLOATING POINT NUMBER. CONVERT FLOATING NUMBER TO ASCII STRING WITH FORMAT CONTROL.

STANDARD 8080 OR 8085 HARDWARE

INTEL FPAL.LIB

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*

THE OUTPUT RESULTS CONFORM TO PLM CONVENTION:
A=FP STATUS FIELD
H-L=FP ERROR FIELD
ENCODING LOADS THE FP NUMBER AT THE FP NUMBER ADDRESS. DECODING FILLS THE STRING TO THE LENGTH AND PRECISION SPECIFIED. LEADING ZERO’S ARE SUPPRESSED.
*SEE ATTACHED DOCUMENTATION INCLUDED WITH LISTING*

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>A,D,E,H,L</th>
<th>Programmer:</th>
<th>P. M. Callihan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>23 Bytes</td>
<td>Company:</td>
<td>Goodyear Atomic Corp.</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>587 Bytes</td>
<td>Address:</td>
<td>P. O. Box 628</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>3</td>
<td>City:</td>
<td>Piketon</td>
</tr>
<tr>
<td>Max stack 36-Includes caller&amp;FPAL.LIB</td>
<td></td>
<td>State:</td>
<td>Ohio, 45661</td>
</tr>
</tbody>
</table>

Assembler/Compiler Used: ISIS ASSEMBLER
LISY -- Linear System (Gauss Elimination) - (FPAL)

The routine solves the linear system \([A] x = B\) of order \(N\) with the Gauss elimination method.

None (Math. routine)

8080/8085 Floating point arithmetic library - FPAL
Help : Diagnostic routine called in case of math error.

\([A]\) 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

\(x\) unknown vector = \(N \times 4\) Locations ; symbolic name XV

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>G. DE GRANDI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>12H</td>
<td>COMMISSION OF EUROPEAN COM-</td>
</tr>
<tr>
<td></td>
<td>MUNITIES, JRC EURATOM,</td>
</tr>
<tr>
<td></td>
<td>ELECTRONIC DIVISION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A8H</td>
<td>MUNITIES, JRC EURATOM,</td>
</tr>
<tr>
<td></td>
<td>ELECTRONIC DIVISION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20127 ISPRA (VARESE)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISIS II 8080/8085 MACRO ASSEM.</td>
<td>ITALY</td>
</tr>
</tbody>
</table>
Gamma Function Subroutine

Performs evaluation of the gamma function with the interval (-34, 34) as its domain.

NONE

8008 Floating Point Math Package, Insite User's Library Ref.#BCL, reassembled for the 8080.

Argument of Gamma Function (ARGX) in Floating Point Format (4 bytes).

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
RAM Required: 18 bytes
ROM Required: 494 bytes
Maximum Subroutine Nesting Level: 1 (exclusive of Floating Point Math Package)
Assembler/Compiler Used: 8080 MACRO Assembler, V.2.4

Company: The Univ. of Michigan-Dearborn
Address: 4901 Evergreen Rd.
City: Dearborn
State: Michigan 48128

© Intel Corporation, 1976
PLM MULTIPLE PRECISION ARITHMETIC

Multiple precision twos complement arithmetic package includes addition, subtraction, multiplication, division and convert to decimal. The maximum precision is defined by assembly equates.

8080/8085

REVISED 3/25/79

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.

The package performs the required function and stores the output in the user designated memory locations.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer:</th>
<th>J. Hiley</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>36 Bytes</td>
<td>Company:</td>
<td>Vector International</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>488 Bytes</td>
<td>Address:</td>
<td>Research Park</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>2</td>
<td>City:</td>
<td>B-3044 Haasrode</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>ISIS-II ASM80</td>
<td>State:</td>
<td>Belgium</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1978

8/79

5-307
MATH48

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.

Any MCS-48 processor.

None

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.

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

RAM Required: bytes=4 X precision of data

ROM Required: 96H

Maximum Subroutine Nesting Level: 2

Assembler/Compiler Used: ASM48 V2.1

Programmer: D. Holden
Company: Miltope
Address: 16 Hancock Street
City: Plainville
State: MA 02762
Program Title
Acquisition of a Decimal Number from MDS Console, Conversion to FPAL Floating Point Number and Vice Versa
(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)

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer: G.DeGrandi, N.Coppo</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>5FH</td>
<td>Company: Commission of European</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communities JRC Est. of Ispra</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>D9EH</td>
<td>Address: Electronics Division</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>ISIS-11 8080/8085</td>
<td>City: 21020 Ispra (Varese)</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>Macro Assembler V2.0</td>
<td>State: ITALY</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1978
12/78
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 an 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.
Double Precision Floating Point Package (DFPAL.LIB)

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:

<table>
<thead>
<tr>
<th>Programmer: Larry Brockwell &amp; M. Master</th>
</tr>
</thead>
</table>

|---------------|------------------------------------------------------|

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address: 770 King Edward Ave.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City: Ottawa, Ontario</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State: Canada K1N 6N5</th>
</tr>
</thead>
</table>

© Intel Corporation, 1978
Discrete Fourier Transform

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.

8080 CPU

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 uest Avenue des Pins
Maximum Subroutine Nesting Level: due to FPP
City: Montreal, Quebec
Assembler/Compiler Used: 16K ISIS Assembler Version 1.1
State: Canada H2W 1R7
FCONST -- Floating Point Constant Calculator

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 assembler DB statements.

MDS, Console, Floppy Disk System

ISIS-II. Program is located above ISIS and is called by entering FCONST

Fixed point or scientific notation numbers, entered on console e.g. 1.09, 10.5E-20

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 OF 2H, 04H, 35H, 3FH

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: J. B. Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required: 3743 BYTES</td>
<td>Company: Plessey Marine</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: Templecombe</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City: Somerset</td>
</tr>
<tr>
<td>Assembler/Compiler Used: PLM80 V3.1</td>
<td>State: U.K.</td>
</tr>
</tbody>
</table>
FPAL86.LIB - FLOATING POINT LIBRARY

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.

An MDS or Series-II for assembling, compiling, linking, locating 8086 based programs. Plus hardware necessary to execute 8086 code.

MDS-311 8086 software support package.

Same as 8080/8085 FPAL.

$50.00 (object code only on diskette)

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>A11</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Total FPAL 86LIB uses 6811 bytes</td>
<td>Company:</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
<td></td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>ASM86, PLM86</td>
<td>State:</td>
</tr>
</tbody>
</table>
BINGRY - BINARY TO GRAY CODE CONVERSION SUBROUTINE

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 \( i = 0, 1, 2, \ldots, 15 \)
- \( G_i \) ...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
/E0 ... 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:

\( \emptyset \)

Address:

Pelhrimovska 339/9

Maximum Subroutine Nesting Level:

\( \emptyset \)

City:

145 00 Praha 4, Michle

Assembler/Compiler Used:

Macro Assembler

State:

Czechoslovakia

V2.Ø
"FLN" - NATURAL LOG OF INTEL FLOATING POINT NUMBER

Takes natural LOG (LN(X)) of number in floating point record (FPR)

MDS 800 with ISIS-II linking loader

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

Pass pointer to 18-byte "FPR" in B,C registers
PL/M80 call: CALL FLN (.FPR)

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 -∞ for LN(0)
4) Does not set error/stat bits for "0" operand

REF.NOS. BC32A,B,C
ORDERED AS ONE
PROGRAM

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer: Bart Evans</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Variables: 28 Stack:2</td>
<td>Company: Durrum Instrument</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Code: 462</td>
<td>Address: 1228 Titan Way</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>1</td>
<td>City: Sunnyvale</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>PLM80, V3.0</td>
<td>State: CA 94086</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1978
"FSQR" - SQUARE ROOT OF INTEL FLOATING POINT NUMBER

Extracts square root of number in floating point record (FPR)

MDS 800 with ISIS-II linking loader

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

Pass pointer to 18-byte "FPR" in B,C registers
PLM80 CALL: CALL FSQR (.FPR)

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer:</th>
<th>Bart Evans</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>VARIABLES: 13 STACK: 4</td>
<td>Company:</td>
<td>Durrum Instrument</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>CODE: 330</td>
<td>Address:</td>
<td>1228 Titan Way</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>1</td>
<td>City:</td>
<td>Sunnyvale</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>PLM80, V3.0</td>
<td>State:</td>
<td>CA 94086</td>
</tr>
</tbody>
</table>
"FEXP" - NATURAL EXPONENT OF INTEL FLOATING POINT NUMBER

Takes natural exponential (e^K) of number in floating point record (FPR)

MDS 800 with ISIS-II linking loader

Intel "FPAL.LIB", Intel "LINK" & "LOCATE", PLM80 Compiler

Pass pointer to 18-byte "FPR" in B,C registers

PLM80 CALL: CALL FEXP (.FPR)

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer: Bart Evans</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>VARIABLES: 25 STACK: 2</td>
<td>Company: Durrum Instruments</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>CODE: 498</td>
<td>Address: 1228 Titan Way</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>1</td>
<td>City: Sunnyvale</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>PLM80, V3.0</td>
<td>State: CA 94086</td>
</tr>
</tbody>
</table>
BINGRY - BINARY TO GRAY CODE CONVERSION SUBROUTINE

Converts binary number to Gray cyclic code. Number can be 16 bits long / unused bits = 0 /.

Algorithm: \( G_i = B_i + B_{i+1} \), where \( i = 0, 1, 2, \ldots, 15 \)
- \( G_i \) ... \( i \)th bit of Gray number
- \( B_i \) ... \( i \)th bit of binary number

Any 8080 Microcomputer

None

Binary number stored in regs. D, E /E0 ... LSB, unused bits = 0 /

Gray cyclic code in D, E

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>A, D, E</th>
<th>Programmer:</th>
<th>Nemec D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>None</td>
<td>Company:</td>
<td>VSE</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>( \emptyset )CH</td>
<td>Address:</td>
<td>Pelhrimovska 339/9</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>( \emptyset )</td>
<td>City:</td>
<td>145 00 Praha 4, Michle</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>Macro Assembler V2.0</td>
<td>State:</td>
<td>Czechoslovakia</td>
</tr>
</tbody>
</table>
MTL DIV - SUBROUTINES

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.

Any 8080 or 8085 CPU

See Listing

See Listing

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>All</th>
<th>Programmer:</th>
<th>Ken Bartlett</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>12_{10} Bytes</td>
<td>Company:</td>
<td>Acurex Corporation</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>259_{10} Bytes</td>
<td>Address:</td>
<td>485 Clyde Avenue</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>1</td>
<td>City:</td>
<td>Mountain View</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>8080/8085 Macro Assembler V2.0</td>
<td>State:</td>
<td>California 94042</td>
</tr>
</tbody>
</table>
SQUARE ROOT FOR MCS-48

This routine generates an 8 bits root of a 16 bits number.

None

None

16 bits binary number in R2, R3.
MSBYTE in R2, LSBYTE in R3.

8 bits root in R6

Registers Modified: ALL
RAM Required: 4 bytes
ROM Required: 96 bytes
Maximum Subroutine Nesting Level: 2
ISIS-II MCS-48/UPA-41 Macro Assembler
Assembler/Compiler Used: V2.0

Programmer: Sušumu Urata
Company: TOA Microcomputer Inc.
Address: 5-61 Nipponbashi, Naniwa
City: Osaka 556
State: Japan
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)

None

None

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.

Registers Modified: CDE, BHL

8 Bytes on stack

98H bytes

1

8080/8085 Macro Assembler, V2.0

F.M. Cady

Dept. of Elec. Engineering

University of Canterbury

Christchurch 1

New Zealand
### SECTION 6

**CROSS PRODUCTS SOFTWARE**

<table>
<thead>
<tr>
<th>REFERENCE NUMBER</th>
<th>PROGRAM</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>CROSS ASSEMBLER FOR PDP-11</td>
<td>6-1</td>
</tr>
<tr>
<td>C5</td>
<td>8008 MACRO DEFINITION SET FOR ASSEMBLY ON PDP-11</td>
<td>6-3</td>
</tr>
<tr>
<td>C8</td>
<td>CROSS ASSEMBLER FOR THE PDP-11</td>
<td>6-5</td>
</tr>
<tr>
<td>C9</td>
<td>CROSS ASSEMBLER FOR NOVA 1200</td>
<td>6-7</td>
</tr>
<tr>
<td>C10</td>
<td>CROSS ASSEMBLER FOR NOVA 1220; IBM 360/40 AND CDC 3300</td>
<td>6-9</td>
</tr>
<tr>
<td>C11</td>
<td>NOVA CROSS ASSEMBLER FOR INTEL 8080</td>
<td>6-11</td>
</tr>
<tr>
<td>C13</td>
<td>8008 CROSS INVERSE ASSEMBLER FOR HP2100</td>
<td>6-13</td>
</tr>
<tr>
<td>C17</td>
<td>PL/M 80 PASS 3</td>
<td>6-21</td>
</tr>
<tr>
<td>C18</td>
<td>CROSS ASSEMBLER FOR VARIAN DATA MACHINE</td>
<td>6-23</td>
</tr>
<tr>
<td>C19</td>
<td>INTEL TO OCTAL CODE CONVERSION FOR PDP-11</td>
<td>6-25</td>
</tr>
<tr>
<td>C20</td>
<td>HEX CONVERT--CONVERT INTEL HEX FORMAT TO PROLOG</td>
<td>6-27</td>
</tr>
<tr>
<td>C21</td>
<td>PDP-11 BINARY FILE TO INTEL HEX FILE CONVERTER</td>
<td>6-31</td>
</tr>
<tr>
<td>C22</td>
<td>PDP-11 PROGRAM LOAD TO HEX, DUMP, VERIFY</td>
<td>6-33</td>
</tr>
<tr>
<td>C23</td>
<td>FORMAT</td>
<td>6-35</td>
</tr>
<tr>
<td>C24</td>
<td>8080 CROSS-ASSEMBLER FOR TEKTRONIX 4051</td>
<td>6-37</td>
</tr>
<tr>
<td>C25</td>
<td>I8080 CROSS ASSEMBLER FOR INTEL 8080/8085 MICROPROCESSORS</td>
<td>6-47</td>
</tr>
<tr>
<td>C26</td>
<td>MACRO ASSEMBLER FOR DG NOVA</td>
<td>6-49</td>
</tr>
<tr>
<td>C27</td>
<td>SYMBOL CROSS-REFERENCE</td>
<td>6-53</td>
</tr>
<tr>
<td>C28</td>
<td>ALIGN PROGRAM - INTERMEDIATE</td>
<td>6-59</td>
</tr>
<tr>
<td>C29</td>
<td>8085 CROSS ASSEMBLER FOR THE DEC PDP8 AND PDP11</td>
<td>6-63</td>
</tr>
<tr>
<td>C30</td>
<td>8080 CROSS ASSEMBLER FOR 8085-MACRO DEFINITION SET--M8008. SRC</td>
<td>6-65</td>
</tr>
<tr>
<td>C31</td>
<td>CROSS ASSEMBLER FOR HONEYWELL H316/516/716</td>
<td>6-69</td>
</tr>
<tr>
<td>C32</td>
<td>CROSS ASSEMBLER FOR INTERPRETIVE MCS-48</td>
<td>6-71</td>
</tr>
<tr>
<td>C33</td>
<td>SIM48 - 8048 SIMULATOR</td>
<td>6-74</td>
</tr>
<tr>
<td>C34</td>
<td>8085 CROSS ASSEMBLER FOR NOVA 1200</td>
<td>6-76</td>
</tr>
<tr>
<td>C35</td>
<td>RTCOPY - COPY FROM PDP-11 DISKETTE (RT-11) TO ISIS-II DISKETTE FILE.</td>
<td>6-78</td>
</tr>
</tbody>
</table>

**** SPECIAL NOTE ****

ALL DISKETTES PROCESSED BY INSITE USERS LIBRARY ARE ISIS-II FORMATTED.
CROSS ASSEMBLER ON PDP-11.

PERFORMS SYMBOLIC ASSEMBLY FOR 8080 ASSEMBLY LANGUAGE PROGRAMS. PROGRAM RUNS ON ANY DEC PDP-11 WHICH SUPPORTS THE MACRO ASSEMBLER.

DEC PDP-11 CAPABLE OF SUPPORTING THE MACRO ASSEMBLER AND 4K WORDS OF MEMORY FOR MACRO DEFINITIONS.

PDP-11 MACRO ASSEMBLER, PLUS SUPPORT SYSTEM; E.G., DOS OR RSX.

ASSEMBLY LANGUAGE PROGRAM FOR THE INTEL 8080.

ASSEMBLED LISTING AND/OR PDP-11 FORMAT BINARY LOAD MODULE.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.A.</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.A.</td>
<td>PDP-11 MACRO ASSEMBLER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.A.</td>
<td>JOHN ANDERSON &amp; WILLIAM GALWAY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIVERSITY OF UTAH DEPARTMENT OF CHEMISTRY</td>
</tr>
</tbody>
</table>
8008 Macro Definition Set
Permits cross assembly of 8008 programs on a PDP-11.

Any PDP-11 with 16K core and the Macro Assembler.

Macro Assembler minimum, and RT-11 operating system optional (instructions assume RT-11).

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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.A.</td>
<td>Programmer:</td>
</tr>
<tr>
<td></td>
<td>Tom Seim</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>N.A.</td>
<td>Battelle-Northwest</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>N.A.</td>
<td>Box 999</td>
</tr>
<tr>
<td></td>
<td>Richland, Wash. 99352</td>
</tr>
</tbody>
</table>
Cross Assembler for the PDP-11

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.

Digital Equipment Corp. PDP 11 with RSTS BASIC-plus.
   —could be modified to operate under any extended BASIC

An editor for preparation of the source file

ASCII source file

Hexadecimal file—if punched on paper-tape, suitable for loading into the microprocessor memory via loader

Listing file

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>RSTS BASIC-plus</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>N/A</td>
<td>G.D. Young</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>N/A</td>
<td>Aanderaa Instruments Ltd.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>N/A</td>
<td>560 Alpha Street</td>
</tr>
<tr>
<td></td>
<td>Victoria, B.C., Canada  V8Z 1B2</td>
</tr>
</tbody>
</table>
Cross Assembler for Nova 1200

To serve as a cross assembler enabling the user to assemble programs written for the 8080 or 8080 microcomputers, without using a time sharing service.

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.

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 file name, Intermediate file name, listing options

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: Paul Mennen</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: Sierra Research</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: 24/ Cayuga Road</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City: Cheektowaga, New York</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
</tbody>
</table>
CROSS ASSEMBLER FOR NOVA 1220, IBM 360/40 and CDC 3300

The CPI 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.

A principal objective in the development of the assembler was transportability. To achieve this, it is written in a subset of ANSI Standard Fortran that allows it to be compiled 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.

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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: Bernard Evans, Ph.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: Cal Poly</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: Computer Sciences Dept</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City: San Luis Obispo</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State: California 93407</td>
</tr>
</tbody>
</table>
NOVA Cross Assembler for INTEL 8080

To assemble programs written in the INTEL 8080 language and output a listing and a paper tape of the assembly.

DATA GENERAL NOVA minicomputer with 8K or larger memory and an SOS or better operating system; teletype and line printer.

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).

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.

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.
Intel 8008 Cross Inverse Assembler for HP-2100

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.

Hewlett-Packard 2100 family computer, 4K, Teleprinter, Photoreader.

HP Assembler for assembly.
Program is self-contained.

BNPF object code (see attached sheet)

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.
PLM 83 (PLM/80 PASS 3)

Inter-lists PLM source lines with the resulting assembly language and hex code.

None

PLM 81 & PLM 82 (PLM/80 PASS 1&2)

PLM/80 pass 1 listing, PLM/80 pass 2 listing & Hex file

1) inter-listed file

2) Alphabetical symbol table

Sample input & output files for Intel Square Root Program attached.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.A.</td>
<td>Fortran (IBM/360, SIGMA/9, PDP/10)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.A.</td>
<td>B. Searle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.A.</td>
<td>Ministry of Transport (CANADA)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N.A.</td>
<td>TACD, Tower C, Place de Ville</td>
</tr>
<tr>
<td></td>
<td>OTTAWA, ONTARIO, CANADA K1A 0N0</td>
</tr>
</tbody>
</table>


**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

---

<table>
<thead>
<tr>
<th>Registors Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DASMR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WALTER HEIL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GESELLSCHAFT FÜR KERNFORSCHUNG/ADI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7500 KARLSRUHE</td>
</tr>
<tr>
<td></td>
<td>WEST GERMANY</td>
</tr>
<tr>
<td></td>
<td>P.B. 3640</td>
</tr>
</tbody>
</table>
Intel to Octal Number Conversion (OCT)

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.

PDP-11 computer with disk

RT-11
Fortran IV compiler
System Library Subroutines

INPUT FILE* (enter name of file with Intel numbers)
OUTPUT FILE* (enter name of file to contain octal number source code)

An output file will be created which will contain the octal number source code

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PDP-11 Fortran IV</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jim Ahern</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard Memories</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2221 So. Anne St.</td>
</tr>
<tr>
<td></td>
<td>Santa Ana, Ca. 92704</td>
</tr>
</tbody>
</table>
Hexcnvrt  (Hex Convert)

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.

Time sharing system or other computer system with capability of running Intel cross product software.

Fortran IV
(Source program may have to be modified for other compilers).

"LOGBIN" File produced by Intel Macro assembler (see attached sheets).

"Prolog" File  (see attached sheets).

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>GE Mark III Timesharing</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Joseph Parchesky</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>Datatype Corporation</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Miami, Florida 33169</td>
</tr>
</tbody>
</table>
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 4044/4040 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 TELETYPING 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','C','D','E','F'/
340   END
350*
360*
370*MAIN PROGRAM
380   ALPHA CHAR(24), PAGE,A(16),B(16),C
390   DATA C/O01500000000/
400   PAGE=""
410*INPUT LINE FROM LOGBIN.FILE
420   420 READ("LOGBIN"), 430) CHAR
430   430 FORMAT(A2,A2,A2,A2,A2,A2,A2,A2,A2)
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 DO 630 I=1,J1
590 J3=J2+1-I
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 850 I=1,16
820 IF(G.eq.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)
990 990 Y=HEXA(1)
1000 RETURN
1010 1010 X=HEXA(N-1)
1020 1020 Y=HEXA((J+17)-(N-1)*16)
1030 RETURN
1040 END
PDP-11 binary file to INTEL HEX file converter.

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.

PDP-11 with an operating system
disk, paper tape reader/punch

a set of macro definitions defining the 8080 instruction set
(X8-2) is a possible set
PDP-11 FORTRAN and MACRO

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

INTEL HEX file for use with INTELLEC or INTERP/80
PDP-11 Program to load Intel Hex programs and dump and verify the programs in binary format suitable for PROM programming.

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.

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)

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.

LOADHX With Intel Hex tape in reader

<table>
<thead>
<tr>
<th>SR</th>
<th>Press</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000200</td>
<td>ADDRESS LOAD</td>
<td>Starting address</td>
</tr>
<tr>
<td>XXXXXX</td>
<td>START</td>
<td>Buffer offset in SR</td>
</tr>
</tbody>
</table>

DMPBIN Punch out binary tape

<table>
<thead>
<tr>
<th>SR</th>
<th>Press</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000500</td>
<td>ADDRESS LOAD</td>
<td>Starting address</td>
</tr>
<tr>
<td>XXXXXX</td>
<td>START</td>
<td>Buffer offset in SR</td>
</tr>
</tbody>
</table>

When the computer halts
YYYEEE CONT No. of bytes to be punched

VERIFY With binary tape in the reader

<table>
<thead>
<tr>
<th>SR</th>
<th>Press</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000600</td>
<td>ADDRESS LOAD</td>
<td>Starting address</td>
</tr>
<tr>
<td>XXXXXX</td>
<td>START</td>
<td>Buffer offset in SR</td>
</tr>
</tbody>
</table>

When the computer halts
YYYEEE CONT No. of bytes to be verified

Halts at the following locations are faults:-

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>REASON</th>
</tr>
</thead>
<tbody>
<tr>
<td>246</td>
<td>Checksum error in reading Intel Hex code</td>
</tr>
<tr>
<td>652</td>
<td>Verification error</td>
</tr>
<tr>
<td>446</td>
<td>Reader error, taut tape or no tape</td>
</tr>
<tr>
<td>472</td>
<td>Punch error, punch no ON or out of tape</td>
</tr>
</tbody>
</table>

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 6L U.K.
FORMAT

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)

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

Company:

Gesellschaft für Kernforschung/ IDT

Address:

P.O.B. 3640

City:

7500 KARLSRUHE

State:

West Germany
8080 CROSS-ASSEMBLER FOR TEKTRONIX 4051

This program assembler Intel 8080 source code producing a listing and (optionally) an object file.

Tektronix 4051 Graphic System with 16K or larger memory.
Tektronix 4924 tape unit (optional)

See Attachment

Source Listing available only. Program not available on Paper Tape or Diskette.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bruce W. Bomar</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARO, Inc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arnold AF Station</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended BASIC</td>
<td>TN 37389</td>
</tr>
</tbody>
</table>
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 display. (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:

<table>
<thead>
<tr>
<th>Columns 1 - 6</th>
<th>Symbol followed by &quot;;&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column 7</td>
<td>Space</td>
</tr>
<tr>
<td>Columns 8 - 11</td>
<td>Instruction mnemonic</td>
</tr>
<tr>
<td>Column 12</td>
<td>Space</td>
</tr>
<tr>
<td>Columns 13 -</td>
<td>Operand</td>
</tr>
</tbody>
</table>

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 dimension 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 8080 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", "$-10"; and "$-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 105, 155, 160, 445, 545, 550, 565, 570,
600, 990 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 4951. 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 4951 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.
I$ = 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 100 through 175 request input and output files, dimension variables, and set up register and mnemonic lookup tables.

Pass 1 (lines 180 through 420) 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 990 through 1180 print a hexadecimal code table by address in systems where a 4924 is used.

The subroutine beginning at line 1205 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 = 2    8Ø8Ø instruction,
         M determines the type
         of instruction in order
         of its occurrence within
         data statements which
         start at line 223Ø.
C = 3    DW
C = 4    DB
C = 5    DC
C = 6    DS
C = 7    ORG
C = 8    EQU
C = 9    END
C = 1Ø  Comment

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 1910 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 2105 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.
18080 Cross Assembler for INTEL 8080/8085 Microprocessors.

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.

Computer: Hewlett-Packard 2100 series.
Memory size: 32K-words.
Operating system: RTE-2/RTE-3

HP-ALGOL For the Hewlett-Packard 2000 computer series.

Source language program stored on a disc file.

1. Listing of source code.
2. Symbol table.
3. Listing of generated object code.
4. Possible error messages.
5. Hexadecimal disc file.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: Lara Thrane</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: Electromagnetics Institute</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: Technical University, Bldg 348</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City: DK-2800 Lyngby</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State: Denmark</td>
</tr>
</tbody>
</table>
MACRO ASSEMBLER FOR DG NOVA

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.

DG NOVA or ECLIPSE with at least 16K and a disk operating system.

RDOS or equivalent operating system able to run the NOVA macro assembler.

Source code in NOVA assembly format using 8080 mnemonics and register definitions.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG_NOVA_MACROAssembler</td>
<td>Tom Rust</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Computer Tools</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>205 W. Eureka</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Champaign</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DG_NOVA_MACROAssembler</td>
<td>Illinois 61820</td>
</tr>
</tbody>
</table>
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 modular code to be generated and separately 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
**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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Eugene Bidwell</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Supreme Egpt. &amp; Systems</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>170 53rd Street</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Brooklyn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP-11 Assembler</td>
<td>New York 11232</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976

Ref.#C27
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

START
mount the output paper tape of the 8080 assembler
execute the program from location 1000
program reads the symbol table to $0
write the symbol table on disk
Message on CRT
Mount source tape
Press continue
operator mounts the first source tape.
operator presses Cont on PDP-11
operator mounts next source tapes. "Momentary press of tape - feed may be required"
symbol reference locations written to disk
the END instruction is read on the last source tape
the X-REF. listing is printed from the disk store.
STOP
Align Program - Intermediate Pass between PIM Pass 1 and 2.

Generate consistent storage assignment for global variables in separately compiled program segments.

Computer that supports the PIM Cross Compiler and FORTRAN IV

PIM Cross Compiler, FORTRAN Compiler

Origin (within RAM bank specified by $V in Pass 2) via FORTRAN Unit 9 (decimal, free format)

PIM intermediate File 21 as generated by Pass 1, via FORTRAN Unit 23.

Message Output via FORTRAN Unit 10 (Conversational Terminal Output File)

PIM 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 PIM Cross Compiler.

Program available in source listing only.
The ALIGN program is an intermediate pass between passes One and Two of the PIM 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 PIM 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 PIM 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 PIM 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.
8085 Cross Assembler for the DEC PDP8 and PDP11

Utilizing the file access features of the PDP11 under RTII or the PDP8 under OS8, assembly of programs in standard 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.

DEC PDP8 capable of running OS8 and Fortran II or DEC PDP11 capable of running RTII and Fortran IV (16K words of memory in either machine is enough memory for 300 symbols in the symbol table)

OS8 or RTII and at least Fortran II (although specifically designed for these machines/operating systems, conversion to other machines would be straightforward)

Source file

Listing of assembled program with errors also printed on the console device. And a symbol table.

Intel compatible HEX object file

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: Rex Tracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: Colorado State University</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: Elec. Engr. Dept. - CSU</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City: Ft. Collins</td>
</tr>
<tr>
<td>Assembler/Compiler Used: DEC Fortran II(pdp8)</td>
<td>State: Colorado, 80523</td>
</tr>
<tr>
<td></td>
<td>Fortran II(pdp11)</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1978
8008 Cross Assembler for 8085 - MACRO Definition Set -- M8008.SRC

Permits assembly of programs written in 8008 assembly language with an 8080 Macro Assembler.

ISIS II System

8080/8085 Macro Assembler

A source program written in 8008 assembly language with only slight modifications which are specifically detailed in the source listing of M8008.SRC.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>H. Webster</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company: Bedford Computer Systems</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address: 3 Preston Court</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>City: Bedford</td>
</tr>
<tr>
<td>ISIS-II 8080/8085 Macro Ass.,V2.0</td>
<td>State: Massachusetts 01730</td>
</tr>
</tbody>
</table>
LPSTPR Location Included Post Assembly Processor

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.

ISIS II System

8080/8085 Macro Assembler

List file of an assembled source file written in 8008 assembly language and including at start M8008.SRC, the macro definition set.

Very readable object listing on the lineprinter

C30B is offered as one program with C30A.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>H. Webster</td>
</tr>
<tr>
<td>0389H Bytes</td>
<td>Company:</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Bedford Computer Systems</td>
</tr>
<tr>
<td>0C24H Bytes</td>
<td>Address:</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>3 Preston Court</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>City:</td>
</tr>
<tr>
<td>ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0</td>
<td>State:</td>
</tr>
<tr>
<td></td>
<td>Massachusetts 01730</td>
</tr>
</tbody>
</table>
CROSS ASSEMBLER USING HONEYWELL H316/516/716 (Rev.B)

To generate full program listing and object tape from a source tape written in Intel 8080/8085 Assembly Language.

Honeywell Mini H316/516/716 4K minimum.
Paper Tape Reader, Punch and Teletype.
Line Printer is optional.

Honeywell DAP-16 Assembler

Source Tape written in Intel 8080/8085 Assembly Language using ASCII code forced 8 parity (Honeywell Standard) or Even parity

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
MCS-48 Interpretive Cross Assembler.

MCS-48 Interpretive Cross Assembler that runs on an Intellec 8/MOD80.

Intellec 8/MOD80.

Teletype.

Intellec system monitor Vers. 3.0

Starting address of the program to be assembled.

Assembly language program.

Assembled program in RAM at address specified.

Complete listing of address, machine code, and assembly language mnemonic for each instruction.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer:</th>
<th>M.A. PORDES</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>11 Bytes + Stack</td>
<td>Company: GEC Hirst Research Centre</td>
<td></td>
</tr>
<tr>
<td>ROM Required:</td>
<td>2412 Bytes</td>
<td>Address: East Lane, Wembley, Middx.</td>
<td></td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>8</td>
<td>City: London</td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>MDS 8080/8085 ASSEMBLER V2.0</td>
<td>State: ENGLAND</td>
<td></td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
8/78
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 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 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.
SIM48 VERSION 1.3-8048 SIMULATOR

Simulates an 8048/49 microprocessor with 8243 I/O expander.

8080 Intellec Microcomputer Development System
32K bytes of RAM memory
Flexible diskette drive and controller
Console device

ISIS-II Operating System
ISIS-II 8048 Assembler

Hexadecimal coded file containing the 8048 machine instructions.

Interactive simulation.

Available on diskette only

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A11</td>
<td>E. L. Jones</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>OA27H</td>
<td>Wits. University</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>219BH</td>
<td>1 Jan Smuts Ave.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td>3</td>
<td>Johannesburg</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
<tr>
<td>8080 Assembler PL/M-80 Compiler</td>
<td>South Africa</td>
</tr>
</tbody>
</table>
8085 CROSS ASSEMBLER FOR NOVA 1200

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.

