Addressing Modes

The MPU6800 microprocessor has five addressing modes available to the programmer.

1. Immediate: In this mode of addressing, the operand is contained in the next memory location. For example, to execute a "load accumulator with the hex number 55" instruction, it would look like this in memory.

Memory Location	Binary Contents	Hex Contents
100	10000110	86 (LDA A IMM)
101	01010101	55 (DATA)

86 (in hex) is the LDA A immed. instruction. 55 (in hex) is the data. The result after the above is the hex number 55 has been loaded into the A accumulator.

2. Direct: In this mode of addressing the address of the operand is contained in the next memory location. This enables one to directly address the first 256 bytes of memory (0-255=256 Bytes). As an example, to load accumulator A with the contents of address 67 (in hex), consecutive memory locations would look like this.

Memory Location	Binary Contents	Hex Contents
100	10010110	96 (LDA A DIR)
101	01100111	67(Address that contains data)

96 (in hex) is the LDA A Direct instruction.

67 (in hex) is the address where the data is to be fetched from. So, whatever is in location 67 would be loaded into accumulator A.

3. Extended: This mode of addressing is used to address memory locations above 255. In this mode of addressing, the next memory location contains the higher order 8 bits of the address, and the 2nd memory location contains the lower order 8 bits of the address. For example, to load the A accumulator with the contents of memory location hex 4057, the consecutive memory locations would look like this.

Memory Location	Binary Contents	Hex Contents
100	10110110	B6(LDA A EXT)
101	01000000	40(ADDR HIGH)
102	01010111	57(ADDR LOW)

B6(in hex) is the LDA EXT instruction. 40 (in hex) is the most significant half of the address where the data is stored and 57 (in hex) is the least significant half of the address where the data is stored. After the above execution, whatever is in location 4057 will be loaded into accumulator A.

4. Indexed: In-this mode of addressing, the address contained in the next memory location is added to the contents of the index registers lower 8 bits to form a new "effective address". If there was a carry, it is added to the upper 8 bits of the index register. The new "effective address" is the location in memory which contains the operand. The "effective address" is held in a temporary address register such that the contents of the index register are not destroyed. As an example, if the index register contains hex 14, and a load accumulator A from hex location 21 indexed by the contents of the index register is executed,

the address of 21 (located in the next memory location) is added to the contents of the index register (14) to form a new "effective address" of hex 35.

Memory Location	Binary Contents	Hex Contents
100	10100110	A6(LDA A Indexed)
101	00100001	21

A6(in hex) is the LDA INDEXED instruction.
21 (in hex) contains part of the address of the instruction. To the address of 21 must be added the contents of the index register to form a new "effective address" hex of 35 (21 + 14). After the above execution, the contents of memory location hex 35 will be loaded into accumulator A.

5. Relative: In this mode of addressing, program control is transferred to someplace other than the next sequential memory location. Transfer in this mode, is limited to 125 memory locations back from the present location or 129 locations ahead of the present location. Since this is a 2 byte instruction in that it takes two memory locations, transfer is always referenced from the next instruction which the MPU would execute if it did not transfer control(or relative to the present count of the program counter). All transfers back from the present location are given in 2's complement (represented in hex) from the (present location + 0002).

All transfers forward are given in the actual count forward from (the present memory location + 0002) to the memory location where program control is transferred. The actual

count forward is given in straight binary (represented in hex).

TRANSFER FORWARD FROM PRESENT LOCATION

Assume it is desired to branch from the present location at 0100 + 0002 (in hex) to location 0147 (in hex). First, it should be verified that the branch is not beyond the allowable range of 199 locations from the present location. 45 (in hex) = 5X16° + 4x16'=5+64- 69 (decimal) Therefore 45 hex is within our allowable range. At memory location 0100, a BRA instruction is stored. Memory location 0101 contains the count of memory locations which will be branched over starting from 0102.

Final Destination = 0147

Present Location + 0002 = 0102

Number of Locations to Branch over = 45

Memory Location	Binary Contents	Hex Contents
100	00100000	20 (BRA)
101	01000101	45 (No. of locations to branch over)

20 (in hex) is the BRA (Branch Always) instruction.

45 (in hex) is the number of locations which will be branched over starting with 0102 Therefore, the next instruction the MPU will execute will be located at 102 + 45 or hex location 0147.

TRANSFER SACK FROM PRESENT LOCATION

Assume it is desired to branch from the present location of 0100 back to memory location 0090. This is accomplished in a similar manner as the forward branch, except the number of locations is given in 2's complement (represented in hex) from the present location + 0002. The 2's complement form places a 1 in bit 8 which, in effect tells the processor to branch back rather than forward.