NOVA 1200, 4047 single or dual disk drives and 16K words of memory

RDOS 5 and FORTRAN IV

Source file name

(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

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: Page Communications Engineers, Inc</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: 801 Follin Lane</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City: Vienna</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State: Virginia 22180</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1978
RTCOPY

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 0. 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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>N/A</th>
<th>Programmer:</th>
<th>Steve Freeman</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Less than 1K</td>
<td>Company:</td>
<td>Amer-O-Matic Corp.</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>None</td>
<td>Address:</td>
<td>804-4th Avenue N.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>N/A</td>
<td>City:</td>
<td>Birmingham</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>8080/8085 Macro Assembler V2.0</td>
<td>State:</td>
<td>Alabama</td>
</tr>
<tr>
<td>REFERENCE</td>
<td>PROGRAM</td>
<td>PAGE</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td>QUICKSORT PROCEDURES</td>
<td>7-1</td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>BINARY SEARCH ROUTINE</td>
<td>7-5</td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>DECREMENT H AND L REGISTERS</td>
<td>7-9</td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>MORSE CODE GENERATOR</td>
<td>7-12</td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>CONTROL DATA OUTPUT</td>
<td>7-14</td>
<td></td>
</tr>
<tr>
<td>D12</td>
<td>CLOCK SUBROUTINE</td>
<td>7-18</td>
<td></td>
</tr>
<tr>
<td>D13</td>
<td>INTERRUPT DRIVEN CLOCK ROUTINE</td>
<td>7-22</td>
<td></td>
</tr>
<tr>
<td>D14</td>
<td>CALENDAR SUBROUTINE</td>
<td>7-27</td>
<td></td>
</tr>
<tr>
<td>D15</td>
<td>PASS - PARAMETER PASSING ROUTINE</td>
<td>7-31</td>
<td></td>
</tr>
<tr>
<td>D17</td>
<td>DATA ARRAY MOVE</td>
<td>7-35</td>
<td></td>
</tr>
<tr>
<td>D18</td>
<td>SHELLSORTING ROUTINE</td>
<td>7-39</td>
<td></td>
</tr>
<tr>
<td>D19</td>
<td>TEXT STORAGE PROGRAM</td>
<td>7-42</td>
<td></td>
</tr>
<tr>
<td>D20</td>
<td>TIME SHARING COMMUNICATIONS</td>
<td>7-47</td>
<td></td>
</tr>
<tr>
<td>D21</td>
<td>A GENERALIZED STEPPER MOTOR DRIVE PROGRAM</td>
<td>7-53</td>
<td></td>
</tr>
<tr>
<td>D22</td>
<td>IBM SELECTRIC OUTPUT PROGRAM</td>
<td>7-55</td>
<td></td>
</tr>
<tr>
<td>D23</td>
<td>BINARY SEARCH</td>
<td>7-60</td>
<td></td>
</tr>
<tr>
<td>D24</td>
<td>TIMT - INTERRUPT DRIVEN REAL-TIME CLOCK ROUTINE</td>
<td>7-65</td>
<td></td>
</tr>
<tr>
<td>D25</td>
<td>ABSORBANCE CALCULATION</td>
<td>7-71</td>
<td></td>
</tr>
<tr>
<td>D26</td>
<td>TELEPROCESSING BUFFER ROUTINE</td>
<td>7-73</td>
<td></td>
</tr>
<tr>
<td>D28</td>
<td>RUN O.</td>
<td>7-78</td>
<td></td>
</tr>
<tr>
<td>D30</td>
<td>PAPER TAPE LEADER I. D.</td>
<td>7-81</td>
<td></td>
</tr>
<tr>
<td>D31</td>
<td>LSORT.</td>
<td>7-85</td>
<td></td>
</tr>
<tr>
<td>D33</td>
<td>OCTHEX</td>
<td>7-91</td>
<td></td>
</tr>
<tr>
<td>D34</td>
<td>SDK-80 PAPER TAPE PUNCH ROUTINE</td>
<td>7-93</td>
<td></td>
</tr>
<tr>
<td>D35</td>
<td>TYPE K. T. C. LINEARIZER</td>
<td>7-97</td>
<td></td>
</tr>
<tr>
<td>D36</td>
<td>ENABLE HOLD - SCREEN MODE</td>
<td>7-101</td>
<td></td>
</tr>
<tr>
<td>D37</td>
<td>DISABLE HOLD - SCREEN MODE</td>
<td>7-104</td>
<td></td>
</tr>
<tr>
<td>D38</td>
<td>THERMOCOUPLE LINEARIZATION (TYPE J)</td>
<td>7-108</td>
<td></td>
</tr>
<tr>
<td>D39</td>
<td>INVERT DATA IN RAM</td>
<td>7-110</td>
<td></td>
</tr>
<tr>
<td>D40</td>
<td>RELATIVE JUMP ROUTINE</td>
<td>7-114</td>
<td></td>
</tr>
<tr>
<td>D41</td>
<td>JULIAN DATE ROUTINE</td>
<td>7-118</td>
<td></td>
</tr>
<tr>
<td>D42</td>
<td>TERMINET 1200.</td>
<td>7-122</td>
<td></td>
</tr>
<tr>
<td>D43</td>
<td>TERMINET 300.</td>
<td>7-124</td>
<td></td>
</tr>
<tr>
<td>D44</td>
<td>SETS HORIZONTAL TABS ON TERMINET</td>
<td>7-130</td>
<td></td>
</tr>
<tr>
<td>D45</td>
<td>GRAPH.</td>
<td>7-134</td>
<td></td>
</tr>
<tr>
<td>D46</td>
<td>ANALOG-DIGITAL POLLING ROUTINE</td>
<td>7-140</td>
<td></td>
</tr>
<tr>
<td>D47</td>
<td>THUMBWHEEL SBC 80/10 TEST PROGRAM</td>
<td>7-144</td>
<td></td>
</tr>
<tr>
<td>D48</td>
<td>CALCULATE A CALENDAR FOR ANY YEAR</td>
<td>7-152</td>
<td></td>
</tr>
<tr>
<td>D49</td>
<td>STRING MANIPULATION PACKAGE</td>
<td>7-154</td>
<td></td>
</tr>
<tr>
<td>D50</td>
<td>OUTPUT MESSAGE GENERATOR</td>
<td>7-156</td>
<td></td>
</tr>
</tbody>
</table>
**Program Title**
QUICKSORT PROGRAM

**Function**
SORT AN ARRAY INTO ASCENDING ORDER

**Required Hardware**
8088/8086 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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>2</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>546 bytes</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>368 bytes</td>
</tr>
</tbody>
</table>

**Assembler/Compiler Used:**
PLM80 or PLM

**Programmer:**
KEN BURGETT

**Company:**
00001 1 /* REF. NO. D1 */
00002 1 /* PROGRAM TITLE QUICKSORT */
00003 1
00004 1 /* QUICKSORT PROCEDURE. */
00005 1
00006 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 QUICKSORT:
00025 1
00026 1 PROCEDURE (ARRAY,ARRAY$SIZE);*
00027 1 DECLARE STACK$SIZE LITERALLY '256';
00028 1 DECLARE TRUE LITERALLY 'OFFH', FALSE LITERALLY 'O';
00029 1 DECLARE ARRAY ADDRESS;
00030 1 DECLARE ARRAY$SIZE BYTE;
00031 1 DECLARE LIST BASED ARRAY BYTE;
00032 1 DECLARE LSTACK(STACK$SIZE) BYTE, HSTACK(STACK$SIZE) BYTE;
00033 1 DECLARE TO 8 BYTE;
00034 1 DECLARE (LOW,DPTR,UPTR,HIGH) BYTE;
00035 1 DECLARE (REF, TEMP,DECREMENTING) BYTE;
00036 1
00037 1 PUSH:
00038 1 PROCEDURE (A1,A2);
00039 1 DECLARE (A1,A2) BYTE;
00040 1
00041 1 LSTACK(TOP) = A1;
00042 1 HSTACK(TOP) = A2;
00043 1 TOP = TOP + 1;
00044 1 END PUSH;
00045 1
00046 1 /* MAIN PROGRAM */
00047 1
00048 1 TOP = 0;
00049 1 CALL PUSH(0,ARRAY$SIZE);
00050 1 DO WHILE TOP <> 0;
00051 1 TOP = TOP - 1;
00052 1 IF (DPTR:=(LOW:=LSTACK(TOP)))>(UPTR:=(HIGH:=HSTACK(TOP))) THEN
00053 1 DO;
00054 1 REF = LIST(LOW);
00055 1 DECREMENTING = TRUE;
00056 1 DO WHILE DECREMENTING;
00057 1 DO WHILE LIST(DPTR) <= REF AND HIGH > DPTR;
00058 1 DPTR = DPTR + 1;
00059 1 END;
00060 1 DO WHILE LIST(UPTR) >= REF AND LOW < UPTR;
UPTR = UPTR - 1;
END;
IF DPTR < UPTR THEN
DO;
TEMP = LIST(UPTR);
LIST(UPTH) = LIST(DPTR);
LIST(DPTR) = TEMP;
DPTR = DPTR + 1;
UPTR = UPTR - 1;
END;
ELSE
DO;
IF UPTR > LOW THEN
DO;
LIST(LOW) = LIST(UPTR);
LIST(UPTR) = REF;
UPTR = UPTR - 1;
END;
CALL PUSH(LOW,UPTR);
CALL PUSH(DPTR,HIGH);
DECREMENTING = FALSE;
END;
END;
END QUICKSORT;
/* BEGIN TEST DRIVER */
DECLARE TESTSARRAY (16) BYTE INITIAL
(0,15,1,14,2,13,3,12,4,11,5,10,6,9,7,8);
CALL QUICKSORT(.TESTSARRAY,LAST(TESTSARRAY));
NO PROGRAM ERRORS
BINARY SEARCH ROUTINE

Uses a binary search method to find a character in a table of characters.

None.

Revised 6/78

None.

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.

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, D, H, L</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>MAC 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>45_10</td>
<td></td>
</tr>
</tbody>
</table>

| 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 CHAR.
;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
LOOP:

0003 81
0004 1F
0005 47
0006 85
0007 D2000
000A 24

0008 6F
NCAR1:

000C 7E
000D BB
000E DA1800
0011 CA2A00
0014 50
0015 C31900

0018 48
LOW:

0019 7D
001A 90
001B D21F00
001E 25

NCAR2:

001F 6F
0020 7A
0021 91
0022 FE01
ISR-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 CHAR
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,0 ; 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,A ; ADD ADDRESS OF START OF TABLE
0007 D2000 28 JNC   NCAR1 ; NO CARRY TO SKIP H
0008 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 CA2000 36 JZ    MATCH ; YES, MATCH HAS BEEN FOUND
0014 50    37 MOV   D,B ; NO, SOUGHT CHARACTER IS LESS
0015 C3100 38 JMP   CHECK
39 LOWER:
0018 48    40 MOV   C,B ; CURRENT INDEX BECOMES LOWER IND
41 CHECK:
0019 70    42 MOV   A,L ; RESET H AND L TO START
001A 99    43 SUB   B ; OF TABLE
001B D2100 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 REPEAT
52 INVAL:
INVAL: INZ LOOP :DIFFERENCE MORE THAN 1 SO REPEAT SEARCH

MVI A,1 RET :RETURN WITH 0 AS UNSUCCESSFUL SEARCH

MATCH: MVI A,1 RET :RETURN WITH 1 AS A SUCCESS

END
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>SEQ</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0027</td>
<td>3E00</td>
<td>53</td>
<td>MVI A, 0</td>
</tr>
<tr>
<td>0029</td>
<td>C9</td>
<td>54</td>
<td>RET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55</td>
<td>MATCH:</td>
</tr>
<tr>
<td>002A</td>
<td>3E01</td>
<td>56</td>
<td>MVI A, 1</td>
</tr>
<tr>
<td>002C</td>
<td>0600</td>
<td>57</td>
<td>ADI 0</td>
</tr>
<tr>
<td>002E</td>
<td>C9</td>
<td>58</td>
<td>RET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>59</td>
<td>END</td>
</tr>
</tbody>
</table>

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
DECHL

Macro for decrementing the 16-bit binary contents of the H and L registers.

None.

None.

H and L registers.

(HL-1) + HL

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>H,L</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>MAC 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 bytes</td>
<td>John M. Schulein</td>
</tr>
<tr>
<td></td>
<td>Aeronutronic Ford</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3939 Fabian Way</td>
</tr>
<tr>
<td>Palo Alto, Ca. 94303</td>
</tr>
</tbody>
</table>
; 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
  >08 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
Morse Code

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 time for all good men to.

5v relay around 20-30ma. pull in to operate code oscillator or
transmitter.

None

tty 33 to intellec 8 I/O board UART.

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, D, E, H, L</td>
<td>2</td>
</tr>
<tr>
<td>RAM Required:</td>
<td></td>
</tr>
<tr>
<td>1k</td>
<td></td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Assembler/Compiler Used:</td>
</tr>
<tr>
<td></td>
<td>Intellec 8/Mod 8 Macro</td>
</tr>
<tr>
<td></td>
<td>Assembler, Ver 1.0</td>
</tr>
<tr>
<td></td>
<td>Programmer:</td>
</tr>
<tr>
<td></td>
<td>George B. McConnell</td>
</tr>
<tr>
<td></td>
<td>Hunter Labs</td>
</tr>
<tr>
<td></td>
<td>Company:</td>
</tr>
<tr>
<td></td>
<td>9529 Lee Highway</td>
</tr>
<tr>
<td></td>
<td>Fairfax, Va. 22030</td>
</tr>
</tbody>
</table>
CMDST (CMDS2)

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.

PARALLEL I/O INTERFACE IS USED IN CONJUNCTION WITH THIS
ROUTINE, BUT IS NOT REQUIRED TO RUN IT.

NONE

AS STATED

AS STATED

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,B (A,B,C)</td>
<td>PDP-10 MACRO ASSEMBLER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ONE WORD (TWO WORDS)</td>
<td>STANLEY J. JACZYNSKI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 WORDS (27 WORDS)</td>
<td>EXTRION CORPORATION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNLIMITED</td>
<td>BOX 1226</td>
</tr>
<tr>
<td></td>
<td>BLACKBURN INDUSTRIAL PARK</td>
</tr>
<tr>
<td></td>
<td>GLOUCESTER, MASS 01930</td>
</tr>
</tbody>
</table>
;REGF. NO. D5
;PROGRAM NAME CMDST (CMDST2)
;
;
;
;
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 POINTING 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 CA0000 JZ ZSET ;YES, DO AN "AND"
0006 2F CMA ;COMPLEMENTMENT MASK
0007 B6 ORA M ;"OR" WITH COMMAND WWD
0008 C3000 JMP MSET
000B A6 ZSET: ANA M ;"AND" WITH COMMAND WD
000C 77 MSET: MOV M,A ;RESTORE MEMORY
000D C9 RET

TITLE ::* CMDST2 *
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 MSB IN NEXT LOC.
on RETURN, BC AND ADDRESS HL, HL+1
CONTAIN THE NEW COMMAND WORD

000E FE00 CMDST2: CPI 0 ;IS THE BIT TO BE 0
0010 79 MOV A,C
0011 CA1F00 JZ ZSET2 ;YES, WELL GO !"AND"
0014 2F CMA ;COMPLEMENT MASK
0015 B6 ORA N ;"OR" WITH COMMAND LSD
0016 4F MOV C,A ;STORE NEW LSD IN C
0017 78 MOV A,B ;GET MASK MSB
0018 2F CMA ;COMPLEMENT 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    DCK    H    ;WORD IN RAM
0027 71    MOV    M,C    
0028 C9    RET    
0000    END    

7-17
Clock Subroutine

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.

Clock Routine: No specific equipment is designated.
Test Program: Intellec 8/Mod 80 with TTY connected to the console output port.

Clock Routine: None.
Test Program: Monitor, Version 3.0 installed in Intellec.

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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td>8080 Macro Assembler, V3.0</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>3 bytes</td>
<td>M. M. Dodd</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>52 bytes</td>
<td>Telcom, Inc.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address: 8027</td>
</tr>
<tr>
<td>One (the subroutine itself)</td>
<td>VA 22180</td>
</tr>
</tbody>
</table>
; 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
000C 6A0 ADI (NOT 60H)+1 ; RESET SECS TO 0
000E 323400 STA TIME ; ALSO SET CARRY
0011 3A3500 LDA TIME+1 ; STORE NEW SECS
0014 CE00 ACI 0 ; GET MINUTES
0016 27 DAA ; INCORRECT MINUTES
0017 323500 STA TIME+1 ; ADJUST IT
001A FE60 CPI 60H ; STORE MINS
001C C0 RNZ ; NOT 60 - RETURN
001D 6A0 ADI (NOT 60H)+1 ; RESET MINS TO 0
001F 323500 STA TIME+1 ; ALSO SET CARRY
0022 3A3600 LDA TIME+2 ; STORE NEW MINS
0025 CE00 ACI 0 ; GET HOURS
0027 27 DAA ; INCORRECT HOURS
0028 323600 STA TIME+2 ; ADJUST IT
002B FE24 CPI 24H ; STORE HOURS
002C CPI 12H ; CHECK FOR 24 HOURS
002D C0 RNZ ; CHECK FOR 12 HOURS - REMOVE
002E C6DC ADI (NOT 24H)+1 ; SEMICOLON TO ACTIVATE THIS
002F ; INSTRUCTION AND DELETE
0030 ; ABOVE INSTRUCTION FOR 12-
0032 ; HOUR OPERATION
0032 ADI (NOT 12H)+1 ; NOT MAX. HOURS - RETURN
0033 ; RESET HOURS TO 0
0034 ; ALSO SET CARRY
0035 ; RESET HOURS TO 0 IF
0036 ; USING 12-HOUR SYSTEM. REMOVE
0037 ; SEMICOLON TO ACTIVATE. DELETE A
0038 ; INSTRUCTION

7-20
0030  B20000  STA  TIME+2   ;STORE NEW HOURS
0032  C8  RET
0034  000000  TIME:  DB  0.0.0
0038  END
Interrupt driven clock routine

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 microseconds.

External clock oscillator or divider to produce interrupt pulses 100 ms apart.

CRT for test of program with the supplied test routine

Intellec 8 Monitor Ver 3.0 for testing of program

All software to utilize information from clock

If required: Preset interval and starting time

Incrementing 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.

Address: P.O.B. 83

2007 Kjeller, NORWAY

Maximum Subroutine Nesting Level:

1

98-0348

7-22
; 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
1000 00
1001 00
1002 00
1003 00
1004 00
1005 0A
1006 00

ORG 1000H
MSC: DB 0
SECC: DB 0
MINC: DB 0
HRC: DB 0
INTC: DB 0
PINT: DB 10
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
MVI A.VALUE
CALL INCOM
ENDM

; SUBROUTINE FOR INCREMENTING AND DIFFERENT COUNTERS AND COMPARE IT TO VALUE. IF NOT EQUAL SKIP TO END OF PROGRAM
;
1007 34
1008 BE
1009 C26610
100C 3600
100E C9

INCOM: INR M
CMP M
JNZ FIN1
MVI M, 0
RET

; MAIN CLOCK ROUTINE
;
100F F5
TIME: PUSH PSW
1010 E5  PUSH   H     ;SAVE REGISTERS
1011 210010 + CNT H: 01000H   ;UPDATE MSC COUNTER
1014 3E0A + MVI A: 00000AH   ;SET COUNTER ADDRESS
1016 CD0710 + CALL INCOM   ;SET MAX COUNT
1019 210010 + LXI H: MSC     ;SET COUNTER ADDRESS
101C 3E0A + MVI A: 00000AH   ;SET MAX COUNT
101E CD0710 + CALL INCOM
1021 210110 + COUNT SECC: 60 ;UPDATE SEC-COUNTER
1024 3E3C + MVI A: 0003CH   ;SET COUNTER ADDRESS
1026 CD0710 + CALL INCOM
1029 210110 + LXI H: SECC   ;SET MAX COUNT
102C 3E3C + MVI A: 0003CH   ;SET COUNTER ADDRESS
102E CD0710 + CALL INCOM
1031 210410 + LXI H: INTC
1034 34 + INR M     ;INCR INTERVAL COUNTER
1035 3A0510 + LDA FINT  ;GET PRESENT INTERVAL
1038 BE + CMP M      ;RESET INTERVAL COUNTER
1039 C24310 + JNZ LAB1 ;SET INTERVAL FLAG
103C 3E00 + MVI H: 0001   ;SET INTERVAL COUNTER
103E 3E01 + MVI A: 1
1040 320610 + STA FINT ;UPDATE MINUTE COUNTER
1043 210210 + LXI H: 010002H ;SET COUNTER ADDRESS
1046 3E3C + MVI A: 0003CH ;SET MAX 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
1053 210310 + LXI H: 010003H ;UPDATE HOUR COUNTER
1056 3E18 + MVI A: 00018H ;SET MAX 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 CD6710 + JMP FIN ;DUMMY POP
1066 E1 FIN1: POP H
1067 E1 FIN: POP H ;RESTORE REGISTERS
1068 F1 POP PSW
1069 FB E1
106A C9 RET
0000 END
Calendar Subroutine

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.

Test Program - Intellec 8/Mod 80 with TTY connected to console output port.

Calendar Subroutine - None

Test Program - Monitor V3.0

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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None (all saved)</td>
<td>8080 Macro Assembler V3.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 bytes (including stack)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>109 bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (the subroutine itself)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Donald E. Shorter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telcom, Inc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8027 Leesburg Pike Vienna, Va. 22180</td>
</tr>
</tbody>
</table>
; 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
0003 7E    MOV    A, M
0006 FE31  CPI    31H
0008 DA2400 JC     FIX
000B 3601  CAL1:  MVI    M, 1    ; RESET DAY TO 1
  ^0D 2B    DCX    H    ; TO MONTH
  ^0E 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
003B 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
  4F 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
006C 01  MONTH: DB 1
006E 01  DAY: DB 1
006F 75  YEAR: DB 75H
*00 END
PASS

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 P_n
(Next Instruction)
```

The general form of the subroutine must be:

```
PASS EQU XXXXXH     Starting address of PASS.
IP1 DW 0
IP2 DW 0
.
.
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 ; PARAMETER LIST.
1504 01FCFF LXI B, #4 ; LOAD -4 INTO BC AND ADD TO JHL TO
1505 09 DAD B ; OBTAIN SUBROUTINE ENTRY ADDRESS.
1506 7D MOV A,L ; STORE LOW 8-BITS IN A.
1507 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 PHCHL ; RETURN TO SUBROUTINE.
0000 END
Data Array Move

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.

Any program sequence which stores input parameters into RAM in the format indicated below.

Source array starting address, 2 bytes:
- $<\text{SSA}>$ = Lo order source array starting address
- $<\text{SSA} + 1>$ = Hi order source array starting address

Source array ending address, 2 bytes:
- $<\text{SEA}>$ = Lo order source array ending address
- $<\text{SEA} + 1>$ = Hi order source array ending address

NOTE: Source array ending address must be greater than source array starting address.

Destination array starting address, 2 bytes:
- $<\text{DSA}>$ = Lo order destination array starting address
- $<\text{DSA} + 1>$ = 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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, F, B, C, D, E, H, L</td>
<td>Intellec 8/Mod 80 Ver. 2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bytes</td>
<td>Richard Dérickson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>58 bytes</td>
<td>Logical Services Inc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1901 Old Middlefield Rd., Suite 17 Mountain View, CA 94043</td>
</tr>
</tbody>
</table>
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
0113 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 ; RA = 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 = SOURCE TOP
012A 09 DAD B ; HL = SOURCE TOP
012B EB XCHG ; HL = DEST TOP, DE = DEST TOP
012C CD3701 FWDL: CALL TEST ; DECREMENT ARRAY SIZE
012F C8 RZ ; RETURN WHEN ARRAY SIZE = 0
0130 1A LDAX D ; RA = 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
013A 3EFF MVI A, 0FEH ; A = -2
013B EE CMP C
013C 00 RNZ ; ARRAY SIZE NOT 0, CONTINUE LOOP
013D 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 C44401 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 1100F02 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 111102 LXI D, 0218H ; SEA Assigned 0218H
015D 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
SHELLSORT

SORTS ARRAYS IN PLACE USING SHELL'S METHOD (DIMINISHING INCREMENT)

MCS-80

DRIVER PROGRAM

(ADDRESS OF ARRAY, LENGTH)

ARRAY ELEMENTS ARE SORTED WITH SMALLEST ELEMENT FIRST

Registers Modified:  
Assembler/Compiler Used:  
PL/M 8008/8080

RAM Required:  
12 BYTES

ROM Required:  
approx. 150H BYTES

Maximum Subroutine Nesting Level:  
6

Programmer:  
CHARLES A. PALERMO

Company:  
GENERAL DYNAMICS

Address:  
MZ 2119, Box 748  
FT. WORTH, TEXAS  76101

817/732-4811
/*REF. No. D1a */

/*PROGRAM TITLE SSORT*/

/*DRIVER PROGRAM USED TO VERIFY PROCEDURE.*/

DECLARE H(10) BYTE INITIAL (10, 9, 8, 7, 6, 5, 4,
3, 2, 1);

DECLARE I BYTE, N BYTE INITIAL (10);

SHELLSORT: PROCEDURE (ARRAYPTR,N);

/* THIS PROCEDURE IS USED TO SORT BYTE ARRAYS OF LENGTH N.
RE-DECLARING A AND TEMP AS ADDRESS. ARRAYPTR IS THE
ADDRESS OF THE FIRST ELEMENT OF THE ARRAY TO BE SORTED.
IT IS RESONABLY EFFICIENT FOR N<=1000 AND DOESN'T DEGRADE
IF THE ARRAY IS IN ORDER(POSSIBLE PROBLEM WITH QUICKSORT)
SEE KNUTH VOL. III SEARCHING AND SORTING FOR DETAILS.*/

DECLARE ARRAYPTR ADDRESS, A BASED ARRAYPTR BYTE,
(N,I) ADDRESS, (SWITCHED,J,K,TEMP) BYTE, INCR ADDRESS;

IF N <=1 THEN RETURN;

INCR=N/2;

DO WHILE INCR>0;

DO I=0 TO INCR;

SWITCHED=1;

DO WHILE SWITCHED;

SWITCHED=0; K=I; J=INCR+I;

DO WHILE J<N;

IF A(K)>A(J) THEN

DG;

TEMP=A(K); A(K)=A(J); A(J)=TEMP; SWITCHED=1;

END;

K=J;

J=J+INCR;

END;

END;

END;

INCR=INCR/2;

END;

RETURN;

END SHELLSORT;

/* THE FOLLOWING IS THE REST OF THE TEST DRIVER PROGRAM */

CALL SHELLSORT (.B, 10);

DO I=0 TO N-1;

OUTPUT(I) = H(I);

END;

EOF

NO PROGRAM ERRORS

7-41
Text Storage Program

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.

8080, memory and TTY I/O port

None, other than this program

See program listing heading

Output is previously stored message

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Cross 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>100H &amp; buffer</td>
<td>Paul F. Fitts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>100H max</td>
<td>Innovatek Microsystems Inc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smithfield Road</td>
</tr>
<tr>
<td></td>
<td>Millerton, New York</td>
</tr>
</tbody>
</table>
; 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 ITY KBD
; GIVING 32 POSSIBLE POINTERS

; INNOVATEK MICROSYSTEMS INC
; SMITHEFIELD 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 REQUIRES 160 BYTES PLUS 64 BYTES
; FOR TABLE. MESSAGE BUFFER MAY BE ANY SIZE.
; ITY READER IS ENABLED BY BIT 0 OF OUT PORT 1
; ITY KBD READY INDICATED BY BIT 0 OF IN PORT 1 BEING 0
; ITY PTRR READY INDICATED BY BIT 2 OF IN PORT 1 BEING 0
; ITY PRINTER BUFFER IS OUT PORT 0
; ITY KBD BUFFER IS IN PORT 0
; THIS IS COMPATIBLE WITH INTELLEC 8

; CONSTANT DEFINITIONS

0031 SOH EQU 01H ;CNTRL A WITH PARITY
0033 ETX EQU 03H ;CNTRL C
0034 EOT EQU 04H ;CNTRL D
0032 STX EQU 02H ;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
0004 KDBE EQU 0 ;KBD IN PORT
0004 PTRY 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 ; ADSR OF BUFFER SPACE
0109  0E20  MVI  C, 32
010B  73  INILP:  MOV  M, E
010C  23  INX  H
010D  72  MOV  M, D
010E  23  INX  H
010F  8D  DCR  C
0110  C20B01  JNZ  INILP
0112  C08401  GETC:  CALL  GETCHR ; CHAR FROM ITTY KBD
0115  FE81  CPI  SOH ; CRTR A
0118  CA3601  JZ  LOADC
011B  E61F  ANI  1FH
011D  07  RLC  ; TWO BYTES / ADSR
011E  4F  MOV  C, A
011F  0600  MVI  B, 0

021  210002  LXI  H, TBLA ; TABLE START ADDRESS
024  09  DAD  B ; ADD INDEX, NOW H & L POINT TO TABLE LOCATION WHERE IS STORED START ADSR OF MESSAGE

0125  5E  MOV  E, M ; MOVE START ADSR TO D & E
0126  23  INX  H
0127  56  MOV  D, M
0128  EB  XCHG  ; CHAR ADRS 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 ; CRTR D
013C  FE84  CPI  EOT ; CRTR D
013E  CA1301  JZ  GETC ; LEAVE LOAD CHAR MODE
0141  FE82  CPI  STX ; CRTR B, ARSTART MESSAGE
0143  C23601  JNZ  LOADC
0146  CD9601  CALL  SNDCHR ; ECHO CHAR
0149  2A4802  LHLD  BFPR ; START ADSR OF NEXT MESSAGE
014C  EB  XCHG
014D  78  MOV  A, B
014E  E61F  ANI  1FH

7-45
0154  210002  LXI   H, TBLA
0157  09    DAD   B     ;H & L POINT TO TBL WHERE
0158  73    MOV   M, E     ;STRT ADDS IS TO BE STORED
0159  23    INX   H
015A  72    MOV   M, D
015B  EB    XCHG   ;BUFFER ADDS IN H & L
015C  065E  MVI   B, \n015E  CD9601  CALL  SNDCHR
0161  CD8401  LDCLP:  CALL  GETCHR
0164  FEFF  CPI   RUBOUT
0166  CA7E01  JZ    RBUF
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  C26101  JNZ   LDCLP
0175  224002  SHLD  BFPTR   ;SAVE ADDS OF NEXT MESSAGE
0178  C26001  JMP   LOADC
017B  2B    RBUF:  DCX   H     ;ERROR CORRECTION ROUTINE
017C  065C  MVI   B, \n017E  CD9601  CALL  SNDCHR
0181  C26101  JMP   LDCLP
0184  2E01  GetCHR:  MVI   A, REN   ;ENABLE READER
0186  D201  OUT   STRD
0188  9F    XRA   A
0189  D301  OUT   STRD
018B  DB01  GTCHLP:  IN   STRG   ;CHAR READY?
018D  E601  ANI   CHRDY
018F  C28001  JNZ   GTCHLP
0192  DB00  IN   KBDB     ;INPUT CHAR AND COMPLEMENT
0194  2F    CMA `   KBDB
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   A     ;COMPLEMENT AND OUTPUT
019F  D300  OUT   PTRB
01A1  C9    RET
0200  ORG   200H
0200  TBLA:  DS   64
0240  4202  BF PTR:  DW   BFRSP
0242  BFRSP:  DS   1000H
1FFE    ORG  1FFE
1FFE  00  STKA:  DB   0H
10  END
Time Sharing Communications

To communicate with medium to large scale computer system as an external time-share user.

Intellec 8/mod 80, 4k RAM, 2 I/O ports, Acoustic Coupler(Modem) (Floppy disk for storage of Hex programs - optional)

PLM compiler on host machine
Monitor on 8080
Disk Driver on 8080 - Optional

NA

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,B,C,D,E,H,L,SP</td>
<td>PLM</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>4K</td>
<td>Jim Jowell</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>NA</td>
<td>Univ. of Texas at Houston(UTHERCC)</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address: 6519 Fannin</td>
</tr>
<tr>
<td>NA</td>
<td>Houston, Texas 77025</td>
</tr>
</tbody>
</table>
8080 PLM VERS 2.0

00001 1 8001: /* ASCII TIME SHARING COMMUNICATIONS */
00002 1 /* INTELLEC 8/MOD 80 */
00003 : /* THIS PROGRAM WAS WRITTEN TO BE ABLE TO COMMUNICATE */
00004 1 /* WITH A COG 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 LITERALLY,
00014 1 DCL OA LIT #01H#, OR LIT #04H#;
00015 1 /* OA = DATA AVAILABLE, OR = TRANSMITTER BUFFER EMPTY */
00016 1 DCL PRO LIT #PROCEDURE#, OAT 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 HOS 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 DIPV BYTE, (S, Y) ADDRESS;
00030 1 /* STARTING FILE LOCATIONS ON THE FLOPPY DISK */
00031 1 DCL TRACK DATA (04H, 0EH, 017H, 020H, 029H, 032H, 03BH, 044H):
00032 1 /* 0 1 2 3 4 5 6 7 */
00033 1
00034 1 WRDOSK: PRO (OATT);
00035 2 DCL WR LIT #3743H#;
00036 2 DCL OATT ADDRESS;
00037 2 GO TO WR;
00038 2 END WRDOSK;
00039 1
00040 1 CONFPRNT: PRO (CHAPS);
00041 2 DCL CHAPS BYTE, PRNT LIT #350H#;
00042 2 PRT: GO TO PRT;
00043 2 END CONFPRNT;
00044 1
00045 1 LISTPRNT: P-0 (DIU);
00046 2 DCL DTX BYTE, LO LIT #340FH#;
00047 2 GO TO LO;
00048 2 END LISTPRNT;
00049 1
00050 1 CONSTAT: PRO (STS) BYTE;
00051 2 DCL STS BYTE;
00052 2 K = INPUT (I);
00053 2 RETURN K;
00054 2 END CONSTAT;
00055 1
00056 1 CRTSTAT: PRO (C. STS) BYTE;
00057 2 DCL CRTS BYTE;
00058 2 K = INPUT (S);
00059 2 RETURN K;
00060 2 END CRTSTAT;
00061 1

7-49
CRT$OUT: PRO (CHAT);
DCL CHAT BYTE, DUMMY BYTE;
DUMMY = NOT CHAT;
OUTPUT (*;) = DUMMY;
END CRT$OUT;

CRT$IN: PRO (INP) BYTE;
DCL INP BYTE, K BYTE;
INP = INPUT (4);
K = NOT INP;
RETURN K;
END CRT$IN;

DCL MEMORY (0) BYTE;
DCL CHAR BYTE;

HEA:
/* PRINT HEADING DATA TO CONSOLE */
DO I = 0 TO LAST (MHD);
DAT = MHD (I);
CALL CONSPRT (DAT);
END;

LOOP:
/* LOOP TO GET DISK FILE NUMBER FOR HEX OUTPUT */
/* OR S TO RETURN TO THE INTEL MONITOR */
STATUS = CONSTAT (K);
WORK = STATUS AND OA;
IF WORK <> 0 THEN GO TO LOOP;
ELSE
DAT = INPUT (0) AND 07FH;
OUTPUT (0) = DAT;
IF DAT = 0 THEN GO TO LOOP;
IF DAT < 031H THEN GO TO HEA:
/* FILE ONE */
IF DAT > 037H THEN GO TO HEA:
/* FILE SEVEN */
/* CALCULATIONS NECESSARY TO CREATE DISK ADDRESSES */
/* AND BUFFER LOCATION */
DAT = DAT AND 0FH;
TRACK = TRACK (DAT);
DATX = DAT + 7;
/* SYSTEM ON UNIT 0, DATA ON UNIT 1 */
ORIV = 1;
/* 26 SECTORS PER TRACK */
S = TRACK * 26;
I = 0;
COUNT = 0;
SECTR = 0;
H = 0;
SWITCH = 15;
BUFF = .MEMORY;

MAIN:
/* STATUS LOOP FOR CONSOLE AND INTELLECT CRT PORT */
STATUS = CONSTAT (K);
WORK = STATUS AND OA;
IF WORK = 0 THEN GO TO CONSOLE$DATA;
STATUS = CRT$STAT (K);
WORK = STATUS AND OA;
IF WORK = 0 THEN GO TO CRT$DATA;
ELSE GO TO MAIN;

CONSOLE$DATA:
DAT = INPUT (0) AND 07FH;
GET$STATUS:
STATUS = CRT$STAT (K);
WORK = STATUS AND IBNE;
ELSE
OUTPUT (+) = DAT;
GO TO MAIN;

DAT = CRTBIN (K) AND 0?FH;

CON&STATUS:
STATUS = CON&STAT (K);
WORK = STATUS AND IBNE;
IF WORK <> 0 THEN GO TO CON&STATUS;

/* LIST WILL DEFAULT TO THE CONSOLE */
CALL LISTPNT (DAT);

/* A STRING OF 40 ASTERISKS PRECEDE THE HEX FILE */
IF DAT = ### THEN COUNT = COUNT + 1;
ELSE IF COUNT <> 0 THEN COUNT = COUNT - 1;
IF SWITCH = 0 THEN GO TO SAVE;

/* SWITCH WILL BE SET WHEN 30 ASTERISKS IN A ROW HAVE BEEN RECEIVED, ALONG WITH THE COLON WHICH PRECEDES */

/* EACH DATA LINE */
IF COUNT < 30 THEN GO TO MAIN;

/* IF DAT <> ### THEN GO TO MAIN; */
ELSE SWITCH = 0;

SAVE:
/* A STRING OF 40 ASTERISKS FOLLOW THE HEX FILE */
/* H COUNTS THE ASTERISKS THAT ARE TOGETHER. I AM */
/* ASSUMING THAT THE DATA (HEX INSTRUCTIONS) WILL */
/* NOT HAVE 3 ASTERISK VALUES IN A ROW. */
BUF = DAT;

IF DAT = ### THEN H = H + 1;
IF H = 3 THEN GO TO DSKWRT;
BUFF = BUFF + 1;
GO TO MAIN;

DSKWRT:
BUFF = *MEMORY;
M = 0;
K = 0;
OORD:
/* THIS SEARCH IS FOR THE STRING OF 10 ZEROS THAT */
/* ARE ON THE LIST DATA LINE OF THE HEX FILE. */
/* (DO NOT USE AUTO START FOR THIS FUNCTION) */
/* DSK WRITE IS A 128 BYTE CHARACTER BLOCK. */
DO I = 0 TO 127;

IF BUF(I) = #00 THEN H = H + 1;
ELSE IF H <= 0 THEN H = H - 1;
IF H > 7 THEN K = 1;
END;

/* IS BUFFER LOCATION FOR DISK ROUTINE */
B = BUFF;
CALL WRISK (#DRIV);

/*SECTOR I3 SECTOR NUMBER FOR DISK ROUTINE */
SECTOR = SECTOR + 1;
IF K <> 0 THEN GO TO STZ;
ELSE BUFF = BUFF + 128;
GO TO OORD;

STZ:
SECTOR = SECTOR + 1;
/*SECTOR COUNT REPRESENTS SECTOR + 1 TO SYSTEM */
/* DSK IS SECTOR NUMBER FOR SYSTEM DISK FILE CONTROL */
S = DAT;
B = *SECTOR;
CALL WRDSK (.DRIV);
/* ALLOWS FOR MORE THAN ONE FILE TRANSMISSION */
GO TO 800H;

EOF
~ PROGRAM ERRORS

NO PROGRAM ERRORS

8080 PLM2 VERS 2.0
A Generalized Stepper Motor Driver Program

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.

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.

Existing firmware on Intellec 8, MOD80 - particularly subroutines "CO","CRLF", and "CSTS"

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.

A 2-bit binary counter is incremented or decremented and stored to keep track of motor position. Prior to outputting 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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum one to three</td>
<td>8080 MACRO assembler, ver. 2.0</td>
</tr>
<tr>
<td>Maximum = progr. size</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>About 90 loc. for a useful progr.</td>
<td>Floyd L. Nordin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nordin Enterprises</td>
</tr>
<tr>
<td>Box 1277 - Cupertino, Calif.</td>
</tr>
</tbody>
</table>
IBM Selectric output program

Allows IBM Selectric model 731 to be used as output device.

MCS-8 system, IBM model 731, appropriate drivers

MCS-8 monitor

Entered with character in register B.

Character printed by Selectric.
Input character returned in registers A and B.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, H, &amp; L</td>
<td>8008 MacroAssembler Ver 2.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>J. Harrison/W. Haskett</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 Bytes</td>
<td>Northeast Electronics</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Airport Road, Concord, NH 03301</td>
</tr>
</tbody>
</table>
SELECTRIC OUTPUT CONVERSION

3100 ORG 3100H
3100 3E80 START: MVI A, 80H
3102 B0 ORA B ; SET BIT 8
3104 6F MOV L, A ; USE FOR TABLE ADDR.
3104 26F1 MVI H, START/100H
3106 0620 WAIT: MVI B, 20H ; ROUTINE TO WAIT FOR
3108 D002 WAIT2: IN 2 ; SELECTRIC TO BE IDLE
310A E630 ANI 30H
310C C20831 JNZ WAIT2
310F 05 DCR B
3110 C20831 JNZ WAIT2
3113 D002 IN 2
3115 E640 ANI 40H ; CHECK FOR RIGHT MARGIN
3117 C26431 JNZ ENDEL
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 D002 IN 2
3125 AE XRA M ; GET SELECTRIC CODE
3126 E680 ANI 80H ; TEST FOR SHIFT CHANGE
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
3145 D002 CHANGE: IN 2
3147 E680 ANI 80H
3149 C25131 JNZ SHIFT ; CHANGE SHIFT
314C 3E01 MVI A, 01H
314C 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 CA3631 JZ RESTOR
315E CD6931 CALL SEND
3161 C33B31 JMP RESTOR
3164 3E04 ENDOL: MVI A.04H ;START NEW LINE
3166 C35331 JMP CSEND
3169 D308 SEND: OUT 11 ;SEND CONTROL
316B CD7331 CALL TIME ;WAIT FOR RESPONSE
316E 3E00 MVI A.0
3170 D308 OUT 11 ;RESET OUTPUT
3172 C9 RET
3173 0620 TIME: MVI B.20H ;ROUTINE TO WAIT
3175 D802 TIME2: IN 2 ;FOR RESPONSE
3177 E630 ANI 30H
3179 CA7531 JZ TIME2
317C 05 DCR B
317D C0 RZ
317F C35331 JMP TIME2
; CODE CONVERSION 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 00400H
3192 0000 DW 00000H
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 958E DW 0BE95H
31A4 F9F5 DW 0F5F9H
31A6 BD15 DW 015BDH
31A8 F081 DW 0B1F0H
31AA FCC6 DW 0C6FCH
31AC 4CC0 DW 0C04CH
31AE 1649 DW 04916H
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31B0</td>
<td>317F</td>
</tr>
<tr>
<td>31B2</td>
<td>763E</td>
</tr>
<tr>
<td>31B4</td>
<td>7975</td>
</tr>
<tr>
<td>31B6</td>
<td>343D</td>
</tr>
<tr>
<td>31B8</td>
<td>7C70</td>
</tr>
<tr>
<td>31BA</td>
<td>8D00</td>
</tr>
<tr>
<td>31BC</td>
<td>0046</td>
</tr>
<tr>
<td>31BE</td>
<td>00C9</td>
</tr>
<tr>
<td>31C0</td>
<td>F69C</td>
</tr>
<tr>
<td>31C2</td>
<td>A0AC</td>
</tr>
<tr>
<td>31C4</td>
<td>ED85</td>
</tr>
<tr>
<td>31C6</td>
<td>8E8F</td>
</tr>
<tr>
<td>31C8</td>
<td>E1D4</td>
</tr>
<tr>
<td>31CA</td>
<td>87E4</td>
</tr>
<tr>
<td>31CC</td>
<td>A09F</td>
</tr>
<tr>
<td>31CE</td>
<td>A699</td>
</tr>
<tr>
<td>31D0</td>
<td>C584</td>
</tr>
<tr>
<td>31D2</td>
<td>DD01</td>
</tr>
<tr>
<td>31D4</td>
<td>E7EE</td>
</tr>
<tr>
<td>31D6</td>
<td>DE90</td>
</tr>
<tr>
<td>31D8</td>
<td>AF81</td>
</tr>
<tr>
<td>1DA</td>
<td>B700</td>
</tr>
<tr>
<td>31DC</td>
<td>0000</td>
</tr>
<tr>
<td>31DE</td>
<td>0000</td>
</tr>
<tr>
<td>31E0</td>
<td>001C</td>
</tr>
<tr>
<td>31E2</td>
<td>202C</td>
</tr>
<tr>
<td>31E4</td>
<td>6D25</td>
</tr>
<tr>
<td>31E6</td>
<td>0E8F</td>
</tr>
<tr>
<td>31E8</td>
<td>6154</td>
</tr>
<tr>
<td>31EA</td>
<td>0764</td>
</tr>
<tr>
<td>31EC</td>
<td>291F</td>
</tr>
<tr>
<td>31EE</td>
<td>2619</td>
</tr>
<tr>
<td>31F0</td>
<td>4504</td>
</tr>
<tr>
<td>31F2</td>
<td>5D51</td>
</tr>
<tr>
<td>31F4</td>
<td>676E</td>
</tr>
<tr>
<td>31F6</td>
<td>5E10</td>
</tr>
<tr>
<td>31F8</td>
<td>2F01</td>
</tr>
<tr>
<td>31FA</td>
<td>3700</td>
</tr>
<tr>
<td>31FC</td>
<td>0000</td>
</tr>
<tr>
<td>31FE</td>
<td>0000</td>
</tr>
<tr>
<td>0000</td>
<td>END</td>
</tr>
</tbody>
</table>
Binary Search

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.

8080 with adequate memory for program and table.

None

Search parameter stored in location K.
Prestructured entry table (Note: assembly listing shown for 128 entries and a result field of 3 bytes).

Address of 1st byte of result field in HL if a match is found. Also, carry bit +0 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

**ARGUMENT TABLE** (must be in ascending order)

<table>
<thead>
<tr>
<th>( A_1 ) smallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_2 )</td>
</tr>
<tr>
<td>( A_3 )</td>
</tr>
<tr>
<td>( A_n ) largest</td>
</tr>
</tbody>
</table>

\[ \begin{array}{c}
\vdots \\
\vdots \\
\vdots \\
\end{array} \]

up to 128 arguments

**RESULT TABLE**

<table>
<thead>
<tr>
<th>( \text{RESULT} )</th>
<th>( R_1 ) (associated with ( A_1 ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1 of ( R_1 )</td>
<td>byte 2 of ( R_1 )</td>
</tr>
<tr>
<td>byte 1 of ( R_2 )</td>
<td>byte 2 of ( R_2 )</td>
</tr>
<tr>
<td>( R_3 )</td>
<td>( \vdots )</td>
</tr>
</tbody>
</table>

up to 128 results

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
0066 0600 MVI B, 0
0066 0E80 MVI C, NOENT
0068 78 MID: MOV A, B
0069 81 ADD C
006A 0F RRC
006B C67F ADI 07FH
006D 5F MOV E, A
006E 79 MOV A, C
006F B8 CMP B
0070 D400 JC NONE
0072 218000 LXI H, TABAD
0076 1600 MVI D, 0
0077 19 DAD D
0079 3AB200 LDA K
007A BE CMP M
007D C8D00 JZ MATCH
0080 D8800 JG LESS
0083 1C INR E
0084 43 MOV B, E
0085 C36800 JMP MID
0088 1D LESS: DCR E
0089 4B MOV C, E
008A C36800 JMP MID
008D 0600 MATCH: MVI B, 0
008F 1609 MVI D, 9
0091 4B MOV C, RSULG
0092 79 MUL1: MOV A, C
0093 1F RAR
0094 4F MOV C, A
0095 15 DCR D
0096 CAA300 JZ ADDR
0098 78 MOV A, B
0099 D29E00 JNC MUL2
009D 83 ADD E
009E 1F MUL2: RAR
009F 47 JMP MUL1
00A0 C39200 MOV B, A
00A2 21A00 ADDR: LXI H, RSUAD
00A6 09 DAD B
00A7 17 STC
00A8 3F CMC
00A9 C38000 JMP DONE

; INITIALIZE LOWER PTR TO 0
; INITIALIZE UPPER PTR TO N
; FIND MID PTR
; SUM UPPER & LOWER PTRS
; DIVIDE BY 2
; CLEAR SIGN BIT
; SAVE MID PTR
; SET UP FOR FAILURE TEST
; LOWER PTR > GT. UPPER PTR
; GET BASE OF TABLE
; CLEAR MSB
; COMPUTE MID PTR ADDRESS
; GET SEARCH PARAMETER VALUE
; COMPARE K=K(MID)
; ARGUMENT FOUND
; K.GT.K(MID), INC MID PTR
; SET NEW LOWER PTR
; K.LT.K(MID), DEC MID PTR
; SET NEW UPPER PTR
; COMPUTE RESULT ADDRESS
; BIT COUNTER
; GET RESULT LENGTH
; MULTIPLIER
; LOW-ORDER BYTE OF RESULT
; FINI
; MULTIPLICAND ADDED IF BIT = 1
; CARRY=0 SHIFT MSB
; BASE ADDR OF RESULT
; COMPUTE MATCH RESULT ADDR
; SUCCESS, C=0

7-63
00AC 37   NONE:    STC                        ; FAILURE, C:1
00AD 76   DONE:    HLT
00AE 3301  RSUAD:  DW  RESULT              ; RESULT BASE ADDRESS
00B0 B300  TABAD:  DW  TABLE               ; ARGUMENT BASE ADDRESS
00B0 00    NOENT EQU  128                  ; NO TABLE ENTRIES
00B3 00    RSULG EQU  3                     ; LENGTH OF RESULT ENTRY
00B3 00    K:    DB  0                     ; SEARCH PARAMETER
0133 00    TABLE:  DS  128                 ; ARGUMENTS
0000      RESULT:  DS  384                 ; RESULTS
TIMIT

Interrupt driven, real time clock routine. Provides time and calendar information.

100 Hz time base oscillator to drive interrupt circuitry of CPU. Vectored interrupt circuitry recommended.

Routine to initialize TIMIT's counters (symbolic location TIMER) with current time, day and year.

Time: hours, minutes, seconds, hundredths of seconds. Calendar: day of the year and year.

Maintains time in 24 hour format to hundredths 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)

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,B,C,D,E</td>
<td>MAC80, version 2.2</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>105 bytes</td>
<td>Steve Becquer</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td></td>
<td>U.C. Berkeley (student)</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>4</td>
<td>365 Clipper St.</td>
</tr>
<tr>
<td></td>
<td>San Francisco, Ca. 94114</td>
</tr>
</tbody>
</table>
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 LOSE COUNTS.

SYMBOLIC LOCATION 'TIMER' (SEE PROGRAM LISTING) IS THE COMMUNICATION AREA BETWEEN 'TIMIT' AND THE USER. 'TIMER' IS INITIALIZED BY THE USER, AND UPDATE THE 'TIMIT', FOR READ OUT BY THE USER. ALL TIME COUNTERS 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:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>CONTENTS</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HUNDRETHS OF A SECOND</td>
<td>0-99</td>
</tr>
<tr>
<td>1</td>
<td>SECONDS</td>
<td>0-59</td>
</tr>
<tr>
<td>2</td>
<td>MINUTES</td>
<td>0-59</td>
</tr>
<tr>
<td>3</td>
<td>HOURS</td>
<td>0-23</td>
</tr>
<tr>
<td>4</td>
<td>LOW ORDER TWO DIGITS OF DAY</td>
<td>1-99 OR 0-66 *</td>
</tr>
<tr>
<td>5</td>
<td>HIGH ORDER DIGIT OF DAY</td>
<td>0-3</td>
</tr>
<tr>
<td>6</td>
<td>LOW ORDER TWO DIGITS OF YEAR</td>
<td>0-99</td>
</tr>
<tr>
<td>7</td>
<td>HIGH ORDER DIGITS OF YEAR</td>
<td>0-99 **</td>
</tr>
</tbody>
</table>

* 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'.
ORG 8

JMP TIMIT

ORG 400

TIMIT: AN INTERRUPT DRIVEN, REAL TIME CLOCK.

BY STEVE BECOUER,
VERS 1.0.76008.

PUSH PSW; SAVE STATUS AND REGISTERS
PUSH B; OF INTERRUPTED ROUTINE.
PUSH D;

LXI D, TIMER; GET ADDRESS OF TIME COUNTERS.

MVI E, 0; SET MODULO TO 100
MVI C, 0; SET STARTING COUNT TO 0

CALL DIVIT; UPDATE 100THS OF SEC COUNTER.

MVI B, 60H; SET MODULO TO 60.

CALL DIVIT; UPDATE SECONDS COUNTER.

CALL DIVIT; UPDATE MINUTES COUNTER.

MVI B, 24H; SET MODULO TO 24.

CALL DIVIT; UPDATE HOURS COUNTER.

INX D; POINT TO HIGH ORDER DIGIT
OF DAY COUNT.

LDAX D; LOAD COUNT.

CPI 3; CHECK IF DAY COUNT IS OVER 300.

MVI B, 0; SET MODULO TO 100.

JNZ JULDY; IF LESS THAN 300,
RESET AT COUNT 100.

INX D; POINT TO LOW ORDER DIGITS
OF YEAR COUNT.

LDAX D; CHECK IF YEAR IS LEAP.

MVI B, 66H; SET MODULO TO 66,
IF YEAR IS NOT LEAP.

RRC; DIVIDE BY 2.

JC NLEAP; CARRY INDICATES NOT DIVISIBLE.

ORI 0F0H; IF LEAST SIGNIFICANT 4 BITS,
ARE GREATER THAN 9,

JC DIV4;

SUI 3; DECIMAL ADJUST.

DIV: RRC; DIVIDE BY 2, AGAIN.

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    JUDDY: DCX  D    ;TO LOW ORDER DIGITS OF DAY.
01CD CDE101 CALL DIVIT    ;UPDATE LOW ORDER DIGITS OF DAY.
01D0 0E04  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 0E00  MVI  B, 0    ;SET MODULO TO 100.
01D9 CDE101 CALL DIVIT    ;UPDATE YEAR COUNT.
01DC 01    EXIT: POP  D    ;RESTORE REGISTERS.
01DD 00    POP  B    ;
01DE F1    POP  PSW    ;RESTORE STATUS.
01DF FB    EI    ;ENABLE FURTHER INTERRUPTS.
01E0 C9    RET    ;RETURN TO INTERRUPTED ROUTINE.

DIVIT:

01E1 1A    LDAX  D    ;LOAD COUNTER TO BE INCREMENTED.
01E2 77    STC    ;
01E3 3F    CMC    ;CLEAR CARRY.
01E4 3C    INR  A    ;INCREMENT COUNT.
01E5 27    DAR    ;DECIMAL ADJUST.
01E6 B8    CMP  B    ;TEST IF RESET COUNT
01E7 C2EB01 JNZ  SAVE    ;HAS BEEN REACHED.

01EA 79    MOV  A, C    ;SAVE COUNT AND EXIT.
01EB 12    SAVE: STAX  D    ;RETURN COUNT TO MEMORY.
01EC 13    INX  D    ;POINT TO NEXT COUNTER.
01ED C8    RZ    ;IF RESET OCCURRED.
01EE D1    POP  D    ;GO UPDATE NEXT COUNTER.
01EF C3DC01 JMP EXIT    ;IF NOT, ISSUE A DUMMY RETURN.

TIMER:

;HERE TIME IS STORED, IN BCD, AS FOLLOWS:

;BYTE CONTAINS
01F2 00    DB  0    ;0 HUNDREDTHS 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
01F7 00    DB  0    ;5 HIGH ORDER DIGIT
01F8 00    DB  0    ;6 LOW ORDER TWO DIGITS
01F9 00    DB  0    ;7 HIGH ORDER TWO DIGITS
00    END
Absorbance Calculation

Calculate absorbances from % transmission data. Transmission data (I) is in fixed point format (16 bits maximum). Absorbance is fixed point times 10^4

None

Required
Software

Multiply 16 X 16 to give 16
Divide 16 X 16 to give 16 fraction (decimal at left)
Shift right

PRZER contains I₀ (reference intensity)
PRDAT contains I (sample intensity)
(where I/I₀ = % transmission)

Output
Results

Registers D & E holds absorbance X10^4 in fixed point format

Registers Modified:
    all

Assembler/Compiler Used:
Intellec 8 MAC80

RAM Required:
    8

Programmer:
W. B. Telfair

ROM Required:
    192

Company:
Wilks Scientific Corp.

Maximum Subroutine Nesting Level:
    3

Address: 140 Water Street
Box 449, S. Norwalk, Conn. 06856
; 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 ABSC: DS 40 ; A1, A2, A3, ETC.

015F 2A0501 ABSOR: LHLD PRZER ; ABSORBANCE CALCULATION: A = LOG(I0/I) =
0162 EB XCHG ;
0163 2A0701 LHLD PRDAT ;
0168 7C LSHFT: MOV A, H ; MAKE I0>1> 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
~188 CD6002 CALL DAVE
018E EB XCHG
018F 220A01 SHLD PRRES

7-72A
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
 01C7 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 B4 CMP D
01F0 C2AA01 JNZ ITERA
01F3 B8 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
^?01 112E16 LXI D,162EH; LN 2
^784 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 210887 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 SHFR: 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 SHFR
022B 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
024D 44 MOV B, H
024E 4D MOV C, L
024F EB XCHG
0250 29 DAD H
0251 23 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  MULTI:  MOV  A, B
028B 1F  RAR
028C 47  MOV  B, A
028D 79  MOV  A, C
028E 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
02AA 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

~02C6 010000 ABCAL: LXI B,0 ; CALCULATE ABSORBANCE FOR SERIES OF DATA (I)
  C5 LPCAL: PUSH B ; WITH SAME I0
02CA 2A0001 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 SLHLD 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
TELEPROCESSING BUFFER ROUTINE

To allow a microprocessor to buffer input/output to a USART to allow for speed differences between receiver/transmitter.

Intel 8251 USART

Intel PL/M compiler

Accepts input from 8251 and places character into buffer.

Transmits character from input buffer.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: William Speary</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: SPEARCO</td>
</tr>
<tr>
<td>ROM Required: 7CH</td>
<td>Address: 19837 Olney Mill Road</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level: 12H</td>
<td>City: Olney</td>
</tr>
<tr>
<td>Assembler/Compiler Used: PLM VERS 2.0</td>
<td>State: Maryland 20832</td>
</tr>
</tbody>
</table>
* REF. NO. D25 */
* PROGRAM TITLE TELEPROCESSING BUFFER ROUTINE */
* THIS IS AN EXAMPLE ON A BUFFER HANDLING PROGRAM. */
* IT CAN BE USED TO BUFFER INPUT IO AND FROM USERS. */
* IS DATA COME IN AT DIFFERENT RATE THEN THE OUTPUT */
* CAN ACCEPT IT, THE PROGRAM WILL BUFFER IT. */
* THE TIME DIFFERENCE BETWEEN THEN INPUT AND OUTPUT */
* CAN BE HANDLED BY VARYING THE SIZE OF 'BUFFER' AND */
* 'BUFLEN'. NOTE: 'BUFLEN' MUST BE ONE (1) LESS THAN */
* THE SIZE OF 'BUFFER' */

DECLARE LIT LITERALLY 'LITERALLY', DCL LIT 'DECLARE';
DCL VAD LIT '021H';
DCL VAD$IN LIT '020H';
DCL PROC LIT 'PROCEDURE';
DCL RSRDY LIT '002H';
DCL TSRDY LIT '01H';
DCL BUFLEN LIT '100';
DCL BUFFER(101) BYTE;
DCL COUNT BYTE;
DCL FREEPTR BYTE, OUTPTR BYTE;

SET COUNT TO NUMBER OF ITEMS THAT BUFFER CAN HOLD */
THIS NUMBER WILL BE REDUCED FOR EACH ITEM PLACED INTO */
BUFFER. IF THE BUFFER SPACE IS EXHAUSTED (I.E., DATA */
IS COMING IN FASTER THAN CAN BE OUTPUTED) THIS PROGRAM */
HALTS. */
IF 'COUNT' AND 'BUFLEN' ARE THE SAME THE BUFFER IS */
EMPTY AND THERE IS NO DATA TO BE OUTPUTED. */

COUNT=BUFLEN;

FREETR' POINTS TO NEXT FREE SLOT IN BUFFER. */
'OUTPUT' POINTS TO THE NEXT CHARACTER TO BE OUTPUTED. */

FREEPTR,OUTPTR= 1;

GETCHAR: PROC;
IF THERE IS NO MORE ROOM IN BUFFER 'HALT' */
IF (COUNT := COUNT - 1) = 0 THEN HALT;

GET CHARACTER AND PUT IT IN BUFFER. */
BUFFER(FREEPTR)= INPUT(VAD$IN);

UPDATE NEXT BUFFER EMPTY SLOT, IF END OF BUFFER */
WRAP TO THE BEGINING. */

IF (FREEPTR := FREEPTR +1) = BUFLEN THEN
FREEPTR=1;

PUTCHAR: PROC;
IF THEN LEN OF BUFFER AND THE COUNT OF THE NUMBER */
/* IF FREE SLOTS IN THE BUFFER ARE THE SAME, THEN RETURN */
/* CAUSE THERE IS NOTHING TO OUTPUT. */

IF (COUNT := COUNT+1)= BUFLEN+1 THEN
  DO ;
  COUNT:=BUFLEN ;
  RETURN ;
END ;

/* OUTPUT BUFFER CHARACTER. */
/* IF AT END OF BUFFER, WRAP BACK TO BEGINING. */

OUTPUT(VADSIN)= BUFFER(OUTPTR) ;
IF (OUTPTR := OUTPTR+1) = BUFLEN THEN
  OUTPTR:=1 ;
END ;

/* IF THERE IS A CHARACTER IN THE 8251 BUFFER, THEN */
/* GO GET IT. */

LOOP: IF (INPUT(VAD) AND R$RDY) <> 0 THEN
  CALL GETSCHAR ;

/* IF THEN IS A CHARACTER IN THE BUFFER AND THE 8251 CAN */
/* ACCEPT A CHARACTER THEN GO OUTPUT A CHARACTER. */

IF COUNT <> BUFLEN THEN
  IF (INPUT(VAD) AND $RDY) <> 0 THEN
    CALL PUTSCHAR ;
    GO TO LOOP ;

EDF
NO PROGRAM ERRORS
RUN Ø

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.

MDS with Diskette System

ISIS, PL/M Compiler

Name of binary file that is to be loaded or a default 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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PL/M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Victor H. Grinich</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Stanford University</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>26346 Esperanza Dr.</td>
</tr>
<tr>
<td></td>
<td>Los Altos Hills, CA, 94022</td>
</tr>
</tbody>
</table>
STAGE 2

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.

MDS 800 Intellec Microcomputer Development System with console device diskette drive(s) and controller, at least 48K RAM memory.

ISIS-II Operating System

Channel assignments equivalencing STAGE 2 channels with ISIS-II files and devices. This implementation of STAGE 2 supports nine non-dummy I/O channels.

User defined.

Program offered on diskette only.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PL/M 80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>48K minimum</td>
<td>Bruce W. Ravenel</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intel - MCD Software Development</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3065 Bowers Ave.</td>
</tr>
<tr>
<td></td>
<td>Santa Clara, Ca. 95051</td>
</tr>
</tbody>
</table>
PAPER TAPE LEADER I.D.

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 BUF$IZ 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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Ken Paul MS#14</td>
</tr>
</tbody>
</table>

RAM Required:
1K

ROM Required:
MDS Monitor (2K)

Maximum Subroutine Nesting Level:
N/A

Assembler/Compiler Used:
8080 MDS Macro Assembler V1.0

Programmer:
Ken Paul MS#14

Company:
Control Data Corporation

Address:
11615 I Street

City:
Omaha

State:
Nebraska 68137
**PAPER TAPE LEADER I.D.**

(see page 1)

Hardware required is the standard MDS-800 with TTY (with tape reader and punch) connected to the MDS TTY port connector.

**MDS Monitor**

Read in program. Give starting address command (G300) and input message on console device.

Readable alphanumerics and symbols punched into paper tape in paper tape punch.

---

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Ken Paul MS*14</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>Control Data Corporation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS Monitor (2K)</td>
<td>11615 I Street</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>Omaha</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080 MDS Macro Assembler V1.0</td>
<td>Nebraska 68137</td>
</tr>
</tbody>
</table>
LSORT

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).

None

None

HL = Address of first character in file.

Each record in the file is in alphabetical order.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Dick Springer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>III HEX</td>
<td>General Microwave Corp.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>155 Marine Street</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Farmingdale</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>New York</td>
</tr>
</tbody>
</table>

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  CD9100  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  2A9F00  LHLD  SPD
002F  CD7F00  CALL  CLOSE
0032  2A9000  LHLD  SPA
0035  CD6000  CALL  OPEN
0038  2AB000  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     LHLH     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
7066 4D        MOV      C, L
0067 2AB00     LHLH     SPB
006A 1A        LDAX     D
006B 02        STAX     B
006C 7C        MOV      A, H
006D 9A        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 C35000     JMP      LOAD+3

CLOSE:
007F CD5300     CALL     LOAD
0082 EB        XCHG
0083 2AA000     LHLH     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
108F 2AA000     LHLH     SPD
0092 C9        RET
COMP:

0093  22A000    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  C29000    JNZ     $-12
00A8  C9       RET
00A9  0000    SPA:    DW     0
00AB  0000    SPB:    DW     0
00AC  0000    CPC:    DW     0
00AF  0000    SPD:    DW     0
00B1  IBUF:    DS      60H

0000    END
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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>L. Leipuner</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Brookhaven National Laboratory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>105.</td>
<td>Building 510</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Upton</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDP11 Macro's</td>
<td>New York 11973</td>
</tr>
</tbody>
</table>
SDK-80 PAPER TAPE PUNCH ROUTINE

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
- NMOUT Ø2C3
- CO Ø1E3
- CI Ø1D0
- HILO Ø29C

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.

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 inputting 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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ 50 bytes + stack</td>
<td>Steve Richards</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>VORTEK Industries Ltd.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1820 Pandora St., Van., B.C.,</td>
</tr>
</tbody>
</table>
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
120F 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 0E20  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
Type K T.C. Linearizer (Procedure)

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).

Intellect 8/Mod 80 or any 8080 based Microprocessor

PLM Compiler on Host Machine

Two: Raw$Temp = Measured Junction (0.091 MV per Bit)
Ref$Temp = Reference Junction Compensation (0.091 MV per Bit)

One: Comp$Temp = Binary Equivalent to Measured Temperature
From 0° to 1000°F With Transfer Function
Equal to 4.07°F Per Bit.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, D, E, H, L, &amp; SP</td>
<td>O. Leon Lindsey, Sr. Research Eng.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Bytes (Variables) + 6 Bytes (Stack) = 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>111 Bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Six</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLM</td>
</tr>
</tbody>
</table>
TCK-LPT

00001 1 /* HOUSEKEEPING ENTRIES */
00002 1 DECLARE DCL LITERALLY 'DECLARE';
00003 1 DCL лит LITERALLY 'LITERALLY';
00004 1 DCL PRC LIT 'PROCEDURE';
00005 1 DCL ADR LIT 'ADDRESS';
00006 1
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

READY

?V
?