Present Location + 0002 = 0102

Final Destination = 0090

Number of Locations to Branch back over = 72

Number of Locations to Branch back over = 01110010 (72 hex)

1's complement = 10001101

2's complement = 10001110 (8E)

Memory Location	Binary Contents	Hex Contents
100	00100000	20 (BRA)
101	10001110	8E (No. of locations to branch back over)

20 (in hex) is the BRA (Branch Always) instruction. 8E is the number of locations (in 2's complement) which will be branched back over starting from 0102. (present location + 0002 which is the count in the program counter). Therefore, the next instruction the MPU will execute will be located at memory location 0090 (hex)

Sample Program

Problem: Write a program, in machine language and in M6800 source language, to add the decimal numbers 25, 35, 50, and 17. Store the answer at RAM location 0A. Assemble the source program and compare the assembled program with the machine language program.

Solution: $35_{10} = 43_8 = 100011_2 = 23_{16}$

$50_{10} = 62_8 = 110010_2 = 32_{16}$

$17_{10} = 21_8 = 010001_2 = 11_{16}$

$25_{10} = 31_8 = 011001_2 = 19_{16}$

Memory(in HEX) Location	Machine Language		Comment
000B	10000110	(86)	LDA A IMM
000C	00011001	(19)	DATA TO BE PUT IN A
000D	11000110	(C6	LDA B IMM.
000E	00100011	(23)	DATA TO BE PUT IN B
000F	00011011	(1B)	ADD THE A & B REGISTER
0010	11000110	(C6)	LDA B IMM
0011	00110010	(32)	DATA TO BE PUT IN B
0012	00011011	(1B)	ADD THE A&B REGISTER
0013	11000110	(C6)	LDA B IMM
0014	00010001	(11)	DATA TO BE PUT IN B
0015	00011011	(1B)	ADD THE A&B REGISTER
0016	10010111	(97)	STORES A IN LOCATION
0017	00001010	(0A)	0A

```
LIST
RON4       09:30  PHENIX 05/29/74

100  NAM SAMP
105  ORG 10
110 TEMP RMB 1
130  LDA A #25
140  LDA B #35
150  ABA
160  LDA B #$32
170  ABA
180  LDA B #@21
190  ABA
200  STA A TEMP
205  END

READY
RUN:MPCASM,M001


MPCASM       09:31        05/29/74


 MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPCASM
           COPYRIGHT 1974 BY MOTOROLA INC

      MOTOROLA MPU CROSS ASSEMBLER, RELEASE 1.0

   ENTER SI FILENAME
 ? RON4

----

 PAGE  1  SAMP   05/29/74 09:32.00


 00100                          NAM     SAMP
 00105 000A                     ORG     10
 00110 000A 0001     TEMP       RMB     1
 00130 000B 86 19               LDA A   #25
 00140 000D C6 23               LDA B   #35
 00150 000F 1B                  ABA
 00160 0010 C6 32               LDA B   #$32
 00170 0012 1B                  ABA
 00180 0013 C6 11               LDA B   #@21
 00190 0015 1B                  ABA
 00200 0016 97 0A               STA A   TEMP
 00205                          END


 SYMBOL TABLE


  TEMP   000A
   ENTER SI FILENAME
 ? EOF

ERRLN=

RUNNING TIME:    9.1 SECS   I/O TIME :    6.2 SECS
```

SAME PROGRAM WRITTEN IN MPU MNEMONIC CODING

\# INDICATES IMMEDIATE

$ INDICATES HEX NUMBER

@ INDICATES OCTAL NUMBER

MPU MNEMONIC CODED PROGRAM ASSEMBLED BY TIME SHARING ASSEMBLER

Sample Program: Loading and Storing Data

Write a program for the following sequence.

1. Begin with data 7F and load it into the A accumulator then store the data in memory location 50.

2. From location 50, load the data into the B accumulator then store it extended in memory location 0113.

3. Reload data into the A accumulator from the extended memory location and store the data in location 6A then Jump back to the beginning.

Assume this program will be used in a microcomputer system with Hex Ram addresses 000 through 200 (512 bytes) and ROM addresses 800 through FFF (2048 bytes). All numbers are in Hex relation.