EOF

NO PROGRAM ERRORS
Enable Hold - Screen Mode

Enable Hold-Screen-Mode of DEC Scope Model VT-52

MDS 800 with DEC Scope Model VT-52

No additional software required.

None

Enables Hold-Screen-Mode of DEC Scope Model VT-52

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>D. Brenneman</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company: General Electric</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>City: Waynesboro</td>
</tr>
<tr>
<td>ISIS 8080 MACRO Assembler V1.1</td>
<td>State: VA 22980</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
; REF. NO. D36
; PROGRAM TITLE ENHSM
; 
; TITLE' ENHSM 15 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
70BE05B  MVI C,1 ;EN H-S-M
750A 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 DESTROYED)

F51A DBF7  CRTOU: IN CRTS
F51C E601  ANI TBR
F51E CD1AF5 JZ CRTOU ;LOOP TILL READY
F521 79  MOV A,C
F522 D3F6  OUT CRTDP
F524 C9  RET

F525 25F5 STACK: DW $

F500  END START
Disable Hold - Screen Mode

Disable Hold-Screen-Mode of DEC Scope Model VT-52

MDS 800 (with DOS) and DEC Scope

No Additional software required

None

Disable Hold-Screen-Mode of DEC Scope

Registers Modified: D. Brenneman

RAM Required: General Electric

ROM Required: Address:

Maximum Subroutine Nesting Level: Waynesboro

Assembler/Compiler Used: VA 22980
ISIS 8080 MACRO Assembler V1.1
; REF. NO. D37
; PROGRAM TITLE DIHSM
;
;
TITLE ' DIHSM 15 77 DB ';

; THIS WILL DISABLE THE DECSCOPE HOLD-SCREEN-MODE

00F6 CRTDP EQU 0F6H ; CRT DATA PORT
00F7 CRTS EQU 0F7H ; CRT STATUS PORT
0011 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 0E5C MVI C,'\n'; 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 DESTROYED)

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
F524C9 RET

F525 25F5 STACK: DW $

F500 END START
THERMOCouple LINEarization (Type J)

Program will convert millivolt values from a Type J thermocouple to degrees centigrade. (Reference junction 0 degrees centigrade.) Range from -220 deg. C to 300 deg. C.

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.

May use own math package if compatible.

Register pair D & E contain millivolt readings times 1000.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer:</td>
<td>PETER C. SCHMITT</td>
</tr>
<tr>
<td>Company:</td>
<td>OMICRON, INC.</td>
</tr>
<tr>
<td>Address:</td>
<td>1111 E. TEN MILE RD</td>
</tr>
<tr>
<td>City:</td>
<td>MADISON HEIGHTS</td>
</tr>
<tr>
<td>State:</td>
<td>MICHIGAN 48071</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
INVERT Data in RAM

Invert Data in RAM

INTELLECT 8/Mod 80

INTELLECT 8/Mod 80 MONITOR (1975)

- I
- ADDRESS (Hex) of Beginning
- , (comma)
- ADDRESS (Hex) of Finish
- CR

Contents of RAM (from beginning thru finish) is complemented.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D. Brenneman</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td></td>
<td>General Electric</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td></td>
<td>GE Drive</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ISIS 8080 MACRO ASSEMBLER, V1.1

INVERT 1/18/77 DB.

;REF. NO. D39
;PROGRAM TITLE INVERT
;
;
TITLE 'INVERT 1/18/77 DB.'

; THIS IS DESIGNED TO BE A MODIFICATION TO THE
; INTELLECT/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 (CARRIAGE 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
3FEF CDA43D CALL 3DA4H ;COMPARE HL WITH DE (AND INX H)
3FEE D2E3F JNC $-6 ;LOOP TILL HL=DE
3FF2 C36B38 JMP 386BH ;START - ISSUE PROMPT WHEN DONE

0000 END
RELATIVE JUMP ROUTINE

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.

THE CALLING SEQUENCE IS AS FOLLOWS:

RST X ; RESTART TO RELATIVE JUMP ROUTINE
DB N ; ADDRESS OFFSET

OR, WITH THE INCLUDED ADDRESS CONVERSION MACRO:

JR ADDR ; JR CONVERTS ADDR TO OFFSET

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)
; REF. NO. D40
; PROGRAM TITLE JR RELATIVE 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.
;  EGE:  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 -
;  D -
;  H -
;  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 ; UNSTACK REGISTERS
0044 D1  POP D
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
**Program Title**

JULIAN DATE ROUTINE

Check validity of, and/or convert to Julian, dates from January 1, 1901 and December 31, 2099.

**Function**

8080 System

**Required Hardware**

Driver Routine

**Required Software**

H, L contains address of ASCII MMDDYY

**Input Parameters**

D, E contains Julian Date (integer)
Carry set if and only if invalid date

---

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>D, E, PSW, A</td>
<td>Leonard J. Jowers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$078H</td>
<td>Sullivan, Long &amp; Hagerty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>P.O. Box 2247</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 levels</td>
<td>Birmingham</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080 MACRO ASSEMBLER</td>
<td>AL 35201</td>
</tr>
</tbody>
</table>
; REF. NO. D41
; PROGRAM TITLE JULIAN DATE ROUTINE
; 
; --------------------------------------
; JULIAN DATE
; 
; H, L ---- ASCII MMDDYY ADDR
; D, E <---- JULIAN DATE
; CARRY IFF ERROR
; 
; 0000 E5
; 0001 C5
; 0002 CD4700 CALL JS1; CARRY IFF ERROR
; 0005 57
; 0006 D44700 CNC JS1;
; 0009 5F
; 000A D44700 CNC JS1;
; 000D DA4400
; 0010 216300 LXI H, JULTBL+1;
; 0013 0600
; 0015 4A
; 0016 09
; 0017 09
; 0018 46
; 0019 2B
; 001A 4E
; 001B 2B
; 001C C5
; 001D 46
; 001E 2B
; 001F 4E
; 0020 6B
; 0021 2600
; 0023 09
; 0024 C1
; 0025 E603
; 0027 7A
; 0028 C23500
; 002B FE02
; 002D DA3500
; 0033 23
; 0034 03
; 0035 FE0D
; ~937 D24300 JNC JERX;
; J3A EB
; 003B 7A
; 003C 2F
; 
; 
;
MOV H, A;
MOV A, E;
CMA;
MOV L, A;
INX H;
DAD B;
JERX: CMC;
JERR: POP B;
POP H;
RET;

CALL JS2;
RC;
RLC;
MOV B, A;
RLC;
RLC;
ADD B;
MOV B, A;
CALL JS2;
RC;
ADD B;
CPI 1;
RET;

MOV A, M;
INX H;
SUI '0';
RC;
CPI 10;
CMC;
RET;

JULTBL: DW 000, 031, 059, 090, 120, 151;
JUL-DEC, JAN

DW 181, 212, 243, 273, 304, 334, 365;
JUL-DEC, JAN

END;
TERMINET* 1200

Software Interface for Terminet* 1200

MDS-800 & TERMINET* 1200 with Punch (Note: Hardware modification required to get MDS to operate at 1200 Baud)

No additional software required

None

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:  
State:  
VA 22980

ISIS 8080 MACRO ASSEMBLER V1.1

© Intel Corporation, 1976  
6/8/77  
98-034C  
7-122
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:
State:
ISIS 8080 MACRO ASSEMBLER VI.1 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:
; :LI: USER LIST DEVICE
; :IL: USER CONSOLE INPUT
; :OL: 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

7-126
F608 0E06 MVI C, 6 ; INSTALL USER LIST DEVICE
F609 114AF6 LXI D, L01
F610 CD1EF8 CALL IODEF

F613 0E00 MVI C, 0 ; INSTALL USER CONSOLE IN
F615 117BF6 LXI D, C11
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, P01
F628 CD1EF8 CALL IODEF

; CHECK WHICH DEVICE IS COLD START CONSOLE
F62B CD15F8 CALL I0CHK ; MONITOR CALL
F62E EE03 ANI CRTMSK ; CHECK IF CRT
F630 FE01 CPI CRTDEV
JZ CRT ; DO NOTHING

; CONSOLE IS TERMINET, INSTALL USER CONSOLE
F635 0EE3 MVI C, 0E3H ; MONITOR CONSOLE
F637 CD18F8 CALL I0SET
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

P01: ; OUTPUT TO PUNCH
C01: ; OUTPUT TO CONSOLE (IF TTY)
L01: ; OUTPUT TO LIST DEVICE

F64A DBF5 IN TTYS ; CHECK USART XMIT STATUS
F64C E601 ANI TBE ; CHECK USART XMIT STATUS
F64E CA4BF6 JZ C01
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
F55B FE0A CPI LF ; LOOK FOR LINE FEED
J5D CO RNZ
F65E C5 PUSH B
F65F 0E09 MVI B, 9 ; NUMBER OF FILL REQUIRED
I. 8080 MACRO ASSEMBLER, V1.1
TN300 2777 DB

F661 0E7F  MVI  C, 7FH   ; USING DELETE AS FILL
F663 CD4AF6  FILL:  CALL  C01   ; 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 CD3BF6  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
F7F CB7BF6  J2  CI1
F682 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  '11: '
F698 20  
F699 3A4F313A  OUTP:  DB  '01: '
F69D 20  
F69E  KSTAT:  DS  2
F6A0 A0F6  STACK:  DW  $  
J00  END  START
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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D. Brenneman</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>General Electric</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GE Drive</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Waynesboro</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISIS 8080 MACRO ASSEMBLER VI.1</td>
<td>VA</td>
</tr>
<tr>
<td></td>
<td>22980</td>
</tr>
</tbody>
</table>
; 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

F406 2143F4 LXI H, TBL
F406 160E MV1 D, 14 ; NUMBER OF SETTINGS
F408 1E05 MV1 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 MV1 E, 8 ; NUMBER OF SPACES BETWEEN SETTINGS
F415 0E20 MV1 C, CR
F417 CD38F4 L2: CALL TTYOU
F41A 1D DCR E
F41B C217F4 JNZ L2

F41E 2B DCX H
F41F 2B DCX H
F420 1E02 MV1 E, 2
F422 15 DCR D
F423 C20AF4 JNZ L1

F426 0E0D MV1 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
F432 CD4000  CALL   ISIS
F436 FB     EI
F437 76     HLT

F438 DBF5   TTYOU:  IN    TTS
F43A EE01   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
GRAPH

Plots on console a graph of one variable as well as axes and a grid.

Teletype (or similar)

'Console Output' subroutine

A block of memory holding data bytes to be plotted.

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. #D45

Intel Corporation, 1976
; REF. NO. D45
; PROGRAM TITLE GRAPH
;
;
SUBROUTINE GRAPH
;
; THIS SUBROUTINE WILL PLOT A GRAPH OF ONE FUNCTION
; OF ONE VARIABLE WHERE THE INDEPENDENT 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
; 68 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 PLEASURES. 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 113DF LXI D,0FF3DH ; YES
0107 46 MOV B,M ; GET Y-COORD.
~108 05 GR01: DCR B ; CHECK-REACHED Y YET?
109 FA2201 JM GR03 ; YES- JUMP
010C 14 INR D ; INCR. SPACE COUNTER
010D C21501 JNZ GR02

7-136
I. 5 8080 MACRO ASSEMBLER, V1.1

0110 0E2B       MVI C,'+'
0112 C32801     JMP GR04

0115 0E2D       MVI C,'-'
0117 7A         MOV A,D
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       MVI B,00H
                ;INSURE B NEVER GETS -VE AGAIN
0124 0E58       MVI C,'X'
0126 14         INR D
0127 23         INX H

0128 CD09F8     CALL CO
012B 1D         DCR E
012C C20801     JNZ GR01
                ;NO- JUMP
012F 0E0D       MVI C,CR
0131 CD09F8     CALL CO
0134 0E0A       MVI C,LF
0136 CD09F8     CALL CO
0139 C9         RET
                ;RETURN FROM 1ST CALL

013A 1EFF       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         DCR B
                ;RESTORE DATA
0144 78         MOV A,E
                ;EVAL. LINE NO. MODULO 5
0145 D605       SUI 005H
0147 C24B01     JNZ GR10
014A 5F         MOV E,A

014B 05         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

0158 15         DCR D
                ;CLEAR ACC.
0159 AF         XRA A
015A 83         ADD E
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'
0176 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
Analog/Digital Polling Routine

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.

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").

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.

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.

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.
Thumbwheel  SBC 80/10 Test Program

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.

**SBC 80/10**
TTY
6 digit Thumbwheel

**SBC 80P Monitor (PROM)**

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.

Digits are typed out on console, along with a prompt from the program saying you may change switch settings.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALL</strong></td>
<td>R. O. Christie</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company: Dept. of Environment Inland Water Directorate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>10 BYTES + Stack</strong></td>
<td>Address: 131 Greber Blvd.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>City: Pte Gatineau, P. O.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>320 BYTES</strong></td>
<td>State: Quebec, Canada</td>
</tr>
</tbody>
</table>

Assembler/Compiler Used:

**ISIS MACRO ASSEMBLER**
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.
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 31FF3F ORG 0800H ;TOP OF SBC80/10 RAM
0800 31FF3F LXI SP, 3FFFH
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, 1001011B ;CONTROL WORD
0818 D3EB OUT 0E8H ;CONTROL REG. ADDR.

;-------READ IN BCD DIGITS------------------------

081A 21E03F START: LXI H, STORE ;FIRST BCD PAIR
081D DBE8 IN 0E8H
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
082F 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 -
0888 C31A08   JMP      START   ;NEW THUMBWHEEL SETTING -
                                  ;THEN PROGRAM RESTARTS WITH -
                                  ;NEXT G COMMAND.

;--------CONSOLE DATA---------------------

0888 20202020   SINON:  DB
088F 20202020
0893 20202020
0897 20202020
089B 20202020
089F 2020
08A1 53424320   DB    ’SBC 80/10 TEST PROGRAM #1’
08A5 38302F31
08A9 30205445
08AD 52542050
08B1 524F4752
08B5 414D2023
08B9 31
08BA 0D
08BB 0A
08BC 0A
08BD 52454144
08C1 5320494E
08C5 20534958
08C9 20424344
08CD 20444947
08D1 49545320
08D5 46524F4D
08D9 20544855
08DD 4D425748
08E1 45454C20
08E5 53574954
08E9 43482051
08ED 45545449
08F1 4E475320
08F5 0D
08F6 0A
3F7 414E4420
08FB 44495350
08FF 4C415953
0903 20544845
0907 4D204F4E
090B 20434F4E
090F 534F4C45
0913 2E20
0915 0D
0916 0A
0917 0D
0918 0A
0919 24242424
091D 594F5520 NEWTH: 0000 END
0921 4D415920
0925 4348414E
0929 47452054
092D 48554D42
0931 57484545
0935 4C205345
0939 5454494E
093D 472E
093F 24242424

DB 'READS IN SIX BCD DIGITS'

DB 'FROM THUMBWHEEL SWITCH SETTINGS'

DB 'AND DISPLAYS THEM ON CONSOLE.'

DB 'YOU MAY CHANGE THUMBWHEEL SETTING.'

DB '****'

DB '****'
Program Title

Calculate a calendar for any year.
To generate and print on a list device or console a calendar for any operator specified year.

Function

8080 system with console device and optional list device.

Required Hardware

Monitor console out (CO), console in (CI) and list out (LO).

Required Software

A year accepted from the console keyboard.

Input Parameters

See sample run.

Output Results

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

8/8/77 © Intel Corporation, 1976
String Manipulation Package

See Attached Sheets

8080 Computer

No Special Software Required

See Attached Sheets

See Attached Sheets

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Frederick A. Stearns</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>See Attached Sheets</td>
<td>Systems Consultants, Inc.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>See Attached Sheets</td>
<td>3255 Wing Street</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>San Diego</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS-800 Assembler</td>
<td>California 92110</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MACRO Assembler V1.0</th>
</tr>
</thead>
</table>
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 preceded 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.
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 '0' 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.

<table>
<thead>
<tr>
<th>Registers Modified: none</th>
<th>Programmer: R. Genoud</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: Micro Control</td>
</tr>
<tr>
<td>ROM Required: 83 Bytes + Message Table</td>
<td>Address: 5107 Schinzach-Dorf Switzerland</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level: 3</td>
<td>City:</td>
</tr>
<tr>
<td>Assembler/Compiler Used: ASM80</td>
<td>State:</td>
</tr>
</tbody>
</table>
Program Title

SBC 80P REAL TIME CLOCK

MONITOR RESIDENT, REAL TIME CLOCK ROUTINE.
PROVIDES TIME AND CALENDAR INFORMATION.

Function

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

Company:

NUCLEAR RESEARCH CENTER

Address:

P.O.B. 9001

City:

BEER-SHEBA

State:

ISRAEL

© Intel Corporation, 1976
REAL TIME CLOCK SERVICE ROUTINE

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.

System real time clock

Example:
DATE: 06/07/77 for JULY 6/77
TIME: 15:05:00 for HOUR:MIN:SEC
Note that leap year is considered. With LEAP = 8H, the program is good until the end of 1983 by which time LEAP must be set equal to 84H.

Time will be stored in appropriate buffer, i.e. from SEC to YEAR in buffer.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer: J.L. MARCEL LALONDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>244H bytes</td>
<td>Company: AGRICULTURE CANADA</td>
</tr>
<tr>
<td>ROM Required:</td>
<td></td>
<td>Address: ENGINEERING RES. SERV., RESEARCH BRANCH</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>4</td>
<td>City: OTTAWA, ONT. K1A 0C6</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>ISIS-II 8080/8085 ASSEMBLER V1.0</td>
<td>State: ONTARIO</td>
</tr>
</tbody>
</table>
FIELD

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.

8080 System

None

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

EXAMPLES:

VALUE 00000023  VALUE 6789012345
MASK >>>>>   MASK <<>>>## ##
RESULT 23     RESULT #67,890,123.45

VALUE 000000008  VALUE 0000123456
MASK $$$*##  MASK **,***,### DOLLARS AND ## CENTS
RESULT 88     RESULT ****1,234 DOLLARS AND 56 CENTS

VALUE 0789012345  VALUE 608LL51212PC1234GETTY J.
MASK $$,###,##  MASK (###) (###) EXT. (###) GETTY J.
RESULT $7,090,123.45 RESULT (609) L51212 EXT. FC-1234 GETTY J.

 Registers Modified:
 ABDEHL

 RAM Required:
 34 Mask Length

 ROM Required:
 B6

 Maximum Subroutine Nesting Level:
 None

 Assembler/Compiler Used:
 8080 Macro Assembler

 Programmer:
 Austin C. Nester

 Company:  Coordinated Systems Corp.
 Address:  7300 Industrial Park
 City: Pennsauken
 State: NJ 08110

 © Intel Corporation, 1976
 12/6/77
; 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 SIGNIFICANT
; 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:
J0A 7E MOV A,M ;GET MASK BYTE
090B FE24 CPI '$'
090D CA2F09 JZ DOLR
0910 FE3E  CPI    ""]
0912 CA5009 JZ     ZSR
0915 FE2C  CPI    ""]
0917 CA8409 JZ     COMR
091A FE2E  CPI    ""]
091C CA0E09 JZ     DTOR
091F FE23  CPI    ""]
0921 CA9E09 JZ     NINR
0924 FE2A  CPI    ""]
0926 CAEE09 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 C2A009 JNZ    EDITG ; PROCESSED, DO NEXT BYTE
092E C9  RET    ; ELSE, EXIT

; DOLR:
092F 33AA09 LDA    SDF
0932 B7  ORA    A ; IF NO SIGNIFICANT DIGIT HAS BEEN
0933 CA0C09 JZ     DOLR3 ; EXAMINE THIS DIGIT

DOLR1:
0936 1A  LDAX    D ; GET THIS DIGIT

DOLR2:
0937 13  INX    D ; ADVANCE TO NEXT SOURCE DIGIT

DOLR3:
0938 77  MOV    M, A ; STORE DIGIT OVER MASK BYTE
0939 C32909 JMP    EDITH ; PROCESS NEXT DIGIT

DOLR4:
093C 1A  LDAX    D ; GET THIS DIGIT
093D FE30  CPI    ""] ; IF IT IS A 0
093F CA4C09 JZ     DOLR5 ; SUPPRESS IT
0942 CD5409 CALL   DOLR6 ; INSERT THE # 1 LOCATION LEFT

DOLR5:
0945 3D  DCR    A ; SET THE SIGNIFICANT DIGIT FOUND
0946 32B009 STA    SDF ; STORE DIGIT
0949 C33609 JMP    DOLR1 ; STORE DIGIT

DOLR6:
094C 3E20  MVI    A, ""] ; GET A SPACE
094E 32B909 STA    FDF ; SET FLOATING DOLLAR FLAG
0951 C33709 JMP    DOLR2 ; OVERWRITE MASK DIGIT

DOLR7:
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
095D 33AA09 LDA    SDF
IF A SIGNIFICANT DIGIT HAS BEEN
COPY THIS DIGIT, ELSE
GET THIS DIGIT
SUPPRESS IT
IF NO FLOATING $ REQUIRED
SET SIGNIFICANT DIGIT FOUND FLAG
INSERT THE $ AND STORE THE DIGIT

ZSR1:
INX D

ZSR2:

IF CHECK PROTECT IS IN EFFECT
GET AN ASTERISK, ELSE
GET A SPACE
OVERWRITE MASK BYTE
IF NO SIGNIFICANT DIGIT HAS BEEN
BLANK OUT MASK BYTE ELSE LEAVE THE COMMA

SET SIGNIFICANT DIGIT FOUND FLAG
IF A FLOATING $ IS NOT REQUIRED
LEAVE THE DECIMAL POINT ELSE INSERT THE $ THEN
LEAVE THE DECIMAL POINT

SET SIGNIFICANT DIGIT FOUND FLAG
IF A FLOATING $ IS NOT REQUIRED
STORE THE DIGIT ELSE
INSERT THE $ AND STORE THE DIGIT

SET THE CHECK PROTECT FLAG DO CHECK PROTECTION
GET AN ASTERISK AND OVERWRITE THE MASK BYTE
FLOATING DOLLAR FLAG
SIGNIFICANT DIGIT FOUND FLAG
CHECK PROTECT FLAG
T1: ; 23
09C6 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'
0A40 30303233

V2: 
0A44 30303030 DB '0000000008'
0A48 30303030

7-167
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 11300A LXI D, V1 ; GET SOURCE ADDRESS
0A92 213009 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
0AAB 062F MVI B, 47
0AB0 CD0009 CALL EDITF
0AB3 76 HLT
0000 END
SERIAL PROM PROGRAMMER

Programs Intel 1K x 8 (2708) EPROMS on Prompt-80 using an external serial input.

Prompt 80 modified for RS-232c (300 Baud) monitor V1.0

External device must transmit to the Prompt-80 in Intel modified Hex Format.

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.

A 16 byte Field in PROM is programmed and prompt of ASC II "L" sent to host device requesting next 16 bytes of data.
; 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
302   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  C3080C XC:  JMP XC2
0C06  3EFF XP2:  MVI A, 0FFH
0C08  C30C0C JCMP 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
E111  D3ED OUT TTC
E133  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      XOR   A
0C30 57      MOV   D, A
0C31 CD760C   CALL   BYTE
0C34 FE10     CPI   10H
0C36 C23800   JNZ   ERR
;39 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 C23800   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 C23800   JNZ   ERR
0C5D D1      POP   D
;  SET UP PROGRAM/VERIFY PARAMETERS

0C5E 21180C   LXI   H, READY
0C61 E5      PUSH   H
0C62 21003D   LXI   H, BFR
0C65 E5      PUSH   H
0C66 210F3D   LXI   H, BFR+15
0C69 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
0C8D 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 E7   ORA    A
0CA6 C9   RET

; SERIAL INPUT ROUTINE

0CA7 DBED  CI:   IN    TTC
0CA9 E602  ANI    RXRDY
0CAB CA870C JZ    CI
0CAB DBEC  IN    RXD
0C80 E67F  ANI    7FH
0C82 C9   RET
7000 END    START
Program Title

Mailing Label Program - LABEL

The program takes a list of names and addresses and outputs them to mailing labels on a line printer.

Function

MDS-800 and printer.

Required Hardware

ISIS

Required Software

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.

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

© Intel Corporation, 1976

8/78

Ref. #D56
Mailing Label Program

The program takes a list of names and addresses and outputs them to mailing labels on a line printer. 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 Dunning, MI 46716 245 3722

The program is invoked by the command

```
LABEL FILENAME
```

Where FILENAME is the file containing the names and addresses.

The user is queried for the number of names in the file. The maximum number is 255 decimal.

```
1 DECLARE FILENAME(16) BYTE;
2 DECLARE INBUF(3) BYTE;
3 DECLARE MSG(99) BYTE DATA('How many names in file?');
4 DECLARE AFTAB(8) BYTE;
5 DECLARE BUCKET(99) BYTE DATA('BB:');
6 DECLARE DO_BYTE INITIAL(36); /* D0 is the number of spaces from the start of one label to the beginning of the next */
7 DECLARE (L,M,N,NUMCHARS, NUMTABS, NUMSPACES) BYTE;
8 DECLARE NUM NAMES BYTE INITIAL (31);
9 DECLARE (K, ACTUAL COUNT, STATUS, AFTIN) ADDRESS;
10 DECLARE LABELBUF(36) BYTE;
11 DECLARE READACCESS LITERALLY 'l';
12 DECLARE LINESPTR(*) BYTE DATA ('LP:');
13 DECLARE CRLF(2) BYTE DATA (0DH, 0AH);
14 OPEN: PROCEDURE (AFTPTR, FILE, ACCESS, MODE, STATUS) EXTERNAL;
15 DECLARE (AFTPTR, FILE, ACCESS, MODE, STATUS) ADDRESS;
16 END OPEN;
17 CLOSE: PROCEDURE (AFT, STATUS) EXTERNAL;
18 DECLARE (AFT, STATUS) ADDRESS;
19 END CLOSE;
20 READ: PROCEDURE (AFT, BUFFER, COUNT, ACTUAL, STATUS) EXTERNAL;
21 DECLARE (AFT, BUFFER, COUNT, ACTUAL, STATUS) ADDRESS;
22 END READ;
```
WRITE: PROCEDURE (AFT, BUFFER, COUNT, STATUS) EXTERNAL;
DECLARE (AFT, BUFFER, COUNT, STATUS) ADDRESS;
END WRITE;

EXIT: PROCEDURE EXTERNAL;
END EXIT;

ERROR: PROCEDURE (ERRNUM) EXTERNAL;
DECLARE (ERRNUM) ADDRESS;
END ERROR;

LINEEDIT#BFR:
PROCEDURE BYTE;
CALL READ (1, L, 1, ACTUAL#COUNT, STATUS);
RETURN L;
END;

CONVERT: PROCEDURE;
NMBR#NAMES = 0;
DO N = 0 TO 2;
   IF (INABFR(N) AND 1FH) = 30H
      THEN NMBR#NAMES = (10 * NMBR#NAMES) + (INABFR(N) - 30H);
   ELSE RETURN;
   END;
END CONVERT;

/*/ START OF MAIN PROGRAM */

/*/ READ FILE NAME FROM LINE EDIT BUFFER */

DO N = 0 TO 15;
FILENAME(N) = 20H;
END;

DO WHILE LINEEDIT#BFR = '/';
END;

FILENAME(0) = L;
N = 1;

DO WHILE LINEEDIT#BFR <> 0DH AND N < 16;
FILENAME(N) = L;
N = N + 1;
END;

/*/ CLEAR BUFFER */

CALL OPEN (AFT#BB., BUCKET, 2, 0, STATUS);
CALL READ (1, BUCKET, 122, ACTUAL#COUNT, STATUS);

/*/ QUERY FOR NUMBER OF NAMES IN FILE */

CALL WRITE (0, MSG1, LENGTH(MSG1), STATUS);
CALL READ (1, INABFR, 3, ACTUAL#COUNT, STATUS);
CALL CONVERT;
CALL OPEN (AFT$IN, FILENAME, READ$ACCESS, 0, STATUS);
IF STATUS <> 0 THEN
   DO;
      CALL ERROR(STATUS);
      CALL EXIT;
   END;
/* READ FILE OF NAMES AND ADDRESSES FROM INPUT FILE */
CALL READ (AFT$IN, MEMORY, (255 * 80), ACTUAL$COUNT, STATUS);
CALL CLOSE (AFT$IN, STATUS);
CALL OPEN (AFT$IN, LINE$PTR, 2, 0, STATUS);
/* SKIP HEADER LINES OF INPUT FILE */
K, N = 0;
DO WHILE N < 3;
   IF MEMORY(K) = 0AH THEN
      N = N + 1;
   K = K + 1;
END;
/* START OF PRINTING ROUTINE */
DO ALL: DO N = 1 TO NMBR$NAMES;
   DO ONE$ROW: DO M = 1 TO 3; /* WRITE 3 LINES OF DATA PER LABEL */
      DO NR$CHARA = 0 TO 99; /* FILL LABEL BUFFER WITH SPACES */
         LABEL$BFR$(NR$CHARA) = 20H;
      END;
   NR$CHARA, NMBR$SPACES, NMBR$TAB = 0;
   FILL$LABEL$BFR: DO WHILE (NMBR$CHARA <> DD) AND (NMBR$SPACES < 2) AND (NMBR$TAB = 0);
      IF MEMORY(K) = 09 /* TEST FOR TAB */
         THEN NMBR$TAB = 1;
      ELSE DO;
         IF MEMORY(K) = '/
            THEN NMBR$SPACES = NMBR$SPACES + 1;
         ELSE NMBR$SPACES = 0;
         LABEL$BFR$(NR$CHARA) = MEMORY(K);
         NR$CHARA = NR$CHARA + 1;
      END;
   K = K + 1;
   END FILL$LABEL$BFR;
   DO L = 1 TO 2;
      CALL WRITE (AFT$IN, LABEL$BFR, DD, STATUS);
   END;
   CALL WRITE (AFT$IN, CRLF, 2, STATUS); /* PRINT THE LINE */
/* FIND THE START OF NEXT FIELD */
   DO WHILE ((MEMORY(K) = '/') OR (MEMORY(K) = 09));
8/78
98  4          K = K + 1;
99  4          END;

100 3      END DO$ONE$ROW;

    /* FIND NEXT LINE */

101 2    NEXT$LINE: DO WHILE MEM$ORY(K) < @AH;
102  3          K = K + 1;
103  3          END;

104 2    NEXT$LABEL: DO L = 1 TO 3;
105  3          CALL WRITE(AFT$IN$. CR$L.F, 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    = 8345H  837D
  VARIABLE AREA SIZE = 0045H  72D
  MAXIMUM STACK SIZE = 000AH  100
  186 LINES READ
  0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
INTEL® USER'S LIBRARY SUBMITTAL FORM

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 Strom 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

© Intel Corporation, 1976
8/78
FIFO - First-in, First-out Buffer Routine

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 MDTA. 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.

CRT and TTY Software Drivers.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Mervin Doda</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>up to 259 bytes</td>
<td>CANADAIR LTD.</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>118 bytes</td>
<td>P.O. Box 6087</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td></td>
<td>Montreal</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
<tr>
<td>MDS Macro Assembler Ver. 1.0</td>
<td>CANADA H3C 3G9</td>
</tr>
</tbody>
</table>
; F I F O

; F F I 0

EQU $080F

EQU $0800

; M N DTA DEFINES THE START OF FIFO MEMORY

ORG $1000

LXI SP, $2000

; INITIALIZE CONTENTS OF LIP = M N DTA.

LXI H, M N DTA

SHLD LIP

; MAIN PROGRAM RECEIVES ONE CHARACTER FROM CI.

; IF THIS CHARACTER IS A SPACE, THE PROGRAM CALLS

; LOAD SUB-Routine IF THE CHARACTER IS NOT A SPACE,

; THE CHARACTER IS PRINTED ON CO AND STORE

; SUB-Routine IS CALLED.

; F F I 0: CALL CI

ANI $7F

CPI $20

JZ FIF02

; NO, CHAR. IS STORED.

MOV C, A

CALL CO

JMP FIF01

; F F I 0: CALL LOAD

LDA BUF

; BUF CONTAINS FIRST-OUT

MOV C, A

; DATA FROM FIFO.

CALL CO

JMP FIF01

; STORE SUB-Routine

; F F I 0: PUSH PSW

; F F I 0: PUSH H

LHLD LIP

; LIP POINTS TO FIFO

MOV M, A

; MEMORY BYTE THAT WILL

INK H

; STORE PRESENT CHARACTER.

SHLD LIP

POP H

POP PSW

RET

; LOAD SUB-Routine

; F F I 0: PUSH B

; F F I 0: PUSH D

; F F I 0: PUSH H

; BUF IS LOADED WITH FIRST-OUT DATA.

LDA M N DTA
LOC | OBJ | SEQ | SOURCE STATEMENT
--- | --- | --- | ---
0130 | 327701 | 53 | STA BUF
54 | ;MNTA IS SUBTRACTED FROM THE CONTENTS OF LIP.
0140 | 117001 | 55 | LXI D, MNTA
0143 | 2A7801 | 56 | LHLD LIP
0146 | AF | 57 | XRA A
0147 | 47 | 58 | MOV B.D
0148 | 7B | 59 | MOV A.E ;FORM 2'S COMPLIMENT
0149 | 2F | 60 | CMA ;OF MNTA.
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 | ADD D
68 ;DIFFERENCE IS STORED IN C.
0151 | 4D | 69 | MOV C.L
0152 | 79 | 70 | ;IF LIP POINTS TO MNTA, SET CARRY AND RETURN.
0153 | FE00 | 71 | CPI 00H
0155 | 072031 | 72 | JZ LD2
73 | ;INITIALIZE MEMORY TO MEMORY TRANSFER.
0158 | 217A01 | 75 | LXI H, MNTA
015B | 117001 | 76 | LXI D, MNTA
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 | 8D | 83 | DCR C
0164 | C2F01 | 84 | JNZ LD1
85 ;DECREMENT CONTENTS OF LIP.
0167 | 2A7801 | 86 | LHLD LIP
0168 | 2B | 87 | DCX H
0168 | 227801 | 88 | SHLD LIP
016E | 87 | 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 | MNTA: DS 256
100 | END

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS
BUFF A 0177  CI  A F003  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
COS: A cassette operating system for the MDS-800

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 compatible with the MDS monitor. The operating system allows file naming, search and directory, INTEL format HEX files, and opening and closing of ASCII files.

Cassette I/O on ports E0 and E1 HEX ) schematic available
Cassette remote control on port E3 HEX )
1 or 2 audio cassette recorders
MDS-800

MDS Monitor Version 2.0 (may be adapted to others)

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer:</th>
<th>Robert A. McCormick</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>.75 Kbytes</td>
<td>Company:</td>
<td>Frye Electronics, Inc.</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>1.25 Kbytes</td>
<td>Address:</td>
<td>PO Box 23391</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td></td>
<td>City:</td>
<td>Tigard</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>ISIS-II MACRO ASS. VER. 2.0</td>
<td>State:</td>
<td>OR 97223</td>
</tr>
</tbody>
</table>
INTEL® USER'S LIBRARY SUBMITTAL FORM

Program Title

INTELLEC MDS MAILING LIST - FORTRAN-80

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
Tractor feed labels (Dennison #42-551-0, 3 1/2 X 15/16

MDS-PRN (line printer)

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:

A11

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
PLOTA

This program plots up to 100 coordinates, by the TTY (of other console device), using 64 columns by 64 lines.
All coordinates must be integer, positive, from 0 to 1023.

Basic Intelec MDS 800, i.e., Central Processor Module, Front Panel Control Module, Monitor Module, one 16K RAM module, connected to a teletype.

Routines CO, CRLF, TI, PARAM, LBYTE from Intelec 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).

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!)

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
<th>Fernando Jordan</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
<td>IPT - AIA</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
<td>P. O. box 7141</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
<td>Sao Paulo</td>
</tr>
<tr>
<td>Assembler/Compiler Used: Intelec MDS Macro Assembler V2.2</td>
<td>State:</td>
<td>Brazil</td>
</tr>
</tbody>
</table>
SORT - GENERAL SORTING PROCEDURES

General sorting procedure: working on complete lines or on fields (e.g. locate file: 1234H - PUB -- NAME)

MDS 800, diskette driver (DD), console, line printer (optional)

ISIS-II, monitor

Any file to be sorted

- SORT :Fi:FILENAME.EXTENSION

Questions (see listing)

:Fi:FILENAME.CLS on disk
and on :CO: or :LP: as well

<table>
<thead>
<tr>
<th>Registers Modified</th>
<th>All</th>
<th>Programmer: Maessen JL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required</td>
<td>1000H (CODE + ISIS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6500H (DATA)</td>
<td></td>
</tr>
<tr>
<td>ROM Required</td>
<td>Monitor</td>
<td></td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used</td>
<td>MDS 8080/8085</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Macro Assembler</td>
<td></td>
</tr>
<tr>
<td>Company</td>
<td>Bell Telephone Mfg C'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ITT</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Bell Telephonelaan 2</td>
<td></td>
</tr>
<tr>
<td>City</td>
<td>B-2440 GEEL</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td>Belgium</td>
<td></td>
</tr>
</tbody>
</table>
MAILING LIST MERGE

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.

MDS 800/220/230
Disc drive

ISIS-II disc operating system version 2.2 or 3.4
Intellec MDS Mailing List (Ref.No. D60)

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.

The system console will show what records are being appended to the destination file from the source file.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer:</th>
<th>Kerry Howell</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>32K RAM</td>
<td>Company:</td>
<td>Almac/Stroum Electronics</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>2K system monitor</td>
<td>Address: #134</td>
<td>18760 NW Rock Creek Circle</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>NA</td>
<td>City:</td>
<td>Portland</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>FORTRAN80, V1.0</td>
<td>State:</td>
<td>Oregon 97229</td>
</tr>
</tbody>
</table>
FILES

This module consists of a group of utility procedures which ease file oriented I/O under ISIS-II.

MDS with Disk System, Console Device

ISIS-II
PL/M 80 Compiler

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer: D.M. Brockman/E. Kozel</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>380 Bytes</td>
<td>Company: Boeing Commercial Airplane Company</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>656 Bytes</td>
<td>Address: P.O. Box 3707 MS:25-09</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>1</td>
<td>City: Seattle</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>PL/M 80</td>
<td>State: Washington 98124</td>
</tr>
</tbody>
</table>
SECTION 8

GAMES

<table>
<thead>
<tr>
<th>REFERENCE NUMBER</th>
<th>PROGRAM</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>*E1</td>
<td>NIM</td>
<td>8-1</td>
</tr>
<tr>
<td>E2</td>
<td>NIM</td>
<td>8-4</td>
</tr>
<tr>
<td>*E3</td>
<td>BLACKJACK</td>
<td>8-6</td>
</tr>
<tr>
<td>*E4</td>
<td>THE WORD GAME</td>
<td>8-8</td>
</tr>
<tr>
<td>E5</td>
<td>GAMBOL</td>
<td>8-11</td>
</tr>
<tr>
<td>*E6</td>
<td>MASTERMIND</td>
<td>8-13</td>
</tr>
<tr>
<td>E7</td>
<td>MAZE</td>
<td>8-18</td>
</tr>
<tr>
<td>E8</td>
<td>GAME OF LIFE</td>
<td>8-20</td>
</tr>
<tr>
<td>E9</td>
<td>NUMBERS</td>
<td>8-26</td>
</tr>
<tr>
<td>E10</td>
<td>KALAH</td>
<td>8-28</td>
</tr>
<tr>
<td>E11</td>
<td>AN ADAPTIVE GAME PROGRAM</td>
<td>8-32</td>
</tr>
<tr>
<td>E12</td>
<td>MATCH GAME</td>
<td>8-35</td>
</tr>
<tr>
<td>*E13</td>
<td>MAZE</td>
<td>8-39</td>
</tr>
<tr>
<td>E15</td>
<td>LEWTHWAITE'S GAME</td>
<td>8-45</td>
</tr>
<tr>
<td>*E16</td>
<td>TIC-TAC-TOE</td>
<td>8-46</td>
</tr>
<tr>
<td>*E17</td>
<td>BANDIT, STATIC DISPLAY VERSION 1</td>
<td>8-48</td>
</tr>
<tr>
<td>E18</td>
<td>KILL THE ROTATING BIT</td>
<td>8-50</td>
</tr>
<tr>
<td>E19</td>
<td>PROMPT PONG</td>
<td>8-54</td>
</tr>
<tr>
<td>E20</td>
<td>REACT</td>
<td>8-56</td>
</tr>
<tr>
<td>E21</td>
<td>SLOT MACHINE</td>
<td>8-62</td>
</tr>
<tr>
<td>E22</td>
<td>MATCH</td>
<td>8-64</td>
</tr>
<tr>
<td>E23</td>
<td>MASTERMIND</td>
<td>8-66</td>
</tr>
<tr>
<td>E24</td>
<td>&quot;MASTERMIND 8080&quot; RUN ON SBC 80/10 PROTOTYPING BOARD</td>
<td>8-68</td>
</tr>
<tr>
<td>E25</td>
<td>LANDER</td>
<td>8-70</td>
</tr>
<tr>
<td>E26</td>
<td>TIC-TAC-TOE - 3 DIMENSIONAL</td>
<td>8-72</td>
</tr>
<tr>
<td>*E27</td>
<td>CRAP'S</td>
<td>8-74</td>
</tr>
<tr>
<td>E28</td>
<td>VDU DARTS</td>
<td>8-76</td>
</tr>
<tr>
<td>E29</td>
<td>HANG</td>
<td>8-78</td>
</tr>
<tr>
<td>E30</td>
<td>BIORIM</td>
<td>8-80</td>
</tr>
<tr>
<td>E31</td>
<td>SLALOM VERSION 1.4</td>
<td>8-82</td>
</tr>
<tr>
<td>E32</td>
<td>HORSEFACE</td>
<td>8-84</td>
</tr>
<tr>
<td>E33</td>
<td>MASTERMIND FOR SDK-86</td>
<td>8-86</td>
</tr>
<tr>
<td>E34</td>
<td>TECH-NEL - FRUIT MACHINE GAME, V1.2</td>
<td>8-88</td>
</tr>
<tr>
<td></td>
<td>(FOR MDS SERIES II)</td>
<td></td>
</tr>
<tr>
<td>E35</td>
<td>MOUSE</td>
<td>8-90</td>
</tr>
</tbody>
</table>

*PROGRAM HAS BEEN CONVERTED TO RUN ON THE SDK-80 AND IS AVAILABLE TO INSITE MEMBERS UPON REQUEST.
THE GAME NIM

Please see attachment

Intellec 8/80 with 4K RAM (Minimum) and TTY (or equivalent) Console.

None - The program runs independently of other software or ROMs. However, a hex tape loader such as in Monitor is required to read in the hex paper tape.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>8080 Assembler Ver. 3.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>700 Hex Bytes</td>
<td>William R. Ott</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Applied Data Communications</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>1509 E. McFadden</td>
</tr>
<tr>
<td></td>
<td>Santa Ana, Ca. 92705</td>
</tr>
</tbody>
</table>
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).
NIM (A computer game)

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.

Intellec 8 Mod 8 with standard TTY connected to ports 0, 1, and 8H.

None.

With program loaded jump to 0100H to run.
Use TTY to input 2 characters for each player turn.

TTY will show
Number of counters in each pile and game status.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Intellec 8 Mod 8</td>
</tr>
<tr>
<td></td>
<td>Macro Assembler Ver 2.0</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer: B. Weston</td>
</tr>
<tr>
<td>260H</td>
<td>Company: Transcom Inc.</td>
</tr>
</tbody>
</table>
| ROM Required:       | Address: 580 Spring Street  
| None                | Windsor Locks, Conn. 06096 |
BLACKJACK

INTELEC/8 PLAYS BLACKJACK WITH OPERATOR

INTELEC/8 MOD 80
TELETYPE

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"?

BLACKJACK IS PLAYED, AND THE RESULTS OF EACH GAME ARE PRINTED.

CONTINUOUS 'BELL' CHARACTERS AT ANY TIME INDICATE THAT THE PROGRAM IS "SHUFFLING" THE CARDS.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>5</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Programmer:</td>
</tr>
<tr>
<td>APROX. 700 HEX</td>
<td>8080 MACRO ASSEMBLER, VER.3.0</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>ALAN ROSENBAUM</td>
</tr>
<tr>
<td>Address: General Microwave Corp.</td>
<td>155 Marine St. - Farmingdale NY</td>
</tr>
</tbody>
</table>
THE WORD GAME

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.

TTY on PORT 0 - 1

Initially 16 words of up to 16 letters in length are input via the teletype and stored in RAM (Locations 900 - 1000H). Thereafter each guess (input via teletype) is stored in 1020 → 1030.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>J. M. Milby</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/8K for Program</td>
<td>B.O.C. Ltd. Small Scale Automation</td>
</tr>
<tr>
<td>Approx. 1/8K for Storage Buffer</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 Morris Rd.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Daventry, Northamptonshire</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intellec 8/Mod80 Macro Assembler V.2.0</td>
<td>ENGLAND</td>
</tr>
</tbody>
</table>
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:-

ARCADE: 0,2
FGHIJKL: 1,0
MNOPQRST: 1,1
UVWXYZ: 0,1
ABC: 0,2
CDE: 2,0

ETCETERA UNTIL CATAPULT: 8,0
AFTER THE CORRECT WORD HAS BEEN INPUT THE COMPUTER SELECTS AGAIN FROM THE SIXTEEN WORD DICTIONARY.
**Program Title**

GAMBOL

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
Mastermind

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 *

Intelec 8, Mod 80, TTY.

TTY input and output routines in the resident monitor, version 1.0.

---

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>PL/M1, version 1.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>110H</td>
<td>M. Cheeseman</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5DOH</td>
<td>AERE HARWELL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Oxfordshire, ENGLAND</td>
</tr>
</tbody>
</table>
/* PROGRAM MASTERMIND. M. CHEESEMAN AUG 1975 */

20: DECLARE (SIAH,ODTS,TRYS,I,J) BYTE ,XN ADDRESS;
00009 DECLARE (TEMP,SEIUP) (4) BYTE;
00010 DECLARE BUFFER(72) BYTE;
00011 DECLARE M1 DATA ("*MASTERMIND*",0CH,0AH,
00012 OAH,"DO YOU KNOW HOW TO PLAY",3FH,": ");
00013 DECLARE M21 DATA (0AH,0AH,"THE COMPUTER SETS UP A ",
00014 "4-DIGIT CODE.",ODH,0AH,"/EACH DIGIT IS IN THE RANGE",
00015 "0 TO 9.",ODH,0AH,"THE GAME IS TO BREAK THE CODE",
00016 "BY ",ODH,0AH,"/TYPING IN A SERIES OF GUESSES",
00017 "FOR ",ODH,0AH,"/EACH GUESS THE PROGRAM RETURNS ",
00018 "THE ",ODH,0AH,"/FOLLOWING INFORMATION: AN * FOR ",
00019 "EACH ",ODH,0AH);
00020 DECLARE M22 DATA ("/CORRECT DIGIT IN CORRECT POSITION",
00021 ",","/ODH,0AH,"/AND A ". FOR EACH CORRECT DIGIT IN A ",
00022 ",","/ODH,0AH,"/WRONG POSITION. E.G. IF THE COMPUTER/
00023 ODH,0AH,"SETS UP 5025 AND THE PLAYER ENTERS ",ODH,
00024 ODH,"/3215 , THE PROGRAM WILL TYPE ",ODH,0AH);
00025 DECLARE M3 DATA (0AH,"PLEASE TYPE A NUMBER FOR THE ",
00026 "RANDOM ROUTINE :");
00027 DECLARE M4 DATA (ODH,0AH,0AH," /");
00028 DECLARE M5 DATA (ODH,0AH,0AH,07H,"CORRECT ",
00029 "YOU TOOK ",07H);
00030 DECLARE M6 DATA ("/ATTEMPTS.",ODH,0AH,"ANOTHER GAME",3FH,": ");
00031 DECLARE M7 DATA (0AH,0AH,0AH,0AH,0AH);
00032 DECLARE M8 DATA ("*****");
00033 DECLARE M9 DATA ("******");
00034 ITYIH:PROCEDURE BYTE;
00036 DJIO 38034H;
00037 END ITYIH;
00038 ITYOUT:PROCEDURE(CHAR);
00040 DJIO 38094H;
00042 END ITYOUT;
00043 RANDOM:PROCEDURE;
00045 DECLARE I BYTE;
00046 DO I=O TO 3;
00047 XK=XK*107;
00048 DATA(XK/10923);
00049 END;
00050 END RANDOM;
00051 MESSAGE:PROCEDURE(N,PTP);
00053 DECLARE (I,N) BYTE;
00054 DECLARE PIR ADDRESS,CHAR BASED PIR BYTE;
00055 IF N=0 THEN
00056 CALL ITYOUT(CHAR(0));
00058 RETURN;
00059 END;
00060 DO I=O TO 4;
CALL TIYOUI(CHAR(I));
END;
00052 2
CALL MESSAGE;
00054 1
READPROCEDURE WHILE;
00057 2
DECLARE (CHAR,1) BYTE;
00060 2
I,CHAR=0;
00067 2
DO WHILE (CHAR<0DH) AND (I<72);
00074 2
CHAR=ITYIN AND 7FH;
00076 2
CALL TIYOUI(CHAR);
00078 2
/* CTRL K */
00080 3
IF CHAR=0AH THEN
00083 3
DO:
00084 3
CALL TIYOUI(23H);
00085 4
IF I>0 THEN I=I-1;
00086 4
GOTO L1;
00087 4
END;
00088 3
/* CTRL L */
00091 3
IF CHAR=0CH THEN
00093 3
DO:
00094 3
CALL MESSAGE(LAST(*M),*M);
00096 1
I=0;
00097 4
GOTO L1;
00098 4
END;
00099 3
BUFFER(I)=CHAR AND 0FH;
00101 3
I=I+1;
00102 3
END;
00105 2
RETURN BUFFER(0);
00108 2
END READ;
00111 2
/* START MAIN PROGRAM PROGRAM */
00114 1
00116 1
CALL MESSAGE(LAST(*M),*M);
00118 1
IF READ=0FH THEN
00120 1
DO:
00122 1
CALL MESSAGE(LAST(*M2),*M2);
00124 1
CALL MESSAGE(LAST(*M2),*M2);
00127 1
END;
00128 1
CALL MESSAGE(LAST(*M3),*M3);
00130 1
I=READ;
00132 1
I=0; I:=1;
00134 1
/* EVALUATE RANDOM NUMBER SEED */
00136 1
L3: IF BUFFER(I)=0DH THEN GOTO L5;
00138 1
XN=XN*10+(15 AND BUFFER(I)));
00140 1
I=I+1;
00141 1
GOTO L3;
00143 1
/* SET UP THE GAME */
00145 1
L5: CALL RANDOM;
00147 1
TRYs,*STARS=0;
00150 1
DO WHILE STARS<4;
00152 1
DO I=0 TO 3;
00154 2
SETUP(I)=TEMP(I);
00156 3
END;
00157 2
CALL MESSAGE(LAST(*M4),*M4);
00160 2
/* READ THE ATTEMPT */
00162 2
I=READ;
00164 2
TRYs=TRYs+1;
00166 2
STARS,DOTS=0;
00168 2
/* SCORE THE ATTEMPT */
DO I=0 TO 3;
  IF BUFFER(I)=SETUP(I) THEN
    DO;
      START=START+1;
      BUFFER(I)=OFFH;
      SETUP(I)=OFFH;
      END;
  END;
DO I=0 TO 3;
  DO J=0 TO 3;
    IF BUFFER(I)=SETUP(J) THEN
      DO;
        DOTS=DOTS+1;
        BUFFER(I)=OFFH;
        SETUP(J)=OFFH;
      END;
    END;
  END;
END;
CALL TTYJUI(0);
CALL TTYJUI(0);
/* OUTPUT THE SCORE */
IF START<>0 THEN CALL MESSAGE(STARS=-1,.M9);
IF DOTS<>0 THEN CALL MESSAGE(DOTS=-1,.M8);
END;
CALL MESSAGE(LAST(M5),.M5);
/* >9 TRIES IS NOT GOOD ENOUGH */
IF TRIES>9 THEN TRIES=21H;
ELSE TRIES=TRIES+10H;
CALL TTYJUI(10H);
CALL MESSAGE(LAST(M6),.M6);
IF READ=9 THEN
  DO;
    CALL MESSAGE(LAST(M7),.M7);
  END;
GOTO 3800H; /* RETURN TO MAIN MENU */
00157 1 EDF
END PROGRAM ERRORS
MAZE

This is a game program to generate random mazes.

8080 System. A line printer can be used, but it is not required.

INTELLECT Monitor. (Any version).
TTY Input (CI), TTY Output (CO), and List Output (L0).

Keyboard inputs (See sample run).

The Maze is printed on the device selected as the LIST device.
LIFE, GAME OF


Intellec/MDS System & Datapoint 3300 CRT

Intellec/MDS Monitor I/O

(X,Y) coordinates of initial cell pattern, entered via CRT keys:

↑, ↓, →, ←, CR, LF, SPACE, AND !

Input is terminated with CTRL-C

Successive generations of cell organism, based on initial pattern and reproduction laws. Very entertaining.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>K. Burgett</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>~ zk bytes</td>
<td>Dharma Systems</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bytes</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bytes</td>
<td>San Jose</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/M, Ver. 2.0</td>
<td>California</td>
</tr>
</tbody>
</table>
/* REF. NO. E8 */
/* PROGRAM NAME GAME OF LIFE */
200H:

/* GAME OF LIFE */

DECLARE BELL LITERALLY '7';
DECLARE CR LITERALLY '0DH';
DECLARE IF LITERALLY '0AH';
DECLARE DOWN LITERALLY '0BH';
DECLARE UP LITERALLY '1AH';
DECLARE LEFT LITERALLY '19H';
DECLARE RIGHT LITERALLY '18H';
DECLARE ERASESEOF LITERALLY '1FH';
DECLARE HOME LITERALLY '1DH';
DECLARE FALSE LITERALLY '0';
DECLARE TRUE LITERALLY '0FFH';
DECLARE BREAKSKEY LITERALLY '5AH';
DECLARE ARRAYSIZE LITERALLY '216';
DECLARE POPULATION(ARRAYSIZE) BYTE;
DECLARE NEXTSGENERATION(ARRAYSIZE) BYTE;
DECLARE FOREVER LITERALLY 'WHILE 1';
DECLARE POPULATIONSADDRESS;
DECLARE GENERATIONADDRESS;
DECLARE MASK DATA (128,64,32,16,8,4,2,1);
DECLARE MSO DATA ('GAME OF LIFE, GENERATION ');
DECLARE MS1 DATA ('POPULATION ');

CI:

PROCEDURE BYTE;
    DECLARE IOC1 LITERALLY '0F803H';
GO TO IOC1;
END CI;

CO:

PROCEDURE (CHAR);
    DECLARE CHAR BYTE;
    DECLARE IOC0 LITERALLY '0F809H';
GO TO IOC0;
END CO;

PRINTSTRING:

PROCEDURE (PNAME,PNLST);
    DECLARE (PNAME,PNLST,Pn1) ADDRESS;
    DECLARE (CHAR BASED PNAME) BYTE;
    DO Pn1 = 0 TO PNLST;
        CALL CO(CHAR(Pn1));
    END;
END PRINTSTRING;

DECIMALSPRN1:

PROCEDURE (VALUE);
    DECLARE VALUE ADDRESS;
    DECLARE POWER ADDRESS;
    POWER = 10000;
    DO WHILE POWER > 0;
IF VALUE >= POWER THEN
    CALL CO((VALUE/POWER) MOD 10) + '0');
    POWER = POWER/10;
END;
END DECENTRALPRINT;

CSTS:
    PROCEDURE BYTE;
    DECLARE IOCSTS LITERALLY 'OF812H';
    GO TO IOCSTS;
END CSTS;

BREAK:
    PROCEDURE BYTE;
    IF NOT CSTS THEN RETURN FALSE;
    IF (CI AND 7FH) <> BREAK$KEY THEN RETURN FALSE;
    RETURN TRUE;
END BREAK;

PROCCREATE:
    PROCEDURE (X,Y);
    DECLARE (X,Y) BYTE;
    DECLARE (WORD,PLACE) BYTE;
    IF Y >= 24 THEN RETURN;
    IF X >= 72 THEN RETURN;
    WORD = SHL(Y,3) + Y + (ROR(X,3) AND 1FH);
    PLACE = MASK(X AND 7);
    NEXT$GENERATION(WORD) = NEXT$GENERATION(WORD) OR PLACE;
    POPULATIONS$COUNT = POPULATIONS$COUNT + 1;
END PROCCREATE;

CENSUS:
    PROCEDURE (X,Y) BYTE;
    DECLARE (X,Y) BYTE;
    IF Y >= 24 THEN RETURN FALSE;
    IF X >= 72 THEN RETURN FALSE;
    Y = POPULATION(SHL(Y,3)+Y+(ROR(X,3) AND 1FH));
    X = MASK(X AND 7);
    IF (Y AND X) = 0 THEN RETURN FALSE;
    RETURN 1;
END CENSUS;

SETUP:
    PROCEDURE;
    DECLARE (X,Y,CHAR) BYTE;
    DECLARE SIZE BYTE;
    CALL CO(HOME);
    POPULATIONS$COUNT = 0;
    DO SIZE = 0 TO LAST(NEXT$GENERATION);
    NEXT$GENERATION(SIZE) = 0;
    END;
    CALL CO(ERASE$EOF);
    X,Y = 0;
    DO WHILE (CHAR=(CI AND 7FH)) <> BREAK$KEY;
    IF CHAR = CR THEN X = 0;
ELSE
  IF CHAR = LF AND Y <> 23 THEN Y = Y + 1;
  ELSE
  IF CHAR = DOWN AND Y <> 23 THEN Y = Y + 1;
  ELSE
  IF CHAR = UP AND Y <> 0 THEN Y = Y - 1;
  ELSE
  IF CHAR = LEFT AND X <> 0 THEN X = X - 1;
  ELSE
  IF CHAR = RIGHT AND X <> 71 THEN X = X + 1;
  ELSE
    IF CHAR = '*' AND X <> 71 THEN
      DO;
      CALL PROCREATE(X,Y);
      X = X + 1;
      END;
    ELSE
      IF CHAR = ' ' AND X <> 71 THEN X = X + 1;
      ELSE CHAR = BELL;
      CALL CO(CHAR);
      END;
      CALL CO(HOME);
      END SETUP;
      DISPLAY:
      PROCEDURE;
      DECLARE SIZE BYTE;
      DECLARE (X,Y,COLUMN) BYTE;
      DO SIZE = 0 TO LAST(NEXTSGENERATION);
      POPULATION(SIZE) = NEXTSGENERATION(SIZE);
      NEXTSGENERATION(SIZE) = 0;
      END;
      CALL CO(ERASESEOF);
      DO Y = 0 TO 23;
      COLUMN = 71;
      DO WHILE NOT CENSUS(COLUMN,Y) AND (COLUMN <> 0);
      COLUMN = COLUMN - 1;
      END;
      IF COLUMN <> 0 THEN
        DO X = 0 TO COLUMN;
        IF CENSUS(X,Y) THEN CALL CO('**');
        ELSE CALL CO(' ');
        END;
        CALL CO(CR);
        END;
        CALL CO(LF);
        END;
        CALL PRINTSTRING(.MS0,LAST(MS0));
        CALL DECIMALSPRINT(GENERATION);
        CALL PRINTSTRING(.MS1,LAST(MS1));
        CALL DECIMALSPRINT(POPBULATIONS COUNT);
        CALL CO (HOME);
        END DISPLAY;
        GENERATE:
        PROCEDURE;
        DECLARE (SIZE,X,Y,CELL) BYTE;
DO SIZE = 0 TO LAST(NEXT$GENERATION);
NEXT$GENERATION(SIZE) = 0;
END;

POPULATION$COUNT = 0;
DO X = 0 TO 71;
DO Y = 0 TO 23;
   CELL = CENSUS(X+1,Y)
      + CENSUS(X+1,Y+1)
      + CENSUS(X,Y+1)
      + CENSUS(X-1,Y+1)
      + CENSUS(X-1,Y)
      + CENSUS(X-1,Y-1)
      + CENSUS(X,Y-1)
      + CENSUS(X+1,Y-1);
   IF (CELL = 3) OR (CENSUS(X,Y)+CELL = 3) THEN
      CALL PROCREATE(X,Y);
   END;
END;

END;

CALL CO(HOME);
END GENERATE;

/* MAIN LOOP */

DO FOREVER;
   GENERATION = 1;
   CALL SETUP;
   DO WHILE (POPULATION$COUNT <> 0) AND NOT BREAK;
   CALL DISPLAY;
   GENERATION = GENERATION + 1;
   CALL GENERATE;
   END;

END;

EOF

NO PROGRAM ERRORS
NUMBERS

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.

It takes a teletype on ports 0 and 1, as the Intellec system does.

It is a stand alone program.

Type in numeric guesses from 1 to 100, terminating each with a carriage return.

Error messages for incorrect guesses, and congratulatory messages for correct ones.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler/Compiler Used:</td>
<td>8080 Macro Assembler, Ver 3.0</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>1K</td>
</tr>
<tr>
<td>Programmer:</td>
<td>Bernard R. Greening</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>None</td>
</tr>
<tr>
<td>Company:</td>
<td>Peoples Computer Company</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>4 levels</td>
</tr>
<tr>
<td>Address:</td>
<td>P.O. Box 310 Menlo Park, CA 94025</td>
</tr>
</tbody>
</table>
KALAH
PLAYS THE GAME OF KALAH AGAINST A USER

PREFERRABLY AN OMRON 8025 CRT TERMINAL, BUT ANY OTHER 8008-BASED
COMPUTER WILL DO; 2K BYTES MEMORY FOR PROGRAM; 2K \text{ bytes} FOR
DISPLAY (OPTIONAL)

THE WITS OF THE USER

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: State: CALIFORNIA

PL/M 8008 VERSION 3.2
DESCRIPTION OF THE GAME

KALAH IS PLAYED ON A BOARD WHICH CONSISTS OF FOURTEEN SHALLOW INDENTATIONS (CALLED "PITS") ARRANGED AS FOLLOWS:

```
 13  12  11  10  9  8  7  6  5  4  3  2  1  14
```

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 EMPTYED 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 OPPONENT'S 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.
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

© Intel Corporation, 1976
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 is 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.
**Match Game**

See additional sheet

**Intellec 8, TTY on port 0 and 1**

Routines in resident monitor, version 1.0, TTY input, TTY output, carriage return and linefeed, convert ASCII to HEX

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.

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

**Company:**

None

**Address:**

291 Waples Estates
Fairfax, Va, 22030

**Maximum Subroutine Nesting Level:**

2
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.
PROGRAM TITLE: Match Game

<table>
<thead>
<tr>
<th>Matches Remaining</th>
<th>Removed by Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 (Lose)</td>
</tr>
</tbody>
</table>

Under some conditions, as directed by the "ODD" subroutine, the program will take 2, instead of the number indicated, for 10 - 14 matches remaining.
Maze

User attempts to find his way out of an invisible (i.e. dark) maze.

TTY on ports 0 and 1.

System monitor version 3.0,

Characters from TTY. Instructions contained in program.

Messages from program.

Registers Modified:

A11

Assembler/Compiler Used:

8080 Macro Assembler version 3.0

RAM Required:

0000H - 0300H

Programmer:

J.B. Miller-Smith

ROM Required:

System monitor, Version 3.0

Company:

Racal-Milgo Ltd.

Maximum Subroutine Nesting Level:

4

Address:

Portman House, Loverock Rd.
Reading, Berks., ENGLAND
; 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

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 FFFFFF44 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 FF54C64 DB 0FFH, 54H, 5CH, 64H, 0FFH, 0FFH
0064 52 FFFF
0066 60FFFF68 DB 60H, 0FFH, 0FFH, 68H
0068 FF0F64FF DB 0FFH, 0FFH, 64H, 0FFH
0100 ORG 0100H
0102 3F72 TI EQU 3F72H ; SYSMON I/P ROUTINE
0109 3809 CO EQU 3809H ; " O/P "
0080 POSN EQU 0080H ; CURRENT POSITION ADDRESS
0100 313F00 START: LXI SP, 003FH
0102 32E40 MVII A, 40H ; SET...
0105 3238000 STA POSN ; INITIAL POSITION
0108 293401 LXI H, MSG0
010B CD7201 CALL OUTPT ; O/P INTRO' MSG.
010E CD723F 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 34701 JMP MOVE ; POINTED & 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 C22A01 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 ; 0/P "OK" MSG.
0163 C30E01 JMP ENTRY ; NEXT TRY
0166 218002 WALL: LXI H, MSG3
0169 CD7801 CALL OUTPT ; 0/P "WALL" MSG.
016C C30E01 JMP ENTRY ; TRY AGAIN
016F 219002 EXIT: LXI H, MSG4
0172 CD7801 CALL OUTPT ; 0/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 0/P
0180 23 INX H ; POINT TO NEXT CHAR.
0181 C37801 JMP OUTPT ; LOOP

0184 0D0A0A0A MSG0: DB 0DH, 0AH, 0AH, 0AH
0189 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 4E45454C DB 'FEEL YOUR WAY OUT BY SPECIFYING MOVES NORTH,
`IS 8080 MACRO ASSEMBLER, v1.0`

```
01B9 20594F55
01BD 52205741
01C1 59204F55
01C5 54204259
01C9 20532045
01CD 43494659
01D1 494E4720
01D5 4D4F5645
01D9 53264E4F
01DD 5254482C
01E1 000A  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
    2E 000A  DB 0DH, 0AH
0210 49462048  DB 'IF HOPELESSLY LOST, TYPE "L" - GAME RESTARTS.'
0214 4F50454C
0218 4553534C
021C 59204C4F
0220 53542C20
0224 54595045
0228 20222C22
022C 202D2047
0230 414D4520
0234 52453354
0238 41525453
023C 2E
0240 000A  DB 0DH, 0AH
0244 474F4F44  DB 'GOOD LUCK!'
0248 204C5543
024C 4B2021
024F 000A0A0A  DB 0DH, 0AH, 0AH, 0AH
0252 00  DB 00H
0255 2020304E  MSG1:  DB '/ N.S.E.W (OR L) ONLY PLEASE - RETYPE'
0258 20532C45
025B 2C572028
025F 4F52204C
0263 45592050
      57 4C454153
026B 4D202D20
026F 52455459
```
0272 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 0D0A0443 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
Lewthwaite's Game

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.

MDS 800 & ADDS 580 CRT (or any cursor addressable CRT)

Monitor routines

Row & column of the counter to be moved toward the blank cell.

CRT will update the matrix in place and trace the moves.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Less than 3K bytes</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Monitor</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>MDS 800 Macro Assembler V2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer:</td>
<td>Tom McHugh</td>
</tr>
<tr>
<td>Company:</td>
<td></td>
</tr>
<tr>
<td>Address:</td>
<td>3130 N Bartlett Milw. WI. 53211</td>
</tr>
</tbody>
</table>
TIC-TAC-TOE

This game lets two players play against each other, while the machine supervises the game.

Intellec/MDS system & CRT-console

Intellec/MDS monitor I/O

Keyboard inputs (1-9)
Square to be deleted and square to be filled.

The square is drawn on the console with Xs and Os and error messages.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>8080Macro assembler V1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>7B0H</td>
<td>Rune Larsen</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>The Norwegian telecom. research establishment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P.O.B. 83</td>
</tr>
<tr>
<td></td>
<td>N-2007 Kjeller NORWAY</td>
</tr>
</tbody>
</table>
Bandit, static display version 1.

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.

MDS80 and Hazeltine 1200 VDU; else any system with a VDU and a CO and CI routine. NB Following ASCII 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.

Resident MDS800 system monitor or other suitable CO & CI software.

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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>8080 Macro Assembler</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000H to 2952H for programme. IFF4H to IFFFH for stack</td>
<td>P.G.R. Kitson MSc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Marconi Radar Systems Ltd.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>New Parks, Leicester, England</td>
</tr>
</tbody>
</table>
Kill the Rotating Bit

Demo Game for Prompt 80
Two versions of the same game are given. One version uses the exclusive-or instruction and is therefore shorter.

Prompt 80

Prompt 80 monitor which does the initialization of Port E8 and E9.

The bit (or bits) input through Port E9 of the Prompt 80.

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
; 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 DESTROY. 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
00E8     SWITCH  EQU  0E8H  ; INPUT SWITCHES
00E9     LGHTS   EQU  0E9H  ; OUTPUT LIGHTS
;
; REGISTER USE
;
; B-LIGHT PATTERN
; H-HIGH ORDER DELAY TIME
; L-LOW ORDER DELAY TIME
;
; INITIALIZE E 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 COMPLEMENT 
; OF OUTPUT.
3D0F D3E8      OUT    LGBTS ; OUTPUT TO LIGHTS
;
; READ SWITCHES FOR INPUT PATTERN
;
3D11 D8E9      IN     SWTCH ; READ SWITCHES FOR PATTERN
3D10 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
PROMPT PONG

A GAME OF PONG USING THE PROMPT 80

PROMPT 80

PROMPT 80 MONITOR

TWO PLAYERS ONE DEPRESSING THE "," KEY AND ONE DEPRESSING "3" KEY OF THE PROMPT AT THE CORRECT TIME.

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.1

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
REACT

Game to test your reaction time

MDS, Output console with "bell" that is audible.

MAC80 Assembler or Cross Assembler

Finger presses interrupt 7 button ASAP after hearing bell. Occurance of bell is randomized in time between 1 and 2 seconds.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>All</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>3 bytes for VSA</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Monitor + 345 bytes</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>2</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>Mac 80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programmer:</th>
<th>V. Grinich</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company:</td>
<td>Stanford University</td>
</tr>
<tr>
<td>Address:</td>
<td>101 GRL</td>
</tr>
<tr>
<td>City:</td>
<td>Stanford</td>
</tr>
<tr>
<td>State:</td>
<td>California 94305</td>
</tr>
</tbody>
</table>
; 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, LME51 AND 0FFH
3111 21C131 LXI H, M51
3114 4E RPT1: MOV C, M
3115 CD09F8 CALL CO
3118 23 INX H
3119 1D DCR E
311A C21431 JNZ RPT1
311D FB NWRTRY: 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

CNDTN:
3128 2140D01 LXI H, 10000/30 ; 10000 USEC=10 MILLISEC
; DELAY LOOP IS 30 USEC
312B CDAD31 CALL DELAY
312E CD8631 CALL COASC ; PRINT OUT 9, 8, . . . , 1, 0
3131 1D DCR E
3132 F22831 JP CNDTN
3135 CDA231 CALL CRLF
3138 211482 LXI H, 10000/30*100 ; 1000 MILLISEC
13B CDAD31 CALL DELAY ; GET ONE SEC OF DELAY
413E 2ABE31 LHLN L0
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 DISPLAYED
3158 C25C31 JNZ MORE
315B 76HLT ; 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
SRVCE7:
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. NIBBLE TO ASCII VALUE
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
318D E60F ANI 0FH ; GET L. O. NIBBLE INTO ASCII VAL NOW
318F 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
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 85 ORA L ; IF BOTH H AND L ARE =0 ACC IS =0
31B2 C2AD31 JNZ MORE1
31B5 C9 RET

; COSC:
31B6 7B MOV H, 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 COUNTDOWN (9, 8, ..., 1, 0) THERE WILL BE
31C5 5220434F
31C9 554E444F
31CD 574E2028
31D1 3920382C
31D5 2E2E2E31
31D9 2C302920
31DD 544E4552
31E1 45205749
31E5 4C4C2042
31E9 4520
31EB 41203120  DB 'A 1 SEC PAUSE, THEN A BELL.'
31EF 53454320
31F3 50415553
31F7 452C2054
31FB 48454420
31FF 41204245
3203 4C4C2042
3207 000A  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
3231 494F4E
234 2054494D  DB 'TIME IN TENS OF MILLISECONDS.'
3238 4520494E
323C 2054454E
3240 53204F46
3244 204D494C
3248 4C495345
324C 434F4E44
3250 532E
3252 000A  DB CR,LF
0093  LMES1 EQU $-MES1
3100  END ENTRY
8080 SLOT MACHINE

Simulates 3 Wheel Slot Machine
Described in Book
    Game playing with computers
    by Donald Spencer

Written for Intellec 8/MOD 80 with
TTY on Ports 0 and 1 or
CRT on Ports 4 and 5

I/O routines in System Monitor Ver. 3.0

(see sample printout)

Registers Modified:
    ALL

RAM Required:
    686 hex bytes

ROM Required:

Maximum Subroutine Nesting Level:
    4

Assembler/Compiler Used:
    MACRO ASSEMBLER V 3.0

Programmer:
    Mark D. Hansen

Company:
    ISCO

Address:
    P. O. Box 5347

City:
    Lincoln

State:
    Nebraska 68505
MATCH (Resembles Invicta Corp's "Mastermind")

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.

Intellec 8/Mod 80, Teletype on Port 0 and 1

Program is self-contained.

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.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C, D, E, H, L</td>
<td>PATRICK F, CASTELAZ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 Bytes (includes 29 for stack)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
</tr>
</thead>
<tbody>
<tr>
<td>957 Bytes (1K)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intellec 8 MACRO Assembler</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marquette University - Engineering</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1515 West Wisconsin Avenue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milwaukee</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wisconsin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>53233</td>
</tr>
</tbody>
</table>

4/77
MASTERMIND 8080

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.

INTEL SBC 80/10 with TTY interface.
Alternatively INTELLEC MDS-800 with TTY or CRT interface.

Console Input/Output routines resident in the System Monitors.

To begin program execution, enter 'G3C3D'. To restart a game, enter any character via the keyboard. The computer then prompts the player to select:

1) the length of code (i.e. number of columns) to be guessed.
   - Enter one digit in the range 1 to 6.
2) 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.
3) 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'.

After each attempt at guessing the code, the computer replies by outputting:

1) the number of digits correct in both value and position.
2) 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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>C. J. Williams</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>39F (HEX) = 927 bytes program</td>
<td>Cable &amp; Wireless Ltd.</td>
</tr>
<tr>
<td>20 (HEX) = 32 bytes workspace</td>
<td></td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td></td>
<td>114 Great Suffolk St.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td>1</td>
<td>London, SE1.</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
<tr>
<td>ISIS 8080 Macro Assembler V1.1</td>
<td>ENGLAND</td>
</tr>
</tbody>
</table>
"MASTERMIND 8080" Run on SBC 80/10 Prototyping Board

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.

INTEL SBC 80/10 with TTY interface. Alternatively INTELLEC MDS-800 with TTY or CRT interface.

Console Input/Output routines resident in the System Monitors.

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'.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>C. J. Williams</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>39F(HEX) = 927 bytes program</td>
<td>Cable &amp; Wireless Ltd.</td>
</tr>
<tr>
<td>20(HEX) = 32 bytes workspace</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>114 Great Suffolk St.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>London, SE1.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISIS 8080 Macro Assembler V1.1</td>
<td>ENGLAND</td>
</tr>
</tbody>
</table>
LANDER

A MOON LANDER GAME WHICH IS PLAYED ON THE INTEL PROMPT 48.

PROMPT 48

RESIDENT MONITOR SUBROUTINES

KEYBOARD ENTRIES OF:
Ø - 9 FOR THRUST,
A FOR ALTITUDE DISPLAY,
F FOR REMAINING FUEL DISPLAY.

DISPLAY SHOWS VELOCITY AND ALTITUDE OR REMAINING FUEL.
VELOCITY IS "FROZEN" AT TOUCHDOWN.

Registers Modified: A,R0,R1,R2,R6,R7,F0,
R0',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
TIC TAC TOE - 3 Dimensional

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.

no special hardware required

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

PROGRAM COMMUNICATES WITH TTY ONLY

GAME MATRIX AFTER EACH ACTION

X=HUMAN ACTION
O=COMPUTER ACTION

P4 J2

<table>
<thead>
<tr>
<th></th>
<th>ABCD</th>
<th>EFGH</th>
<th>IJKL</th>
<th>MNOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-----</td>
<td>X----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>2</td>
<td>-----</td>
<td>-CX--</td>
<td>-O--</td>
<td>-----</td>
</tr>
<tr>
<td>3</td>
<td>-----</td>
<td>--O--</td>
<td>--X--</td>
<td>-----</td>
</tr>
<tr>
<td>4</td>
<td>-----</td>
<td>--O--</td>
<td>-----</td>
<td>-X-X</td>
</tr>
</tbody>
</table>

Registers Modified: all

RAM Required: 1 k Bytes

ROM Required: none except for monitor routines

Maximum Subroutine Nesting Level: 4

Assembler/Compiler Used: INTELLEC 8/80 Macro Assembler

Programmer: R. Vogel/W. Vollenweider

Company: HAMDATA

Address: Bahnhofstrasse 46c
           CH-8305 Dietlikon Switzerland
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)

To start the game, type G C00 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

RAM Required:
20 Bytes

ROM Required:
1K ROM #3

Maximum Subroutine Nesting Level:
8

Assembler/Compiler Used:
8080 Assembler

Programmer:
VAN HERNDON AND DAVE YONLICH

Company:
J. M. PERRY INSTITUTE

Address:
2011 West Washington Avenue

City:
Yakima

State:
Washington 98903

© Intel Corporation, 1976
MCS-80 KIT

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 ESTABLISHING YOUR 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
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 201. (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
Program Title: HANG

Function: Hangman Word Guess Game

Required Hardware:
- MDS-800
- MDS-20S
- 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

RAM Required:
734 Bytes

ROM Required:

Maximum Subroutine Nesting Level:
12 Bytes

Assembler/Compiler Used:
ISIS 8080 Macro Assembler, V1.1

Programmer:
Bernard J. Verreau

Company:
NCR Corporation

Address:
P.O. Box 607 Mitchell Rd.

City:
Millsboro

State:
Delaware 19966

© Intel Corporation, 1976
12/6/77
BIORIM

To print a graphical calendar plotting the operator's 'BIORHYTHM'

MDS 800, Console Input, Line Printer, Disk Drive.

PL/M-80, ISIS II.

Operator's name and date of birth.
On loading under ISIS II program supplies its own instructions.

Graphic calendar.
Information about Biorhythms is now quite widely published.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: David N. Larkins</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: R.C.A. Jersey Ltd.</td>
</tr>
<tr>
<td>64 K Bytes</td>
<td>Address: Rue des Pres, Longueville,</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>City: JERSEY</td>
</tr>
<tr>
<td>None</td>
<td>State: Channel Islands.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level: 8</td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used: PL/M-80 V3.0</td>
<td></td>
</tr>
</tbody>
</table>
Slalom Version 1.4

Game
Swiss Ski Champions Worldcup

MDS with Disk and CRT (9600 recommended)
Line Printer optional

ISIS-II

Numeric Keys 1 to 9 on CRT for Tabulation
Text in Dialog

on CRT
optional on Line Printer

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: Ulrich E. Spörri</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>ISIS + 10K</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company: UES electronics &amp; software</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>City: CH 8143 Stallikon</td>
</tr>
<tr>
<td></td>
<td>State: Switzerland</td>
</tr>
</tbody>
</table>
Program Title

HORSE RACE (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 compatible 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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>All</th>
<th>Programmer: Kerry Howell</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>48K</td>
<td>Company: Almac/Stroum</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>2K MDS System Monitor</td>
<td>Address: 18760 NW Rock Creek Circle</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>N/A</td>
<td>City: Portland</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>Fortran-80 V1.0</td>
<td>State: Oregon 97229</td>
</tr>
</tbody>
</table>
MASTERMIND FOR SDK-86

A game of logic

**RULES**

The computer sets up a 4-digit code. Each digit is in the range 0 to 9. 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 *

SDK-86

PLM86 or ASM86, LINK-86, LOQ86, OH86, SDK86, SDKIOS, LIB

(This program is available in PL/M86 or ASM86. Please specify version when ordering. Each version is considered one program.)

<table>
<thead>
<tr>
<th>Registers Modified</th>
<th>All</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required</td>
<td>5K or SDK-86 Board</td>
<td>Company:</td>
</tr>
<tr>
<td>ROM Required</td>
<td>None</td>
<td>Address:</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>PLM86 or ASM86</td>
<td>City:</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1978
Program Title
TECH-NEL FRUIT MACHINE GAME, V1.0
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
MOUSE

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.

MDS 800, CRT /MiniBee/ cursor commands: ESC,H - home pos., ESC,J-screen erasing, ESC,A-up, ESC,B-down, ESC,C-right, ESC,D-left.

CI and CO from MONITOR and subroutines from SYSTEM.LIB.

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.

On the screen is displayed the outline of the maze with an exit and chosen gate. Free movement is displayed as "+" and wall as "Ω" with a bell. When the exit is successfully reached the score /steps and bumps/ is displayed.

---

Registers Modified:

| ALL |
---|---|

RAM Required:

| cca 3.2K |
---|---|

ROM Required:

| None |
---|---|

Maximum Subroutine Nesting Level:

| 8 |
---|---|

Assembler/Compiler Used:

| HL/M-80 V3.0 |
---|---|

Programmer:

| Dalibor NMEC |
---|---|

Company:

Address:

City:

145 00 PRAHA-4, Michle

State:

Czechoslovakia
## SECTION 9

### RESIDENT LANGUAGE TRANSLATORS

<table>
<thead>
<tr>
<th>REFERENCE</th>
<th>PROGRAM</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>SMAL: SYMBOLIC MICROCONTROLLER ASSEMBLY LANGUAGE.</td>
<td>9-1</td>
</tr>
<tr>
<td>F2</td>
<td>ML80: STRUCTURED ASSEMBLER FOR 8080</td>
<td>9-3</td>
</tr>
<tr>
<td>F7</td>
<td>LLL BASIC-II INTERPRETER.</td>
<td>9-7</td>
</tr>
<tr>
<td>F8</td>
<td>OCTAL DEBUGGING PROGRAM CODT FOR THE MCS-80 COMPUTER.</td>
<td>9-9</td>
</tr>
<tr>
<td>F9</td>
<td>4040 CROSS ASSEMBLER FOR INTELLE8/80 MOD 80 AND MDS-800.</td>
<td>9-11</td>
</tr>
<tr>
<td>F10</td>
<td>ASM08 MACRO ASSEMBLER.</td>
<td>9-13</td>
</tr>
<tr>
<td>F11</td>
<td>8080 MACRO ASSEMBLER VERSION 4.1.</td>
<td>9-15</td>
</tr>
<tr>
<td>F12</td>
<td>8088 MACRO ASSEMBLER VERSION 2.0.</td>
<td>9-17</td>
</tr>
<tr>
<td>F13</td>
<td>SEQUENTIAL PASCAL COMPILER PAS80 VERSION 1.0.</td>
<td>9-19</td>
</tr>
<tr>
<td>F14</td>
<td>RIA80</td>
<td>9-23</td>
</tr>
<tr>
<td>F15</td>
<td>LLL/CHERNACK BASIC INTERPRETER.</td>
<td>9-25</td>
</tr>
<tr>
<td>F16</td>
<td>PILOT-80 ISIS-II VERSION 2.0.</td>
<td>9-28</td>
</tr>
<tr>
<td>F17</td>
<td>LISP INTERPRETER.</td>
<td>9-29</td>
</tr>
<tr>
<td>F18</td>
<td>ASM - ON-LINE ASSEMBLER.</td>
<td>9-31</td>
</tr>
<tr>
<td>F19</td>
<td>8086 TINY BASIC.</td>
<td>9-33</td>
</tr>
<tr>
<td>F20</td>
<td>STAGE 2</td>
<td>9-35</td>
</tr>
</tbody>
</table>
SMAL: A Symbolic Microcontroller Assembly Language

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.

Intellec 8 or Intellec /Mod 80 (or equivalent) with at least 8K

No additional software is required for run-time support, although the 8008 or 8080 resident monitor is useful for loading the program.

Assembler language source, along with editing commands

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Gary A. Kildall</td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td>RAM Required:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable - 8K nominal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROM Required:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unknown - Stack size 16 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL/M (8080, Vers 1.0)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: Luiz Pedroso (contact Gary Kildall)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Company: Naval Postgraduate School</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>12K (excluding DOS)</td>
<td>City: Monterey</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>State: California 93940</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
**Program Title:**
BASIC/M Translator and Interpreter

**Function:**
Translator compiles BASIC/M programs to pseudo machine code; interpreter loads and executes the pseudo 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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: T.J. Logan and K.A. Kildall</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company: Digital Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address: Box 579</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City: Pacific Grove</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State: Ca. 93950</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/M and 8080 Macro Assembler</td>
<td></td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
LLL BASIC-II INTERPRETER

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.

8080 Microcomputer System Terminal Input
Device independent I/O

Manual supplied with program

Revised 12/78

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
<th>Programmer: Eugene Fisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>As required for user space</td>
<td>Company: Lawrence Livermore Lab.</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>8K byte</td>
<td>Address: Box 808 L-403</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td></td>
<td>City: Livermore</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>ISIS-II ASM-80</td>
<td>State: CA 94550</td>
</tr>
</tbody>
</table>
OCTAL DEBUGGING PROGRAM (ODT-80) FOR THE MCS-80 COMPUTER

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.

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

None

See attached description.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Eugene Fisher</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack in ADR $010_8,000_8$</td>
<td>Lawrence Livermore Laboratory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 Bytes</td>
<td>Box 808 L-403</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>Livermore,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080 Assembler 2.2</td>
<td>Ca. 94550</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
4040 Cross Assembler for Intellec 8/Mod 80 and MDS-800

REVISED 8/8/77

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.

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:

RAM Required:

ROM Required:

Maximum Subroutine Nesting Level:

Assembler/Compiler Used:

Programmer:

Company:

Address:

City:

State:

© Intel Corporation, 1976
ASM08 MACRO ASSEMBLER

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required: 8K</td>
<td>Company:</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
</tbody>
</table>
8080 Macro Assembler Version 4.1

Assembles 8080 Assembly Language Programs

REvised 4/8/77

Intellec MDS

MDS Monitor

An Assembly Language Program on Paper Tape.

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>All</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>8K bytes</td>
<td>Company:</td>
</tr>
<tr>
<td>ROM Required:</td>
<td></td>
<td>Address:</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td></td>
<td>City:</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>PL/M</td>
<td>State:</td>
</tr>
</tbody>
</table>
8008 Macro Assembler Version 2.0
Assembles 8008 Assembly Language Programs

Intellec 8/Mod 80

8008 Monitor I/O
Assembly Language Program File

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>All</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>8K bytes</td>
<td>Company:</td>
</tr>
<tr>
<td>ROM Required:</td>
<td></td>
<td>Address:</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td></td>
<td>City:</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>PL/M 8008</td>
<td>State:</td>
</tr>
</tbody>
</table>
PAS80 Ver 1.0, Sequential PASCAL Compiler

To provide a sequential PASCAL compiler and virtual machine implementation for the Intel 8080A microcomputer.

Intel Intellec Microcomputer Development System (MDS) with:
64K RAM
Dual Floppy Disks

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.

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.

Compiled listings and virtual machine code files are produced by the sequential PASCAL compiler.

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, particularly 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.
RIA80

Interpretive Assembler that runs on an Intellec 8/MOD80 or on any SBC system with an SBC-910 system monitor. (See additional sheets.)

Intellec 8/MOD80 or any SBC system with an SBC-910 system monitor, e.g. system 80/10 or SBC-80P.

Teletype.

Intellec system monitor vers. 3.0 or SBC-910 system monitor.

Starting address of the program to be assembled.

Assembly language program.

Assembled program in RAM at address specified.

Complete listing of address, machine code, and assembly language mnemonic for each instruction.

REVISED 4/78

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>M. A. PORDES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Bytes + Stack</td>
<td>GEC Hirst Research Centre</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1740 Bytes (Intellec)</td>
<td>East Lane, Wembley, Middx.</td>
</tr>
<tr>
<td>1751 Bytes (SBC)</td>
<td>City:</td>
</tr>
<tr>
<td></td>
<td>London</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS 8080/8085 Assembler V1.0</td>
<td>ENGLAND</td>
</tr>
</tbody>
</table>
RIA80

Interpretive Assembler that runs on an Intellec 8/MOD80 or on any SBC system with an SBC-910 system monitor. (See additional sheets).

Intellec 8/MOD80 or any SBC system with an SBC-910 system monitor, e.g. system 80/10 or SBC-80P.
Teletype.
Intellec system monitor vers. 3.0 or SBC-910 system monitor.

Starting address of the program to be assembled.
Assembly language program.

Assembled program in RAM at address specified.
Complete listing of address, machine code, and assembly language mnemonic for each instruction.
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.
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.
LLL/CHERNACK BASIC INTERPRETER

Modified LLL BASIC to run under ISIS-II, LOAD and LIST programs to ISIS-II filenames.

ISIS-II hardware configuration.

ISIS-II software configuration.

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.

Note: Source available on Diskette only for $35.00.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Charles Chernack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Consultant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1034 East Rose Circle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Los Altos,</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>California 94022</td>
</tr>
</tbody>
</table>
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 cannot 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 TP0 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.
PILOT-80 Ver.2.0 - INTERPRETER

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.

ISIS-II Hardware Configuration

ISIS-II Software Configuration

PILOT-80 program (see PILOT-80 User's Guide)

Program execution

Source available on diskette only for $35.00. Source listing available for $15.00.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer:</td>
<td>JOHN STARKWEATHER &amp; RON WILLIAMS</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>4K &amp; 2K Editor &amp; Program Requirements</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address:</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>City:</td>
</tr>
<tr>
<td>8080/8085 MACRO Assembler V2.0</td>
<td>State:</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
LISP INTERPRETER (8080 resident)

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.

Terminal, up to 32K of RAM addressed below 8000H.

Terminal driver with read and write character entries; monitor program which CALLs interpreter with stack setup, with entry point for interpreter abort.

Functions and arguments in LISP to be interpreted: via terminal input. Except for DEFINE, all functions are required to be pure LISP functions.

Results of applying functions to arguments are listed on terminal.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>Darrel J. Van Buer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to 32K (below address 8000H)</td>
<td>SELF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1731 bytes for program</td>
<td>1522 Brockton Ave., Apt 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level: Limited only by expression complexity</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Los Angeles</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel compatible assembler</td>
<td>California</td>
</tr>
</tbody>
</table>

9-29
ASM - ON-LINE ASSEMBLER

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.

1 k bytes of ROM if an INTEL debugging monitor or similar piece of software is available, and slightly more if not; enough RAM for a minimal stack and to hold assembled code; terminal interface.

Intel monitor would reduce memory requirement but is not necessary.

8080 mnemonics are typed in from the terminal. 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)

Code is assembled directly into memory. No object code is produced.

Available on non-system diskette for $25.00; paper tape for $15.00.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Bruce C. Wright</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>Company:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(as desired by user)</td>
<td>Duke Medical Center</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ROM Required:</th>
<th>Address:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 K byte</td>
<td>Box 3181</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Subroutine Nesting Level:</th>
<th>City:</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (8 bytes)</td>
<td>Durham</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembler/Compiler Used:</th>
<th>State:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDS MACRO Assembler</td>
<td>North Carolina</td>
</tr>
</tbody>
</table>
8086 TINY BASIC

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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer: Bob Glossman</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Company: Intel</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address: SCVI MS6-226 X5169</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City: Santa Clara</td>
</tr>
<tr>
<td>Assembler/Compiler Used: ASM86</td>
<td>State: CA 95051</td>
</tr>
</tbody>
</table>
STAGE 2

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 modelling, but it is suitable for use as a prepass for any language translator for purposes of providing macro capability.

MDS 800 Intellec Microcomputer Development System with console device diskette drive(s) and controller, at least 48K RAM memory.

ISIS-II Operating System

Channel assignments equivalencing STAGE 2 channels with ISIS-II files and devices. This implementation of STAGE 2 supports inie non-dummy I/O channels.

User defined.

Program offered on diskette only for $35
## SECTION 10

### CROSS REFERENCE TO PL/M PROGRAMS

<table>
<thead>
<tr>
<th>REFERENCE NUMBER</th>
<th>PROGRAM DESCRIPTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB19</td>
<td>TERMINAL EDITOR</td>
<td>4-124</td>
</tr>
<tr>
<td>AB21</td>
<td>8080 DIS-ASSEMBLER</td>
<td>4-136</td>
</tr>
<tr>
<td>AB28</td>
<td>PROPORTIONAL POWER CONTROL IMAGE BUILDER</td>
<td>4-165</td>
</tr>
<tr>
<td>AB77</td>
<td>LOAD</td>
<td>4-380</td>
</tr>
<tr>
<td>AB102</td>
<td>MDS BACK TO BACK DATA TRANSFER</td>
<td>4-474</td>
</tr>
<tr>
<td>AB130</td>
<td>FORMFD: FORMFEED TO LINEFEED CONVERSION PROGRAM</td>
<td>4-642</td>
</tr>
<tr>
<td>AB132</td>
<td>ALPHA: AN ALPHABETIZED LISTING</td>
<td></td>
</tr>
<tr>
<td>AB145</td>
<td>TEXT PROCESSOR/TEXT EDITOR</td>
<td>4-708</td>
</tr>
<tr>
<td>AB159</td>
<td>KEYWORD FILE SEARCH FOR ISIS-II ENVIRONMENT</td>
<td>4-739</td>
</tr>
<tr>
<td>AB161</td>
<td>KAPIAR, V1.2 - GENERAL PURPOSE MACROPROCESSOR</td>
<td>4-743</td>
</tr>
<tr>
<td>AB164</td>
<td>DOWNSO - DOWNLOAD FROM SERIES-II TO PROMPT-80/85</td>
<td>4-749</td>
</tr>
<tr>
<td>AC14</td>
<td>MP 8208 A/D CONVERTER ROUTINE</td>
<td>4-277</td>
</tr>
<tr>
<td>AE6</td>
<td>8089 - BREAK 89 --0089 BREAK POINT ROUTINE</td>
<td>4-757</td>
</tr>
<tr>
<td>AE18</td>
<td>DKDUMP</td>
<td>4-733</td>
</tr>
<tr>
<td>BR1</td>
<td>BCD SUM FOR 8080</td>
<td>5-54</td>
</tr>
<tr>
<td>BB22</td>
<td>DOUBLE PRECISION MULTIPLY</td>
<td>5-120</td>
</tr>
<tr>
<td>BB28</td>
<td>RMSTF</td>
<td>5-162</td>
</tr>
<tr>
<td>BB31</td>
<td>FIXED POINT CHEBYSHEV SINE AND COSINE</td>
<td></td>
</tr>
<tr>
<td>BB36</td>
<td>FOR PL/M USERS</td>
<td></td>
</tr>
<tr>
<td>BB39</td>
<td>RANDOM NUMBER GENERATOR</td>
<td></td>
</tr>
<tr>
<td>BB39</td>
<td>PL/M HISTOGRAM PROCEDURE AND RANDOM NUMBER GENERATOR</td>
<td>5-199</td>
</tr>
<tr>
<td>BB40</td>
<td>RANDOM#BITS</td>
<td>5-204</td>
</tr>
<tr>
<td>BC7</td>
<td>FLOATING POINT PROCEDURES</td>
<td>5-91</td>
</tr>
<tr>
<td>BC8</td>
<td>PL/M FLOATING POINT INTERFACE</td>
<td>5-93</td>
</tr>
<tr>
<td>BC9</td>
<td>FLOATING POINT DECIMAL AND HEX FORMAT CONVERSION</td>
<td>5-95</td>
</tr>
<tr>
<td>BC19A</td>
<td>ASCII STRING TO INTEL FLOATING POINT NUMBERS</td>
<td>5-262</td>
</tr>
<tr>
<td>BC19B</td>
<td>INTEL FLOATING POINT NUMBERS TO ASCII STRING</td>
<td>5-268</td>
</tr>
<tr>
<td>BC30</td>
<td>FCONST</td>
<td>5-317</td>
</tr>
<tr>
<td>BC32</td>
<td>FPAL - EXTENDED MATH PACKAGE</td>
<td>5-321</td>
</tr>
<tr>
<td>C28</td>
<td>ALIGN PROGRAM--INTERMEDIATE PASS BETWEEN PL/M PASS 1 AND 2</td>
<td>6-59</td>
</tr>
<tr>
<td>C33</td>
<td>SIM48 - 8048 SIMULATOR</td>
<td>6-74</td>
</tr>
<tr>
<td>D1</td>
<td>QUICKSORT PROCEDURE</td>
<td>7-1</td>
</tr>
<tr>
<td>D18</td>
<td>SKILLSORT</td>
<td>7-39</td>
</tr>
<tr>
<td>D20</td>
<td>TIMESHARING COMMUNICATIONS</td>
<td>7-47</td>
</tr>
<tr>
<td>D26</td>
<td>TELEPROCESSING BUFFER ROUTINE</td>
<td>7-73</td>
</tr>
<tr>
<td>D35</td>
<td>TYPE K.T.C. LINEARIZER (PROCEDURE)</td>
<td>7-97</td>
</tr>
<tr>
<td>D64</td>
<td>FILES - PL/M UTILITY PROCEDURES</td>
<td>7-201</td>
</tr>
<tr>
<td>E6</td>
<td>MASTERMIND</td>
<td>8-13</td>
</tr>
<tr>
<td>E8</td>
<td>GAME OF LIFE</td>
<td>8-20</td>
</tr>
</tbody>
</table>

8/79
E10  KALAH ......................................................... 8-28
E31  SLALOM V1.4 ................................................. 8-82
E33  MASTERMIND 86 .............................................. 8-86
E35  MOUSE ......................................................... 8-90
G1   CLI - RMX-80 COMMAND LINE INTERPRETER ........... 11-1
G3   MINITH - RMX MINIMAL TERMINAL HANDLER .......... 11-8
G6   RMX/80 DEMONSTRATION PROGRAM WITH TIME ....... 11-17
## SECTION 11

### RMX-80

<table>
<thead>
<tr>
<th>REFERENCE NUMBER</th>
<th>PROGRAM</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>CLI - RMX-80 COMMAND LINE INTERPRETER.</td>
<td>11-1</td>
</tr>
<tr>
<td>G2</td>
<td>WRMIN - RMX MINIMAL TERMINAL OUTPUT.</td>
<td>11-3</td>
</tr>
<tr>
<td>G3</td>
<td>MINITH - RMX MINIMAL TERMINAL HANDLER.</td>
<td>11-8</td>
</tr>
<tr>
<td>G4</td>
<td>RMX/80 - BASED KEYBOARD INPUT HANDLER SUBROUTINE</td>
<td>11-10</td>
</tr>
<tr>
<td>G5</td>
<td>RMX/80 DRIVER FOR ISBC534 - HND534</td>
<td>11-15</td>
</tr>
<tr>
<td>G6</td>
<td>RMX/80 DEMONSTRATION PROGRAM WITH TIME</td>
<td>11-17</td>
</tr>
</tbody>
</table>
CLI - RMX-80 Command Line Interpreter

Provides operator control of RMX tasks.

SBC 80/20

RMX/80 Nucleus, Free Space Manager Terminal Handler

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.

RMX 80 Users Guide $5.00.
Available from Literature.

Note: Source available on diskette only.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>KEN BURGETT</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>46</td>
<td>DHARMA SYSTEMS</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>988</td>
<td>21950 MCKEAN RD</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td>2</td>
<td>SAN JOSE</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
<tr>
<td>PL/M, V3.0</td>
<td>CALIFORNIA 95120</td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
12/6/77
WRMIN - RMX Minimal Terminal Output

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.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>THOMAS ROLANDER</td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Company:</td>
</tr>
<tr>
<td>15</td>
<td>DHARMA SYSTEMS</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>74</td>
<td>21950 MCKEAN RD.</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>City:</td>
</tr>
<tr>
<td></td>
<td>SAN JOSE</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>State:</td>
</tr>
<tr>
<td>8080/8085 MACRO ASSEMBLER</td>
<td>CALIFORNIA 95120</td>
</tr>
<tr>
<td>OC</td>
<td>OBJ</td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DATOUT</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>WRMIN</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>CALL</td>
</tr>
<tr>
<td>8</td>
<td>WR0</td>
</tr>
<tr>
<td>9</td>
<td>LXI</td>
</tr>
<tr>
<td>10</td>
<td>LXI</td>
</tr>
<tr>
<td>11</td>
<td>CALL</td>
</tr>
<tr>
<td>12</td>
<td>PUSH</td>
</tr>
<tr>
<td>13</td>
<td>LXI</td>
</tr>
<tr>
<td>14</td>
<td>DAD</td>
</tr>
<tr>
<td>15</td>
<td>MOV</td>
</tr>
<tr>
<td>16</td>
<td>INX</td>
</tr>
<tr>
<td>17</td>
<td>MOV</td>
</tr>
<tr>
<td>18</td>
<td>INX</td>
</tr>
<tr>
<td>19</td>
<td>PUSH</td>
</tr>
<tr>
<td>20</td>
<td>MVI</td>
</tr>
<tr>
<td>21</td>
<td>INX</td>
</tr>
<tr>
<td>22</td>
<td>MVI</td>
</tr>
<tr>
<td>23</td>
<td>INX</td>
</tr>
<tr>
<td>24</td>
<td>MOV</td>
</tr>
<tr>
<td>25</td>
<td>INX</td>
</tr>
<tr>
<td>26</td>
<td>MOV</td>
</tr>
<tr>
<td>27</td>
<td>INX</td>
</tr>
<tr>
<td>28</td>
<td>MOV</td>
</tr>
<tr>
<td>29</td>
<td>INX</td>
</tr>
<tr>
<td>30</td>
<td>MOV</td>
</tr>
<tr>
<td>31</td>
<td>INX</td>
</tr>
<tr>
<td>32</td>
<td>MOV</td>
</tr>
<tr>
<td>33</td>
<td>INX</td>
</tr>
<tr>
<td>34</td>
<td>MOV</td>
</tr>
<tr>
<td>35</td>
<td>WR1</td>
</tr>
<tr>
<td>36</td>
<td>MOV</td>
</tr>
<tr>
<td>37</td>
<td>DRA</td>
</tr>
<tr>
<td>38</td>
<td>J2</td>
</tr>
<tr>
<td>39</td>
<td>PUSH</td>
</tr>
<tr>
<td>40</td>
<td>PUSH</td>
</tr>
<tr>
<td>41</td>
<td>LXI</td>
</tr>
<tr>
<td>42</td>
<td>LXI</td>
</tr>
<tr>
<td>43</td>
<td>CALL</td>
</tr>
<tr>
<td>44</td>
<td>POP</td>
</tr>
<tr>
<td>45</td>
<td>POP</td>
</tr>
<tr>
<td>46</td>
<td>LDA</td>
</tr>
<tr>
<td>47</td>
<td>INX</td>
</tr>
<tr>
<td>48</td>
<td>OUT</td>
</tr>
<tr>
<td>49</td>
<td>DCX</td>
</tr>
<tr>
<td>50</td>
<td>JMP</td>
</tr>
<tr>
<td>51</td>
<td>WR2</td>
</tr>
<tr>
<td>52</td>
<td>POP</td>
</tr>
<tr>
<td>53</td>
<td>POP</td>
</tr>
<tr>
<td>54</td>
<td>CALL</td>
</tr>
<tr>
<td>55</td>
<td>JMP</td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>DSEG</td>
</tr>
</tbody>
</table>
PUBLIC SYMBOLS

ROL7EX D 0000  WRMIN C 0000

EXTERNAL SYMBOLS

ROELVL E 0000  ROOUTX E 0000  ROSEND E 0000  ROWAIT E 0000

USER SYMBOLS

DATOUT A 000EC  ROELVL E 0000  ROL7EX D 0000  ROOUTX E 0000  ROSEND E 0000

WR1  C 0029  WR2  C 0043  WRMIN C 0000

ASSEMBLY COMPLETE, NO ERROR(S)
MINITH - RMX Minimal Terminal Handler

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.

SBC 80/20

RMX/80 Nucleus

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.

See Applications Note AP33.

RMX 80 Users Guide $5.00. Available from Literature.

Note: Source Available on diskette only.

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>THOMAS ROLANDER</td>
</tr>
<tr>
<td></td>
<td>Company:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>RAM Required:</td>
<td>Address:</td>
</tr>
<tr>
<td>67</td>
<td>21950 McKEAN RD</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>City:</td>
</tr>
<tr>
<td>570</td>
<td>SAN JOSE</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>State:</td>
</tr>
<tr>
<td></td>
<td>CALIFORNIA 95120</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td></td>
</tr>
<tr>
<td>PLM/80</td>
<td></td>
</tr>
</tbody>
</table>

© Intel Corporation, 1976
**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 procedures.

**Output Results**
ASCII characters to teleprinter or display.

**Note:** Source Available on diskette only.

**Registers Modified:**
All

<table>
<thead>
<tr>
<th>RAM Required:</th>
<th>37 Bytes + Stack (this module only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM Required:</td>
<td>85 Bytes (this module only)</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>2 (minimum)</td>
</tr>
</tbody>
</table>

| Assembler/Compiler Used: | ISIS-II 8080/8085 Macro Assem. V2.0 |

<table>
<thead>
<tr>
<th>Programmer:</th>
<th>M. H. Gansler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company:</td>
<td>NTI</td>
</tr>
<tr>
<td>Address:</td>
<td>4041 Home Rd</td>
</tr>
<tr>
<td>City:</td>
<td>Bellingham</td>
</tr>
<tr>
<td>State:</td>
<td>Washington 98225</td>
</tr>
</tbody>
</table>
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
RMX/80 - BASED GENERAL KEYBOARD
INPUT HANDLER SUBROUTINE (KIHS)

NOTE - 1) ALL "RQ" - NAMED PARAMETERS ARE
RMX/80 DESIGNATED SYSTEM DIRECTIVES
2) MESSAGES & EXCHANGES ARE IMPLEMENTED
   ACC. TO STD. RMX/80 NOMENCLATURE
3) DISPLAY-TYPE CODE (DTYPE) IS SET:
   65 = NORMAL CHARACTER
   75 = LINE FEED CHARACTER
   76 = CARRIAGE RETURN CHARACTER
   77 = DELETE OR RUBOUT CHARACTER
4) INPUT-TYPE CODE (ITYPE) IS SET:
   80H = NOT SET
   81H = CARRIAGE RETURN

NAME KIHS
PUBLIC KIHS, INPEXEC,ROLSEX,CHAR
EXTRN RQWAIT,RQSEND,RQELVL,DISEXC

CSEG

KIHS:

MVI C,5 ; ENABLE LEVEL 5
CALL RQELVL ; INTERRUPT
LXI B,ROLSEX ; WAIT FOR A
LXI D,0 ; CHARACTER FROM
CALL RQWAIT ; THE KEYBOARD
LXI H,B1 ; SET RETURN
PUSH H ; ON THE STACK
MVI A,80H ; INITIALIZE
STA ITYPE ; INPUT CHARACTER TYPE

IN KBI ; GET THE CHARACTER
STA CHAR ; SAVE CHARACTER
CPI 20H ; IS IT A CONTROL
JL CNTRL ; YES GO DECODE
CPI DEL ; IS IT DELETE ?
JL C3 ; YES, OTHERWISE
MVI A,65 ; SET FOR SUBSEQ.
JMP DISOT ; CHARACTER DISPLAY
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>SEQ</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>002B FE0A</td>
<td></td>
<td>53</td>
<td>CPI LF ; IS CHAR. A LINE FEED?</td>
</tr>
<tr>
<td>002D C23F00</td>
<td>C</td>
<td>54</td>
<td>JNZ C2 ; NO</td>
</tr>
<tr>
<td>0030 3E4B</td>
<td></td>
<td>55</td>
<td>MVI A,75 ; YES, SET CODE &amp; OUTPUT</td>
</tr>
<tr>
<td>0032 322200</td>
<td>D</td>
<td>56</td>
<td>DISOT: STA DTYPE</td>
</tr>
<tr>
<td>0035 010000</td>
<td>E</td>
<td>57</td>
<td>LXI B,DISEXC ; SEND MSG TO THE</td>
</tr>
<tr>
<td>0038 111E00</td>
<td>D</td>
<td>58</td>
<td>LXI D,DISMSG ; DISPLAY EXCHANGE</td>
</tr>
<tr>
<td>003B CD0000</td>
<td>E</td>
<td>59</td>
<td>CALL RQSEND</td>
</tr>
<tr>
<td>003E C9</td>
<td></td>
<td>60</td>
<td>RET</td>
</tr>
<tr>
<td>003F FE0D</td>
<td>C2:</td>
<td>61</td>
<td>CPI CR ; IS IT CARRIAGE RTN?</td>
</tr>
<tr>
<td>0041 C0</td>
<td></td>
<td>62</td>
<td>RNZ ; NO, IGNORE CHAR.</td>
</tr>
<tr>
<td>0042 3E4C</td>
<td></td>
<td>63</td>
<td>MVI A,76 ; YES, SET</td>
</tr>
<tr>
<td>0044 CD3200</td>
<td>C</td>
<td>64</td>
<td>CALL DISOT ; CODE FOR CR &amp; ALSO</td>
</tr>
<tr>
<td>0047 3E81</td>
<td></td>
<td>65</td>
<td>MVI A,81H ; SET INPUT TERMINATE</td>
</tr>
<tr>
<td>0049 321000</td>
<td>D</td>
<td>66</td>
<td>STA ITYPE ; CODE</td>
</tr>
<tr>
<td>004C 010F00</td>
<td>D</td>
<td>67</td>
<td>LXI B,INPMSG ; SEND MSG TO THE</td>
</tr>
<tr>
<td>004F 111900</td>
<td>D</td>
<td>68</td>
<td>LXI D,INPMSG ; INPUT EXCHANGE</td>
</tr>
<tr>
<td>0052 CD0000</td>
<td>E</td>
<td>69</td>
<td>CALL RQSEND</td>
</tr>
<tr>
<td>0055 C9</td>
<td></td>
<td>70</td>
<td>RET</td>
</tr>
<tr>
<td>0056 3E4D</td>
<td>C3:</td>
<td>71</td>
<td>MVI A,77 ; SET DEL CODE</td>
</tr>
<tr>
<td>0058 C33200</td>
<td>C</td>
<td>72</td>
<td>JMP DISOT ; &amp; OUTPUT</td>
</tr>
<tr>
<td>005F C33</td>
<td></td>
<td>73</td>
<td>DSEG</td>
</tr>
</tbody>
</table>

PUBLIC SYMBOLS
| CHAR  | 0023 | INPMSG D 000F | KIHS C 0000 | RQLSEX D 0000 |

EXTERNAL SYMBOLS
| DISEXC E 0000 | ROELVL E 0000 | ROSEND E 0000 | ROWAIT E 0000 |

USER SYMBOLS
| B1 | C 0005 | C2 C 003F | C3 C 0056 | CHAR D 0023 | CNTRL C 002 |
| XSEN D 0000 | DISMSG D 001E | DISOT C 0032 | DTYPE D 0022 | INPMSG D 000 |
| A 0022 | KIHS C 0000 | LF A 000A | ROELVL E 0000 | RQLSEX D 000 |

ASSEMBLY COMPLETE, NO ERRORS
HND534 - RMX/80 Driver for iSBC/534

RMX/80 Driver for the iSBC 534 Communications Expansion Board.

iSBC 80/10, 80/10A, 80/20 or 80/20-4 with iSBC 534

RMX/80 (any version)

none

none

Available on diskette only for $35.00.

Registers Modified:

ALL

Programmer:

Joe Barthmaier & Steve Verleye

Company:

Intel Corp.

Address:

City:

State:

RAM Required:

256

Maximum Subroutine Nesting Level:

20

Assembler/Compiler Used:

PL/M 80 Ver. 3.1
RMX/80 DEMONSTRATION PROGRAM WITH TIME

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.

System 8020 connected to MDS 800 via ICE 80 as per Appendix E, RMX/80 User's Guide.

RMX/80 package for SBC 8020 V.1.2.

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.

The output 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

<table>
<thead>
<tr>
<th>Registers Modified:</th>
<th>Programmer:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM Required:</td>
<td>Michael Sussman</td>
</tr>
<tr>
<td>ROM Required:</td>
<td>Company: Amscor</td>
</tr>
<tr>
<td>Maximum Subroutine Nesting Level:</td>
<td>Address: 2512 N. Velasco</td>
</tr>
<tr>
<td>Assembler/Compiler Used:</td>
<td>City: Angleton</td>
</tr>
<tr>
<td>ISIS; PL/M 80, ASM 80</td>
<td>State: Texas 77515</td>
</tr>
</tbody>
</table>
## CROSS REFERENCE TO 8048 PROGRAMS

<table>
<thead>
<tr>
<th>REFERENCE NUMBER</th>
<th>PROGRAM</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB112</td>
<td>8048 - SEVEN SEGMENT DISPLAY INTERFACE</td>
<td>4-520</td>
</tr>
<tr>
<td></td>
<td>SUBROUTINES - SCAN.</td>
<td></td>
</tr>
<tr>
<td>AB114</td>
<td>8048 TUNE GENERATOR.</td>
<td>4-524</td>
</tr>
<tr>
<td>AB138</td>
<td>UPI-41 8-DIGIT LED DISPLAY CONTROLLER</td>
<td>4-680</td>
</tr>
<tr>
<td>AB139</td>
<td>UPI-41A SENSOR MATRIX CONTROLLER</td>
<td>4-688</td>
</tr>
<tr>
<td>AB140</td>
<td>LRC PRINTER CONTROLLER - AP-27</td>
<td>4-698</td>
</tr>
<tr>
<td>AB141</td>
<td>UPI-41A COMBINATION I/O DEVICE (UNIOD)</td>
<td>4-700</td>
</tr>
<tr>
<td>AB147</td>
<td>8278 KEYBOARD/DISPLAY CONTROLLER - UPI-41A</td>
<td>4-711</td>
</tr>
<tr>
<td>AB148</td>
<td>8295 DOT MATRIX PRINTER CONTROLLER - UPI-41A</td>
<td>4-713</td>
</tr>
<tr>
<td>AB149</td>
<td>OLIVETTI 20-COLUMN PRINTER CONTROLLER - UPI--41</td>
<td>4-715</td>
</tr>
<tr>
<td>AB150</td>
<td>8292 (GP1B CONTROLLER) IMPLEMENTATION ON 8741A</td>
<td>4-717</td>
</tr>
<tr>
<td>AB151</td>
<td>SEND48</td>
<td>4-719</td>
</tr>
<tr>
<td>AB152</td>
<td>OUTIN</td>
<td>4-721</td>
</tr>
<tr>
<td>AB153</td>
<td>REMOTE48</td>
<td>4-723</td>
</tr>
<tr>
<td>AE13</td>
<td>PROGRAM TEST-LOADER.</td>
<td>4-513</td>
</tr>
<tr>
<td>BA16</td>
<td>8048 BCD MULT.</td>
<td>5-252</td>
</tr>
<tr>
<td>BB47</td>
<td>8048 DIVISION ROUTINE.</td>
<td>5-288</td>
</tr>
<tr>
<td>BB50</td>
<td>MATH48 - EXTENDED PRECISION ARITHMETIC</td>
<td>5-309</td>
</tr>
<tr>
<td>BD3</td>
<td>DTMHEX - DTMF CODE TO HEX CONVERTER.</td>
<td>5-249</td>
</tr>
<tr>
<td>E25</td>
<td>LANDER</td>
<td>8-70</td>
</tr>
<tr>
<td>REFERENCE NUMBER</td>
<td>PROGRAM</td>
<td>PAGE</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>D45</td>
<td>GRAPH</td>
<td>7-134</td>
</tr>
<tr>
<td>AA13</td>
<td>Intellec MDS Diagnostic Confidence Test</td>
<td>4-360</td>
</tr>
<tr>
<td>BC18</td>
<td>Optimised Ultra-Fast Floating Point Package</td>
<td>5-260</td>
</tr>
<tr>
<td>D29</td>
<td>STAGE 2</td>
<td>7-79</td>
</tr>
<tr>
<td>D49</td>
<td>String Manipulation Package</td>
<td>7-154</td>
</tr>
<tr>
<td>AE14</td>
<td>SYMBOL</td>
<td>4-515</td>
</tr>
<tr>
<td>AB112</td>
<td>SCAN</td>
<td>4-520</td>
</tr>
<tr>
<td>G4</td>
<td>RMX/80 - Based General Keyboard Input Handler</td>
<td>11-10</td>
</tr>
<tr>
<td>AA18</td>
<td>μScope820 Test Instrument, ISBC80/10 Diagnostic</td>
<td>4-545</td>
</tr>
<tr>
<td>F17</td>
<td>LISP Interpreter</td>
<td>9-29</td>
</tr>
<tr>
<td>F16</td>
<td>PILOT-80 ISIS-II Version 2.0</td>
<td>9-28</td>
</tr>
<tr>
<td>D59</td>
<td>COS - A Cassette Operating System for the MDS-800</td>
<td>7-191</td>
</tr>
</tbody>
</table>
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
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. #Fl6

p. 9-28
## HIGHLIGHTS

<table>
<thead>
<tr>
<th>Reference Numbers</th>
<th>Program</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D45</td>
<td>Graph</td>
<td>7-134</td>
</tr>
<tr>
<td>AA13</td>
<td>Intellec MDS Diagnostic Confidence Test</td>
<td>4-360</td>
</tr>
<tr>
<td>BC18</td>
<td>Optimised Ultra-Fast Floating Point Package</td>
<td>5-260</td>
</tr>
<tr>
<td>D29</td>
<td>Stage2</td>
<td>7-79</td>
</tr>
<tr>
<td>D49</td>
<td>String Manipulation Package</td>
<td>7-154</td>
</tr>
</tbody>
</table>
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-axis, 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 through 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 kludging. 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.

Ref. # D45

p. 7-134
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 contiguous or non-contiguous 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
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 occurrence 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:

<table>
<thead>
<tr>
<th></th>
<th>typical (ms.)</th>
<th>maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>0.25</td>
<td>0.53</td>
</tr>
<tr>
<td>Subtract</td>
<td>0.27</td>
<td>0.55</td>
</tr>
<tr>
<td>Multiply</td>
<td>0.75</td>
<td>0.85</td>
</tr>
<tr>
<td>Divide</td>
<td>0.75</td>
<td>0.85</td>
</tr>
<tr>
<td>Square-root</td>
<td>1.50</td>
<td>&lt;2.00</td>
</tr>
</tbody>
</table>

S.N. Cope & S.E. Evans
Dept. Engineering Science
Oxford University
Oxford, England

Ref. # BC18

p. 5-260
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 straightforward 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 alphanumerical 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 preceded 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.
<table>
<thead>
<tr>
<th>REFERENCE NUMBER</th>
<th>PROGRAM</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>D45</td>
<td>GRAPH</td>
<td>7-134</td>
</tr>
<tr>
<td>AA13</td>
<td>INTELLEC MDS DIAGNOSTIC CONFIDENCE TEST</td>
<td>4-360</td>
</tr>
<tr>
<td>BC18</td>
<td>OPTIMISED ULTRA-FAST FLOATING POINT PACKAGE</td>
<td>5-260</td>
</tr>
<tr>
<td>D29</td>
<td>STAGE 2</td>
<td>7-79</td>
</tr>
<tr>
<td>D49</td>
<td>STRING MANIPULATION PACKAGE</td>
<td>7-154</td>
</tr>
<tr>
<td>AE14</td>
<td>SYMBOL</td>
<td>4-515</td>
</tr>
<tr>
<td>AB112</td>
<td>SCAN</td>
<td>4-520</td>
</tr>
<tr>
<td>G4</td>
<td>RMX/80 - BASED GENERAL KEYBOARD INPUT HANDLER</td>
<td>11-10</td>
</tr>
<tr>
<td>AA18</td>
<td>USCOPE 820 TEST INSTRUMENT, ISBC80/10 DIAGNOSTIC.</td>
<td>4-545</td>
</tr>
<tr>
<td>F17</td>
<td>LISP INTERPRETER</td>
<td>9-29</td>
</tr>
<tr>
<td>F16</td>
<td>PILOT-80 ISIS-II VERSION 2.0</td>
<td>9-28</td>
</tr>
<tr>
<td>D59</td>
<td>COS - A CASSETTE OPERATING SYSTEM FOR THE MDS-800</td>
<td>7-191</td>
</tr>
<tr>
<td>BC28</td>
<td>DOUBLE PRECISION FLOATING POINT PACKAGE</td>
<td>5-313</td>
</tr>
</tbody>
</table>
INSITE USER'S LIBRARY

8 BIT PROGRAM INDEX

3-BYTE POSITIVE FRACTIONAL MULTIPLY. 5-213 BB42
3M - DCD1 CASSETTE TAPE DRIVE MONITOR ROUTINES 4-650 AB131
5 LEVEL (BAUDOT) TO 8 LEVEL (ASCII) PAPER
   TAPE CONVERSION. 4-450 AB94
8-BIT MULTIPLY AND DIVIDE. 5-177 BB33
8-BIT RANDOM NUMBER GENERATOR. 5-192 BB37
8-DIGIT LED DISPLAY CONTROLLER - UPI-41. 4-680 AB138
12 X 12 MULTIPLY. 5-228 BB45
16-BIT CRC FOR POLYNOMIAL X16+X12+X5+1 4-44 AB7
16-BIT DIVISION - 16-BIT RESULT. 5-29 BB8
16-BIT DIVISION - 16-BIT RESULT. 5-33 BB9
16-BIT MULTIPLY - 16-BIT RESULT. 5-12 BB2
16-BIT MULTIPLY - 16-BIT RESULT. 5-15 BB3
16-BIT MULTIPLY - 32-BIT RESULT. 5-9 BB1
16-BIT RANDOM NUMBER GENERATOR 5-195 BB38
16-BIT SQUARE ROOT ROUTINE 5-132 BB23
32-BIT BINARY TO BCD CONVERSION.
   LEADING ZERO BLANKING. 5-167 BB29
32-BIT DIVIDE SUBROUTINE 5-175 BB32
32-BIT DIVISION SUBROUTINE - 23 BIT RESULT 5-321 BB51
2708 PROM PROGRAMMER FOR INTELLEC 8/MOD80. 4-364 AB68
4040 CROSS ASSEMBLER FOR INTELLEC 8/MOD80 AND MOD-800. 5-11 F9
8008 CROSS ASSEMBLER FOR 8085-MACRO DEFINITION--M8008.SRC. 6-65 C30
8008 CROSS INVERSE ASSEMBLER FOR HP2100. 6-13 C13
8008 DISASSEMBLER. 5-134 AB20
8008 MACRO ASSEMBLER VERSION 2.0 9-17 F12
8008 MACRO ASSEMBLER VERSION 2.0--ASM88. 9-13 F10
8008 MACRO DEFINITION SET FOR ASSEMBLY ON PDP-11 6-3 C5
8048 BCD MULTIPLY. 5-252 BA16
8048 CRC16 GENERATION SUBROUTINE, IBM. 4-751 AB165
8048-DIV -- DIVISION ROUTINE 5-288 BB47
8048-SEVEN SEGMENT DISPLAY INTERFACE SUBROUTINES--SCAN 4-520 AB112
8048 SIMULATOR - SIM48 6-74 C33
8048 TUNE GENERATOR. 4-524 AB114
8080 CPU EXERCISE ROUTINE. 4-340 RA12
8080 CROSS ASSEMBLER FOR TEKTRONICS 4051. 6-37 C24
8080 DISASSEMBLER. 4-136 AB21
8080 DISASSEMBLER. 4-138 AB22
8080 DOUBLE PRECISION ARC TANGENT. 5-216 BB44

8/79
8080 FLOATING POINT A*B. .......................................... 5-274 BC21
8080 FLOATING POINT EXTENDED MATH PACKAGE .............. 5-183 BC14
8080 FLOATING POINT PACKAGE WITH BCD CONVERSION ROUTINE. 5-87 BC5
8080 Idle analyzer for approximating cpu utilization ............. 4-156 AB26
8080 I/O SYSTEM STATUS DISPLAY ................................ 4-25 AC1
8080 LEAST SQUARES QUADRATIC FITTING ROUTINE .................. 5-89 BC6
8080 MACRO ASSEMBLER 4.1 ....................................... 9-15 F11
8080 RAM MEMORY TEST ........................................... 4-81 AA2
8080 SYMBOL TABLE DUMP ......................................... 4-321 AE3
8085 CROSS ASSEMBLER FOR THE DEC PDP8 AND PDP11 ........... 6-63 C29
8085 CROSS ASSEMBLER FOR THE NOVA 1200 .......................... 6-76 C34
8086 TINY BASIC .................................................. 9-33 F19
8089 -- BREAK.89 ................................................ 4-757 AD6
8278 KEYBOARD/DISPLAY CONTROLLER - UPI-41A ................. 4-711 AB147
8292 (GPIB CONTROLLER) IMPLEMENTATION ON 8741A ............. 4-717 AB150
8295 DOT MATRIX PRINTER CONTROLLER - UPP-41A ............... 4-713 AB148
8741A - 8292 (GPIB CONTROLLER) IMPLEMENTATION .................. 4-717 AB150
9600 INITIALIZE CRT AND UART FOR BAUD .......................... 4-327 AB58

ABSORBANCE CALCULATION ........................................ 7-71 D25
ACQUISITION FROM CONSOLE OF DEIMAL NUMBER WITH ..... 5-311 BC27
CONVERSION TO/FROM FPAL FLOATING POINT .................... 5-311 BC27
A/D CONVERTER ROUTINE ........................................ 4-227 AC14
ADCCP REMAINDER ROUTINE ..................................... 4-353 AB64
ADAPTIVE GAME PROGRAM ....................................... 8-32 E11
ALEGRRAIC COMPARE SUBROUTINE ................................ 5-214 BB43
ALIGN PROGRAM - INTERMEDIATE PASS BETWEEN .................. 6-59 C28
PLM/PASS 1 & 2 ................................................ 4-653 AB132
ALPHA - AN ALPHABETIZED LISTING OF THE DISK DIRECTORY ....... 4-755 AB167
ALPHANUMERIC INPUT FROM NUMERIC KEYBOARD .................. 4-140 D46
ANALOG/DIGITAL POLLING ROUTINE .............................. 4-522 AB113
AP29 "USING THE 8085 SERIAL I/O LINES" ......................... 4-317 AC21
APL GRAPHIC DISPLAY ON A 5 X 7 DOT MATRIX .................. 5-153 BB26
APPROXIMATING ROUTINE ........................................ 5-220 BA13
ARCTAN 2 SUBROUTINE .......................................... 5-207 BB49
ARITHMETIC - MULTIPLE PRECISION FOR PL/M CONVENTIONS .... 5-292 BB48
ARRAY ADDRESSING SUBROUTINE AND CALLING MACRO .............. 4-267 AB45
ASCII DISPLAY ................................................... 5-240 BD2
ASCII TO EBCDIC AND EBCDIC TO ASCII CONVERTERS .............. 5-262 BC19
ASCII STRING TO INTEL FLOATING POINT ......................... 9-13 F10
ASM08 -- 8080 MACRO ASSEMBLER, VERSION 2.0 ..................... 9-31 F18
ASSEMBLER, ON-LINE - ASM ..................................... 4-248 AC13
HANDLER AND ERROR ONLY ASSEMBLER ........................... 4-248 AC13

BANDIT STATIC DISPLAY ......................................... 8-48 E17
BANNER PRINT AND PUNCH ....................................... 4-59 AB11
BASIC CPU STATE VECTOR MAINTENANCE .................................................. 4-273  AB46
BASIC DIGITAL PANEL METER CALL ......................................................... 4-313  AC20
BASIC-II INTERPRETER, LLL ................................................................. 9-7    F7
BASIC INTERPRETER, LLL/CHERNACK .................................................... 9-25   F15
BASIC - TINY FOR 8086 ................................................................. 9-33    F19
BAUD RATE SELECTION FOR MDS 220 OR 230 SYSTEM - RATE .................... 4-737  AB158
BCD TO BIN CONVERSION ROUTINE ..................................................... 5-41   BB11
BCD TO/FROM BINARY CONVERSION ..................................................... 5-45   BB12
BCD INPUT AND DIRECT CONVERSION TO BINARY ROUTINE ....................... 5-78   BB19
BCD MULTIPLICATION ......................................................................... 5-109  BA6
BCD SUM FOR 8086 ........................................................................... 5-54    BA1
BCD UP/DOWN COUNTER .................................................................... 5-212  BA12
BIN TO BCD CONVERSION ROUTINE .................................................... 5-36   BA10
BINARY TO BCD SUBROUTINE ............................................................ 5-72   BB18
BINARY TO GRAY CODE CONVERSION SUBROUTINE - BINGRY ..................... 5-327  BB52
BINARY TO HEX ROUTINE .................................................................. 5-67    BB17
BINARY LOADER FOR MDS. ................................................................. 4-228  AB37
BINARY MULTIPLICATION - 24-BIT ......................................................... 5-24   BB5
BINARY PUNCH TAPE FOR PROM PROGRAMMER - PUNCH ......................... 4-608  AB125
BINARY SEARCH ................................................................................ 7-60    D23
BINARY SEARCH ROUTINE ............................................................... 7-5     D2
BINARY TAPE PROGRAM ................................................................... 4-344  AB62
BINDECBIN - BINARY TO/FROM BCD .................................................... 5-81    BA4
BINGRY - BINARY TO GRAY CODE CONVERSION SUBROUTINE ................ 5-327  BB52
BINLB - 8080 SYSTEM LOADER .......................................................... 4-140  AB23
BIORIM ................................................................................. 8-80    E30
BLACKJACK ..................................................................................... 8-6     E3
BLOCK - LARGE HEX FILE INTO PROM LENGTH BLOCKS CONVERTER .......... 4-662  AB133
BOOT - BOOTSTRAP LOADING AND PROGRAM PATCHING ......................... 4-146  AB24
BREAK.89 --8089 BREAK POINT ROUTINE ......................................... 4-757    AD6
BUFFERED LINE PRINTER DRIVER FOR CENTRONIX 101A - $BLPT. ............... 4-318  AC22
BUS MAPPER FOR PROM ................................................................. 4-669  AB134

CALCULATE A CALENDAR ..................................................................... 7-152  D48
CALENDAR SUBROUTINE ...................................................................... 7-27    D14
CARD READER DRIVER, HOLLERITH TO ASCII CONVERSION .................... 4-533  AB115
CASSETTE (AUDIO) INTERFACE FOR SDK-80 ....................................... 4-706  AB144
CASSETTE OPERATING SYSTEM FOR THE MDS-800 - COS .................... 7-191   D59
CERROR - PL/M-80 COMPILER ERROR DISPLAY PROGRAM ....................... 4-673  AE17
CHARACTER INTERPRETED MEMORY DUMP ........................................... 4-262  AB44
CHECK SUM - CALCULATE TWO VERIFICATION DIGITS FOR .................... 4-626  AB129
DATA STRING .................................................................................. 4-626  AB129
CHECK BOOK BALANCING PROGRAM - FORTRAN-80. ............................. 7-184    D57
CLI - RMX-80 COMMAND LINE INTERPRETER ....................................... 11-1    G1
CLOCK SUBROUTINE .......................................................................... 7-18    D12
COMBINATION I/O DEVICE FOR UPI-41A ............................................ 4-700  AB141
COMPARE ........................................................................................ 4-217  AB34
COMPARE FILES ............................................................................ 4-525  AB116
COMPARE OBJECT CODE TAPE WITH MEMORY .................................... 4-85    AB16
CONTROL DATA OUTPUT ........................................................ 7-14 D5
CONTROLLER FOR HEWLETT-PACKARD 9871A PRINTER ........................ 4-569 AB119
CONTROLLER FOR OLIVETTI 20-COLUMN PRINTER - UPI-41 .............. 4-715 AB149
CONVERSION OF SCIENTIFIC TO EASILY READABLE NOTATION ........... 5-252 BR15
COPY FILE FROM PDP-11 DISKETTE TO ISIS-II DISKETTE ................. 6-78 C35
COS - A CASSIETTE OPERATING SYSTEM FOR THE MDS-800 .................. 7-191 DS9
CRA'S .............................................................................. 8-74 E27
CRC16 IBM BI-SYNC GENERATION SUBROUTINE - 8048 ......................... 4-751 AB165
CRECH - CYCLIC REDUNDANCY CHECK ........................................ 4-53 AB9
CROSS ASSEMBLER FOR HONEYWELL H316/516/716 (REV. B) ............... 6-69 C31
CROSS ASSEMBLER FOR MCS-48 (INTERPRETIVE) ......................... 6-71 C32
CROSS ASSEMBLER FOR NOVA 1200 ........................................... 6-7 C9
CROSS ASSEMBLER FOR THE NOVA 1200 (8085) ............................. 6-76 C34
CROSS ASSEMBLER FOR NOVA 1220, IBM 360/40 AND CDC 3000 ......... 6-9 C10
CROSS ASSEMBLER FOR PDP-11 ............................................. 6-1 C2
CROSS ASSEMBLER FOR PDP-11 ............................................. 6-5 C8
CROSS ASSEMBLER FOR VARIAN DATA MACHINE .......................... 6-23 C18
CROSS REFERENCE FOR PAS80 PASCAL PROGRAMS--XREF80 ............... 4-408 AE8
CRTBZ/GET - INTERFACE WITH HP2640 CRT TERMINAL .................... 4-205 AC16
CUT AND PASTE EDITOR (<PL/9>) ........................................... 4-554 AB117
CYCLIC REDUNDANCY CHARACTER GENERATOR .............................. 4-210 AB33
CYCLIC REDUNDANCY CHECK ................................................ 4-48 AB8
CYCLIC REDUNDANCY CHECK - CRECH ..................................... 4-53 AB9
CYCLIC REDUNDANCY CHECK FOR DATA STRING OF 2*16 BYTES ........ 4-40 AB6

DATA ARRAY MOVE .................................................................. 7-25 D17
DATA GENERAL TO INTELLECG MDS DISKETTE TRANSPORT PACKAGE ...... 4-404 AB83
DATA I/O PROM PROCESSOR .................................................. 4-246 AB41
"DATCON 11" ANALOG TO DIGITAL CONVERSION PROGRAM ............... 4-436 AB91
DDUMP - DISKETTE DUMP ................................................... 4-735 AE19
DECIMAL TO/FROM FPAL FLOATING POINT WITH ........................ 5-311 BC27
ACQUISITION FROM CONSOLE ................................................
DECREMENT H AND L REGISTERS ............................................. 7-9 D3
DELETE COMMENTS ................................................................ 4-297 AB52
DEMONSTRATION PROGRAM WITH TIME FOR RMX/80 ....................... 11-17 G6
DFT - DISCRETE FOURIER TRANSFORM ..................................... 5-315 BC29
DIAGNOSTIC 1003 - MEMORY VALIDITY CHECK ............................ 4-186 AA6
DIGITAL TO ANALOG CONVERSION FOR EIGHT OUTPUTS ............... 5-62 BB16
DISABLE HOLD - SCREEN MODE ............................................ 7-104 D37
DISASSEMBLER .................................................................... 4-289 AB48
DISASSEMBLER FOR SDK-80 (PSEUDO) ..................................... 4-727 AB155
DISK DIRECTORY ALPHABETIZED - ALPHA ................................ 4-653 AB132
DISK DUMP ROUTINE FOR ICOM F DOS-11/MOD80 FLOPPY DOS ........ 4-303 AB55
DISKETTE DUMP - DDUMP .................................................... 4-735 AE19
DISKETTE RECOVERY PROGRAM, RECOVERY 1 ............................ 4-486 AA16
DISPLAY ............................................................................ 4-424 AB88
DKDUMP - ISIS DISK FILE DUMP ........................................... 4-733 AE18
DOUBLE PRECISION FLOATING POINT PACKAGE - FPAL .............. 5-313 BC28
DOUBLE PRECISION INTEGER ARITHMETIC PACKAGE .................... 5-180 BB34
DOUBLE PRECISION MULTIPLY ............................................. 5-128 BB22
DOWN80 ................................................................. 4-749 AB164
DRIVER FOR TEKTRONIX 4010 GRAPHIC SCREEN .................. 4-386 AB78
DTMF TO HEX CODE .................................................... 5-248 BD3
DUMP OF ISIS DISK FILE ............................................... 4-733 AE18

ECCO PAPER TAPE READER ................................................. 4-745 AB162
EDITOR: HEXADECIMAL DISK FILE EDITOR --HXEDIT ............. 4-753 AB166
EID - CALIBRATION FOR MAPPING WITH THE SURFACE OF CRT  
<EIDCAL-IN> ............................................................. 4-622 AB128
ELEMENTARY FUNCTION PACKAGE ...................................... 5-85 BC4
ENABLE HOLD - SCREEN MODE .......................................... 7-101 D36
ENHANCED MDS TEXT EDITOR X111 .................................. 4-482 AB104
ERLIST ............................................................... 4-291 AB49
ERROR DISPLAY FOR PL/M-80 COMPILER – CERROR ............... 4-673 AE17
ERROR MESSAGE PRINT SUBROUTINE (ISIS-II AND USER)  
ERRORP/MESGTP .......................................................... 4-639 AE16
EXAMIN ............................................................... 4-293 AB50
EXEC ................................................................. 4-470 AB100
EXTENDED MATH PACKAGE – FPAL .................................... 5-323 BC32
EXTENDED PRECISION ARITHMETIC – MATH48 ................. 5-309 BB50
EXTRACT SELECTED LINES FROM SOURCE  
OR PRINT FILE – EXTRACT ........................................ 4-600 AB124

FACTORIAL OF A DECIMAL NUMBER ..................................... 5-209 BA11
FAST FLOATING POINT SQUARE ROOT ROUTINE ..................... 5-141 BC12
FAST & SLOW .......................................................... 4-438 AB93
FDUMP ................................................................. 4-480 AB103
FIELD ......................................................... 7-162 D53
FILE GENERATOR – FROM OFFLINE TERMINAL TO ISIS-II FILE  
FILES – PL/M UTILITY PROCEDURES ................................ 4-763 AB170
FIRST-IN, FIRST-OUT BUFFER ROUTINE – FIFO .................... 7-186 D58
FIXED AND FLOATING POINT ARITHMETIC ROUTINES .......... 5-83 BC3
FIXED POINT CHEBYSHEV SINE AND COSINE FOR PL/M USERS  
FLAG PROCESSING ROUTINE ........................................... 5-171 BB31
FLEXIBLE NAME LIST MANAGER ....................................... 4-171 AB29
FLOATING POINT CONSTANT CALCULATOR – FCONST ............. 4-671 AB135
FLOATING POINT CONVERSION ROUTINE ............................... 5-317 BC30
FLOATING POINT DECIMAL AND HEX FORMAT CONVERSION ... 5-301 BC24
FLOATING POINT FORMAT CONVERSION PACKAGE .................... 5-95 BC9
FLOATING POINT <FPAL> TO/FROM DECIMAL  
WITH ACQUISITION FROM CONSOLE .................................. 5-311 BC27
FLOATING POINT INTERPRETER ....................................... 5-222 BC16

FLOATING POINT LOAD AND STORE SUBROUTINES ................. 5-333 BC33
FLOATING POINT MATH PACKAGE ....................................... 5-1 BC1
FLOATING POINT PACKAGE – DOUBLE PRECISION – FPAL ........ 5-313 BC28
FLOATING POINT PACKAGE FOR INTEL 8088 AND
<table>
<thead>
<tr>
<th>Function / Utility</th>
<th>Page</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080 MICROPROCESSORS</td>
<td>5-187</td>
<td>BC15</td>
</tr>
<tr>
<td>FLOATING POINT PROCEDURES</td>
<td>5-91</td>
<td>BC7</td>
</tr>
<tr>
<td>FLOATING POINT SQUARE ROOT</td>
<td>5-127</td>
<td>BC10</td>
</tr>
<tr>
<td>FLOATING POINT UTILITY PROGRAMS FOR USE WITH FPAL.LIB.</td>
<td>5-262</td>
<td>BC22</td>
</tr>
<tr>
<td>FLY READER DRIVER</td>
<td>4-374</td>
<td>AB75</td>
</tr>
<tr>
<td>FORMAT</td>
<td>6-35</td>
<td>C23</td>
</tr>
<tr>
<td>FORMAT INTEL DATA</td>
<td>4-336</td>
<td>AB60</td>
</tr>
<tr>
<td>FORMFD - FORMFEED TO LINEFEED CONVERSION PROGRAM</td>
<td>4-642</td>
<td>AB130</td>
</tr>
<tr>
<td>FOURIER TRANSFORM (DISCRETE) - DFT</td>
<td>5-315</td>
<td>BC29</td>
</tr>
<tr>
<td>FPAL - EXTENDED MATH PACKAGE</td>
<td>5-323</td>
<td>BC32</td>
</tr>
<tr>
<td>FPAL86.LIB - FLOATING POINT LIBRARY</td>
<td>5-319</td>
<td>BC31</td>
</tr>
<tr>
<td>FRONT PAGE IDENTIFIER PRINT PROGRAM</td>
<td>4-678</td>
<td>AB136</td>
</tr>
<tr>
<td>FRUIT MACHINE GAME - TECH-NEL, V1.0</td>
<td>8-88</td>
<td>E34</td>
</tr>
<tr>
<td>GAMBOL</td>
<td>8-11</td>
<td>E5</td>
</tr>
<tr>
<td>GAME OF LIFE</td>
<td>8-20</td>
<td>E8</td>
</tr>
<tr>
<td>GAMMA FUNCTION SUBROUTINE</td>
<td>5-305</td>
<td>BC26</td>
</tr>
<tr>
<td>GENERALIZED STEPPER MOTOR DRIVE PROGRAM</td>
<td>7-33</td>
<td>D21</td>
</tr>
<tr>
<td>GLANCE</td>
<td>4-390</td>
<td>AB80</td>
</tr>
<tr>
<td>GRAPH</td>
<td>7-124</td>
<td>D45</td>
</tr>
<tr>
<td>GRAY TO BINARY CONVERSION</td>
<td>5-50</td>
<td>BB14</td>
</tr>
<tr>
<td>HANDLER FOR TALLY PTP</td>
<td>4-208</td>
<td>AC10</td>
</tr>
<tr>
<td>HANG</td>
<td>8-78</td>
<td>E29</td>
</tr>
<tr>
<td>HAZELTINE 2000 CRT FUNCTION DRIVER</td>
<td>4-426</td>
<td>AB89</td>
</tr>
<tr>
<td>HEWLETT PACKARD CALCULATOR TO MDS800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O CONTROL PROGRAM--HP10</td>
<td>4-563</td>
<td>AB118</td>
</tr>
<tr>
<td>HEX CONVERT - CONVERT INTEL HEX TO PROLOG HEX FILE</td>
<td>6-27</td>
<td>C20</td>
</tr>
<tr>
<td>HEX-FILE FORMAT TO OBJ.-FILE FORMAT WITH SYMBOL</td>
<td>4-611</td>
<td>AB126</td>
</tr>
<tr>
<td>HEX TO ASCII CONVERSION</td>
<td>4-256</td>
<td>BB46</td>
</tr>
<tr>
<td>HEX TO DECIMAL CONVERSION</td>
<td>5-76</td>
<td>BA2</td>
</tr>
<tr>
<td>HEX FILE INTO PROM LENGTH BLOCKS CONVERTER - BLOCK</td>
<td>4-662</td>
<td>AB133</td>
</tr>
<tr>
<td>HEX FORMAT PAPER TAPE DUMP FOR SDK</td>
<td>4-238</td>
<td>AB39</td>
</tr>
<tr>
<td>HEX TAPE LOADER FOR SDK</td>
<td>4-234</td>
<td>AB38</td>
</tr>
<tr>
<td>HIGH SPEED PAPER TAPE READER WITH STEPPER MOTOR CONTROL</td>
<td>4-119</td>
<td>AC7</td>
</tr>
<tr>
<td>HISTOGRAM</td>
<td>5-238</td>
<td>BD1</td>
</tr>
<tr>
<td>HORSEFACE</td>
<td>8-84</td>
<td>E32</td>
</tr>
<tr>
<td>HXEDIT - HEXABCDINAL DISK FILE EDITOR</td>
<td>4-753</td>
<td>AB166</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>I8080 CROSS ASSEMBLER FOR INTEL 8080/8085 MICROPROCESSORS</td>
<td>6-47</td>
<td>C25</td>
</tr>
<tr>
<td>IBM BI-SYNC CRC16 GENERATION SUBROUTINE</td>
<td>4-751</td>
<td>AB165</td>
</tr>
<tr>
<td>IBM SELECTRIC INPUT PROGRAM</td>
<td>4-460</td>
<td>AB96</td>
</tr>
<tr>
<td>IBM SELECTRIC OUTPUT PROGRAM</td>
<td>7-55</td>
<td>D22</td>
</tr>
<tr>
<td>ICE-80 DISASSEMBLER</td>
<td>4-295</td>
<td>AB51</td>
</tr>
<tr>
<td>ICE-80: TRACE.ICE</td>
<td>4-747</td>
<td>AB163</td>
</tr>
<tr>
<td>I-COMMAND - INSERT DATA IN HEX FORM FROM TTY INTO RAM</td>
<td>4-76</td>
<td>AB15</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
<td>Code</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>IDENT1 - FRONT PAGE IDENTIFIER PRINT PROGRAM</td>
<td>4-678</td>
<td>AB136</td>
</tr>
<tr>
<td>INPUT/OUTPUT COMMANDS FOR MDS.</td>
<td>4-223</td>
<td>AB36</td>
</tr>
<tr>
<td>INPUT/OUTPUT DIAGNOSTIC 8080</td>
<td>4-596</td>
<td>AA20</td>
</tr>
<tr>
<td>INSERT TAB CHARACTERS FOR SPACES</td>
<td>4-462</td>
<td>AB98</td>
</tr>
<tr>
<td>INTEL FLOATING POINT NUMBER TO ASCII STRING.</td>
<td>5-268</td>
<td>BC19</td>
</tr>
<tr>
<td>INTEL FORMAT HEX DATA FILE LOAD/READ</td>
<td>4-432</td>
<td>AB90</td>
</tr>
<tr>
<td>INTELLEC 8/MOD80 MONITOR</td>
<td>4-366</td>
<td>AB69</td>
</tr>
<tr>
<td>INTELLEC 8/MOD80 - SILENT 700 INTERFACE</td>
<td>4-128</td>
<td>AC8</td>
</tr>
<tr>
<td>INTELLEC MDS DIAGNOSTIC CONFIDENCE TEST VERSION 1.1</td>
<td>4-360</td>
<td>AA13</td>
</tr>
<tr>
<td>INTELLEC MDS MAILING LIST - FORTRAN-80</td>
<td>7-193</td>
<td>D60</td>
</tr>
<tr>
<td>INTELLEC MDS MAILING LIST MERGE</td>
<td>7-199</td>
<td>D63</td>
</tr>
<tr>
<td>INTELLEC MDS MONITOR VERSION 2.0</td>
<td>4-368</td>
<td>AB70</td>
</tr>
<tr>
<td>INTELLEC 8 TEXT EDITOR</td>
<td>4-362</td>
<td>AB67</td>
</tr>
<tr>
<td>INTERFACING THE MDS AND HP2644</td>
<td>4-388</td>
<td>AC18</td>
</tr>
<tr>
<td>INTERRUPT DRIVEN CLOCK ROUTINE</td>
<td>7-22</td>
<td>D13</td>
</tr>
<tr>
<td>INTERRUPT HANDLER (RE-ENTRANT)</td>
<td>4-132</td>
<td>AD4</td>
</tr>
<tr>
<td>INTERRUPT SERVICE ROUTINE</td>
<td>4-130</td>
<td>AD3</td>
</tr>
<tr>
<td>INVERT DATA IN RAM</td>
<td>7-110</td>
<td>D39</td>
</tr>
<tr>
<td>I/O ROUTINE FOR TI SILENT 700 TERMINAL</td>
<td>4-412</td>
<td>AB85</td>
</tr>
<tr>
<td>I/O SIMULATION MACROS</td>
<td>4-34</td>
<td>AC3</td>
</tr>
<tr>
<td>I/O TEST PROGRAM FOR SBC 80/20 - IOTEST</td>
<td>4-549</td>
<td>AA19</td>
</tr>
<tr>
<td>JOIN - MERGE TWO HEX FILES</td>
<td>4-492</td>
<td>AB108</td>
</tr>
<tr>
<td>JULIAN DATE ROUTINE</td>
<td>7-118</td>
<td>D41</td>
</tr>
<tr>
<td>K, PROGRAM TRAP AND DUMP ROUTINE</td>
<td>4-90</td>
<td>AB17</td>
</tr>
<tr>
<td>KALAH</td>
<td>8-28</td>
<td>E10</td>
</tr>
<tr>
<td>KAPIAR, V1.2 - GENERAL PURPOSE MACROPROCESSOR.</td>
<td>4-743</td>
<td>AB161</td>
</tr>
<tr>
<td>KEYBOARD SCANNER</td>
<td>4-392</td>
<td>AB81</td>
</tr>
<tr>
<td>KEYPAD MONITOR FOR SDK-86</td>
<td>4-704</td>
<td>AB143</td>
</tr>
<tr>
<td>KEYWORD FILE SEARCH</td>
<td>4-739</td>
<td>AB159</td>
</tr>
<tr>
<td>KILL THE ROTATING BIT</td>
<td>8-50</td>
<td>E18</td>
</tr>
<tr>
<td>LANDER</td>
<td>8-70</td>
<td>E25</td>
</tr>
<tr>
<td>LERR - LIST ASSEMBLY ERRORS</td>
<td>4-346</td>
<td>AB63</td>
</tr>
<tr>
<td>LEGIBLE PAPER TAPE</td>
<td>4-57</td>
<td>AB10</td>
</tr>
<tr>
<td>LEWTHWAITE'S GAME</td>
<td>8-45</td>
<td>E15</td>
</tr>
<tr>
<td>LINEAR SYSTEM (GAUSS ELIMINATION)--LISY</td>
<td>5-303</td>
<td>BC25</td>
</tr>
<tr>
<td>LISP INTERPRETER</td>
<td>9-29</td>
<td>F17</td>
</tr>
<tr>
<td>LIST - LIST ERRORS OR SPECIFIED LINES ON CONSOLE</td>
<td>4-319</td>
<td>AE1</td>
</tr>
<tr>
<td>LIST DEVICE PROGRAM</td>
<td>4-190</td>
<td>AA7</td>
</tr>
<tr>
<td>LIST SCR</td>
<td>4-338</td>
<td>AB61</td>
</tr>
<tr>
<td>LIST 1 - HIGH SPEED LIST PROGRAM FOR INTELLEC 8</td>
<td>4-30</td>
<td>AC2</td>
</tr>
<tr>
<td>LIST/PRINT/TYP &quot;LIST SRC&quot; ON DISKETTE</td>
<td>4-307</td>
<td>AB56</td>
</tr>
<tr>
<td>LLL BASIC-II INTERPRETER</td>
<td>9-7</td>
<td>F7</td>
</tr>
<tr>
<td>LLL/CHERNACK BASIC INTERPRETER</td>
<td>9-25</td>
<td>F15</td>
</tr>
</tbody>
</table>

8/79
LOAD ................................................................. 4-380  AB77
LOAD AND STORE SUBROUTINES, FLOATING POINT .......... 5-333  BC33
LOCATION INCLUDED POST ASSEMBLY PROCESSOR ...... 6-67  C30
LOG BASE 2 ............................................................ 5-57  BB15
LRC PRINTER CONTROLLER - UPI-41 ...................... 4-698  AB140
LSORT ................................................................. 7-85  D31

MACRO ASSEMBLER, 8008(VERSION 2.0) --ASM08 ............. 9-13  F10
MACRO ASSEMBLER FOR DG NOVA ............................... 6-49  C26
MACROPROCESSOR, GENERAL PURPOSE - KAPTRON V1.2 ...... 4-743  AB161
MAILING LABEL PROGRAM - PL/M-80 .......................... 7-178  D56
MAILING LIST MERGE PROGRAM ............................... 7-199  D63
MAILING LIST PROGRAM FOR INTELLEC MDS - FORTRAN-80 ... 7-193  D60
MAIN ROUTINE DDUMP (DISKETTE DUMP ROUTINE) ......... 4-406  AB84
MASTERMIND ........................................................... 8-13  E6
MASTERMIND 8080 ....................................................... 8-66  E23
"MASTERMIND 8080" FOR SBC 80/10 ............................. 8-68  E24
MASTERMIND FOR SDK-86 ............................................ 8-86  E33
MATCH .......................... 8-64  E22
MATCH GAME ........................................................... 8-35  E12
MATH48 - EXTENDED PRECISION ARITHMETIC ............... 5-309  BB50
MAZE ................................................................. 8-18  E7
MAZE ................................................................. 8-39  E13
MBCD N1 X N2 BYTES DECIMAL MULTIPLY SUBROUTINE ....... 5-232  BA14
MCS-48 - SQUARE ROOT ROUTINE ............................... 5-331  BB54
MDS BACK TO BACK DATA TRANSFER ......................... 4-474  AB182
MDS SERIES-II - DUMB TERMINAL ......................... 4-761  AB169
MEMORY COMPARE .................................................... 4-199  AB32
MEMORY DIAGNOSTIC PROGRAM ............................... 4-83  AA3
MEMORY DUMP ........................................................... 4-17  AB3
MEMORY TEST FOR THE 8080 ............................... 4-250  AA9
MEMORY TEST PROGRAM ............................................ 4-271  AA11
USCOPE 820 TEST INSTRUMENT ....................................
   ISBC 80/10 DIAGNOSTIC PROGRAM .......................... 4-545  AA18
MINITH - RMX MINIMAL TERMINAL HANDLER ...................... 11-8  G3
MODEL 101 CENTRONICS PRINTER HANDLER ..................... 4-114  AC6
MON 256 -- 256-BYTE PROM MONITOR ......................... 4-260  A843
MONITOR FOR ISBC80/05 OR 80/04 - MON805 .................. 4-590  AB121
MONITOR FOR ISBC80/10 OR 80/10A - MON810 .................. 4-592  AB122
MONITOR FOR ISBC80/20 OR 80/20A - MON820 .................. 4-594  AB123
MONITOR FOR ISBC80/30 - MON830 .......................... 4-679  AB137
MONITOR FOR SDK85 ................................................ 4-731  AB157
MONITOR ROUTINES FOR A 3M-DCD1 CASSETTE TAPE DRIVE .... 4-650  AB131
MORSE CODE GENERATOR ............................................
   7-12  D4
MOUSE ............................................................... 8-90  E35
MSAVE/MLOAD UTILITIES FOR MDS-800 WITH DOS .......... 4-422  AB87
MUL/DIV MULTI-PRECISION PACK FOR 8080 .................... 5-115  BB21
MULTIPLY/DIVIDE SUBROUTINES
24 BIT MULTIPLY FOR 48 BIT QUOTIENT

8/79
48 BIT BY 24 BIT DIVIDE .......................... 5-329 BB53

NATURAL LOGARITHM ................................. 5-146 BB25
N-BYTE BINARY MULTIPLICATION AND LEADING ZERO BLANKING .......................... 5-191 BB20
NIM ..................................................... 8-1 E1
NIM ..................................................... 8-4 E2
NON-ENCODED KEY BOARD SUBROUTINE .................. 4-378 AB76
NOVA CROSS ASSEMBLER - INTEL 8080 ...................... 6-11 C11
NUMBERS .............................................. 8-26 E9

OCTAL CODE CONVERSION FOR PDP-11 ...................... 6-25 C19
OCTAL DEBUGGING PROGRAM (ODT) FOR THE MCS-80 COMPUTER .................. 9-9 F8
OCTAL PROM PROGRAMMING ................................ 4-152 AB25
OCTHEX .................................................. 7-91 D33
OFF-LINE TERMINAL TO ISIS-II FILE GENERATOR .............. 4-763 AB170
OLIVETTI 20-COLUMN PRINTER CONTROLLER - UPI-41 ............ 4-715 AB149
ONLINE, UPLOAD, DOWNLOAD ................................ 4-502 AB111
OPERATING SYSTEM - CASSETTE FOR MDS-800 - COS ................ 7-191 D59
OPTIMISED ULTRA FAST FLOATING POINT PACKAGE ............ 5-260 BC18
OUTIN - PROMPT48 OR PROMPT80 INTERFACE .................. 4-721 AB152
OUTPUT MESSAGE GENERATOR .................................. 7-156 D50

P2708 PROM PROGRAMMING ROUTINE ....................... 4-400 AB82
PAGE BREAK FOR TEKTRONIX 4010 I/O GRAPHICS TERMINAL ........ 4-280 AC15
PAGE LISTING PROGRAM .................................. 4-67 AB13
PAPER TAPE LEADER I.D .................................. 7-81 D30
PAPER TAPE READER (ECCO) ................................ 4-745 AB162
PAPER TAPE REFORMATTER FOR SDK ......................... 4-242 AB40
PASCAL COMPILER, SEQUENTIAL ................................ 9-19 F13
PASS - PARAMETER PASSING ROUTINE ...................... 7-31 D15
PDP-11 BINARY FILE TO INTEL HEX FILE CONVERTER ........... 6-31 C21

PDP-11 DISKETTE FILE COPY TO ISIS-II DISKETTE FILE .......... 6-78 C35
PDP-11 PROGRAM LOAD TO HEX, DUMP, & VERIFY ................ 6-23 C22
PILOT-80 ISIS-II VERSION 2.0 .......................... 9-28 F16
PL/M 80 PASS 3 ........................................ 6-21 C17
PL/M FLOATING POINT INTERFACE .......................... 5-93 BC8
PL/M HISTOGRAM PROCEDURE AND RANDOM NUMBER GENERATOR .... 5-199 BB39
PL/M MULTIPLE PRECISION ARITHMETIC ....................... 5-307 BB49

PL/M UTILITY PROCEDURES - FILES ......................... 7-201 D64
PLOTA .................................................. 7-195 D61
PRINT - LIST FILE ON LP UP TO 255 TIMES .................. 4-488 AB106
PRINT PROGRAM FOR G.E. TERMINET-1200 PRINTER .......... 4-494 AB109
PRINT OUT SOURCE FILE ON FLOPPY DISK ..................... 4-496 AB110
PRINT TEXT FOR SBC 80/10 ................................ 4-580 AB120
QUICKSORT PROCEDURES ........................................ 7-1  D1

RAM CHECK ....................................................... 4-269  AA10
RAM TEST PROGRAM ............................................ 4-5  AA1
RANDOM NUMBER GENERATOR - RINGEN ................... 5-189  BB36
RANDOM#BITS .................................................. 5-204  BB40
RATE - BAUD RATE SELECTION FOR MDS 220 OR 230 SYSTEM 4-737  AB158
REACT .......................................................... 8-56  E20
READ AND INTERRUPT MODIFICATIONS FOR INTELLEC 8/MOD80 4-27  AD2
READ/WRITE ROUTINES FOR INTERCHANGE TAPES .......... 4-164  AC9
READER TEST ................................................... 4-103  AA5
REAL TIME CLOCK SERVICE ROUTINE .................. 7-160  D52
REAL TIME EXECUTIVE ....................................... 4-162  AB27
REAL TIME MONITOR ......................................... 4-334  AB59
RECOVR - TEXT EDITOR RECOVERY PROGRAM ........ 4-418  AB86
RELATIVE JUMP ROUTINE .................................... 7-114  D40
RELOCATABLE FMATH AND XMATH.  
  8085 FLOATING POINT PACKAGE .......................... 5-286  BC23
REMOTE48 - INTERACTIVE CONTROLLER OF PROMPT48 .... 4-723  AB153
RESTART ROUTINE, MODIFIED SDK-80 .................... 4-759  AB168
RETRIEVE - MDS TEXT EDITOR RECOVERY PROGRAM .... 4-617  AB127
RIA80 ........................................................ 9-23  F14
RMSTF - INTEGRATION ROUTINE ......................... 5-162  BB28
RMX/80 - BASED KEYBOARD INPUT HANDLER SUBROUTINE 11-10  G4
RMX/80 COMMAND LINE INTERPRETER - CLI ............ 11-1  G1
RMX/80 DEMONSTRATION PROGRAM WITH TIME .......... 11-17  G6
RMX/80 DRIVER FOR ISBC534 - HND534 ................ 11-15  G5
RMX MINIMAL TERMINAL HANDLER - MINITH. .......... 11-8  G3
RMX MINIMAL TERMINAL OUTPUT - WMIN. .............. 11-3  G2
RTCOPY - COPY FILE FROM PDP-11 DISKETTE (RT-11)  
  TO ISIS-II DISKETTE FILE .............................. 6-78  C35
RUN 0 ........................................................ 7-78  D28
SAMPLE AUTOMATIC TEST EQUIPMENT .................. 4-456  AR15
SAVE/RESTORE CPU STATE ON AN INTERRUPT ....... 4-1  AD1
SBC COMMUNICATOR .................................. 4-458  AR95
SBC 80P REAL TIME CLOCK ......................... 7-158  D51
SBC 80/10 8255 TEST ................................ 4-537  AR17
SBC 80/10 INTERACTIVE MONITOR ................. 4-410  AE9
SBC 80/10 PORT I/O EXERCISER ............... 4-372  RB73
SBC-310 FLOATING POINT SYSTEM FOR USE WITH SBC 80/20 ... 5-284  BC22
SCAN .................................................. 4-384  AE6
SCAN -- 8048 - SEVEN SEGMENT DISPLAY INTERFACE SUBROUTINES ....................................... 4-520  AB112
SCANNER & SELECTIVE FILE LINE PRINTER .......... 4-729  AB156
SDK-80 AUDIO CASSETTE INTERFACE - (TAPE) ...... 4-706  AB144
SDK-80 KEYBOARD MONITOR ......................... 4-302  AB54
SDK-80 PAPER TAPE PUNCH ROUTINE .............. 6-793  D34
SDK-80 PSEUDO DISASSEMBLER ...................... 4-727  AB155
SDK-80 RESTART ROUTINE ............................ 4-759  AB168
SDK-80 TRAP .......................................... 4-466  AE10
SDK-85 - MONITOR FOR THE 8085 SYSTEM DESIGN KIT 4-731  AB157
SDK-86 KEYPAD MONITOR V1.1 ...................... 4-704  AB143
SDK-86 SERIAL MONITOR V1.1 ...................... 4-702  AB142
SDK PROM PROGRAMMER ................................ 4-464  AB99
SEARCH - KEYWORD FILE SEARCH .................. 4-739  AB159
SELECTIVE FILE LINE PRINTER & SCANNER ........ 4-729  AB156
SEND48 DOWNLOAD TO PROMPT48 FOR SERIES II 4-719  AB151
SENSOR MATRIX CONTROLLER - UPI-41A ........... 4-688  AB139
SEQUENTIAL PASCAL COMPILER ...................... 9-19  F13
SERIAL MONITOR V1.1 FOR SDK-86 ................. 4-702  AB142
SERIAL PROM PROGRAMMER ......................... 7-172  D55
SERIES II MDS - DUMB TERMINAL .................. 4-761  AB169
SETS HORIZONTAL TABS ON TERMINET ............. 7-130  D44
SHELLSORTING ROUTINE ........................... 7-30  D18
SIM48 - 8048 SIMULATOR .......................... 6-74  C33
SINX, COSX SUBROUTINE ............................. 5-158  BB27
SLALOM - PL/M ........................................ 8-82  E31
SLOT MACHINE ...................................... 8-62  E21
SMAL: SYMBOLIC MICROCONTROLLER ASSEMBLY LANGUAGE 9-1  F1
SMPY16: 16-BIT 2'S COMPLEMENT SIGNED MULTIPLICATION 5-12  BB4
SNAP DUMP 8080 ...................................... 4-330  AE4
SOFTWARE STACK ROUTINES FOR 8080 ............ 4-176  AD5
SOFTWARE TIMERS - PROGRAMMABLE .............. 4-765  AD7
SORT - GENERAL SORTING PROCEDURE ........... 7-197  D62
SOURCE PAPER TAPE TO MAGNETIC CASSETTE .... 4-72  AB14

SQUARE ROOT - MCS-48 ................................ 5-331  BB54
SQRTF - CALCULATES 8-BIT ROOT OF 16-BIT NUMBER 5-133  BB24
STAGE2 ............................................. 7-79  F20
STATEMENT COUNTER ................................ 4-325  AB57
STEP ................................................ 4-506  AE12
STRING MANIPULATION PACKAGE .................... 7-154  D49

8/79
<table>
<thead>
<tr>
<th>Module</th>
<th>Title</th>
<th>Page</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPI-41</td>
<td>8-DIGIT LED DISPLAY CONTROLLER</td>
<td>4-680</td>
<td>AB138</td>
</tr>
<tr>
<td>UPI-41</td>
<td>LRC PRINTER CONTROLLER</td>
<td>4-698</td>
<td>AB140</td>
</tr>
<tr>
<td>UPI-41</td>
<td>OLIVETTI 20-COLUMN PRINTER CONTROLLER</td>
<td>4-715</td>
<td>AB149</td>
</tr>
<tr>
<td>UPI-41A</td>
<td>8278 KEYBOARD/DISPLAY CONTROLLER</td>
<td>4-711</td>
<td>AB147</td>
</tr>
<tr>
<td>UPI-41A</td>
<td>COMBINATION I/O DEVICE</td>
<td>4-700</td>
<td>AB141</td>
</tr>
<tr>
<td>UPI-41A</td>
<td>SENSOR MATRIX CONTROLLER</td>
<td>4-688</td>
<td>AB139</td>
</tr>
<tr>
<td>UPP-41A</td>
<td>8295 DOT MATRIX PRINTER CONTROLLER</td>
<td>4-713</td>
<td>AB148</td>
</tr>
<tr>
<td>UTILITY MACROS FOR 8080</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Title</th>
<th>Page</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDU DARTS</td>
<td></td>
<td>8-76</td>
<td>E28</td>
</tr>
<tr>
<td>VIDEO DRIVER</td>
<td></td>
<td>4-309</td>
<td>AC19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Title</th>
<th>Page</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIPE</td>
<td>FILE DELETER</td>
<td>4-356</td>
<td>AB65</td>
</tr>
<tr>
<td>WORD GAME, THE</td>
<td></td>
<td>8-8</td>
<td>E4</td>
</tr>
<tr>
<td>WRITEP</td>
<td>OUTPUT PROCEDURES FOR PL/M-80</td>
<td>4-635</td>
<td>AE15</td>
</tr>
<tr>
<td>WRMIN</td>
<td>RMX MINIMAL TERMINAL OUTPUT</td>
<td>11-3</td>
<td>G2</td>
</tr>
</tbody>
</table>