Source Program

```
EDU1          12:09EST     02/06/75

100  NAM LTR1
101  OPT MEM
102  ORG $6A
103 TEMP RMB 1
105  ORG $0800
110 START LDA A #$7F     START OF PROGRAM
120  STA A $50
130  LDA B $50          ADDRESS OF DATA
140  STA B $0113
150  LDA A $0113
180  STA A TEMP
190  JMP START
200  MON
```

Assembled Program

```
00100                          NAM     LTR1
00101                          OPT     MEM
00102 006A                     ORG     $6A
00103 006A 0001     TEMP       RMB     1
00105 0800                     ORG     $0800
00110 0800 86 7F    START      LDA A   #$7F      START OF PROGRAM
00120 0802 97 50               STA A   $50
00130 0804 D6 50               LDA B   $50       ADDRESS OF DATA
00140 0806 F7 0113             STA B   $0113
00150 0809 B6 0113             LDA A   $0113
00180 080C 97 6A               STA A   TEMP
00190 080E 7E 0800             JMP     START
00200                          MON
```

Sample Program: Subtracting absolute value of two numbers

Problem: Calculate a quantity Z which will be the absolute value of Y subtracted from the absolute value of W. If the result is less than or equal to zero, set Z equal to zero.

$Z = |W| - |Y|$ if $|W| > |Y|$

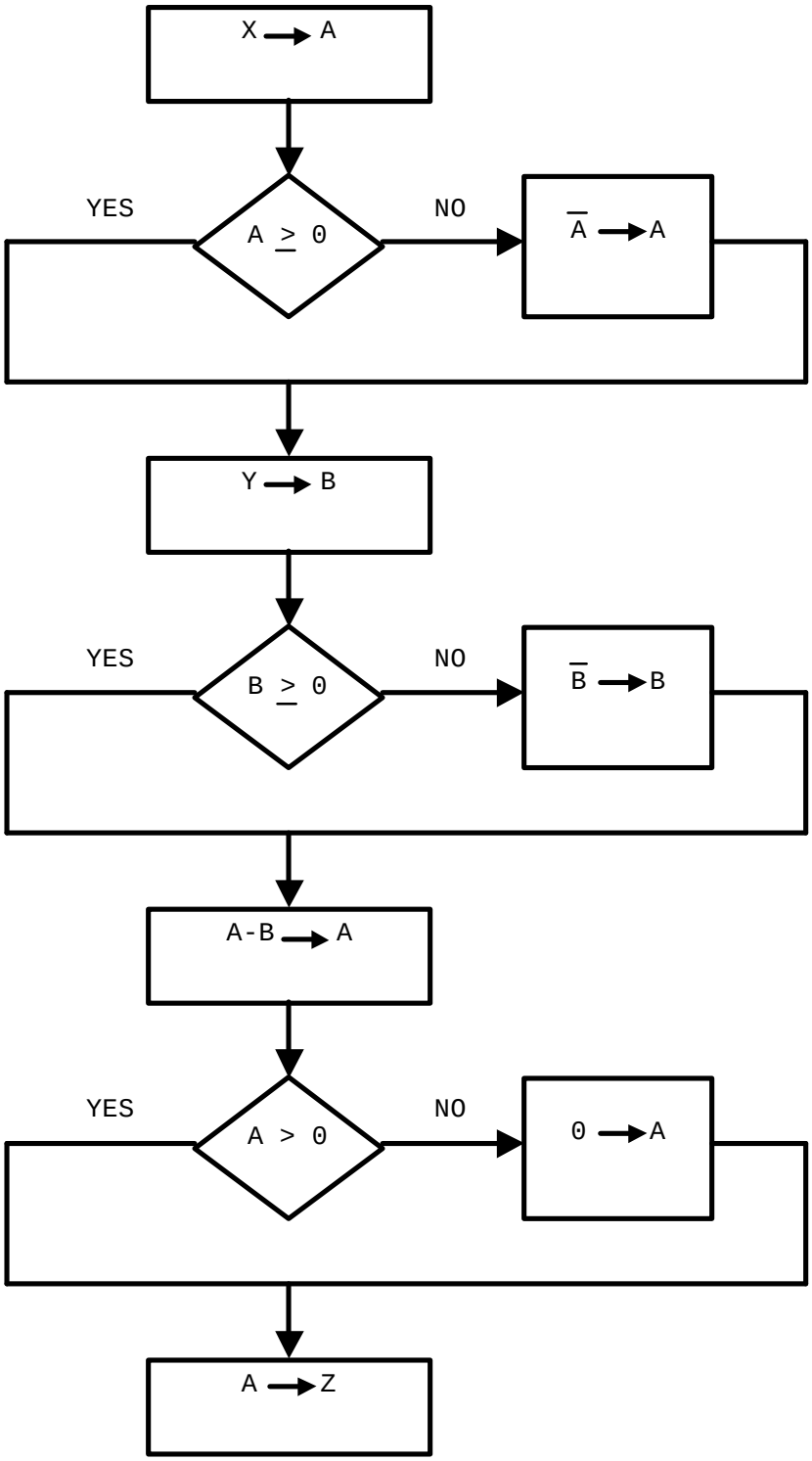
$Z = 0$ if $|W| \leq |Y|$

Source Program

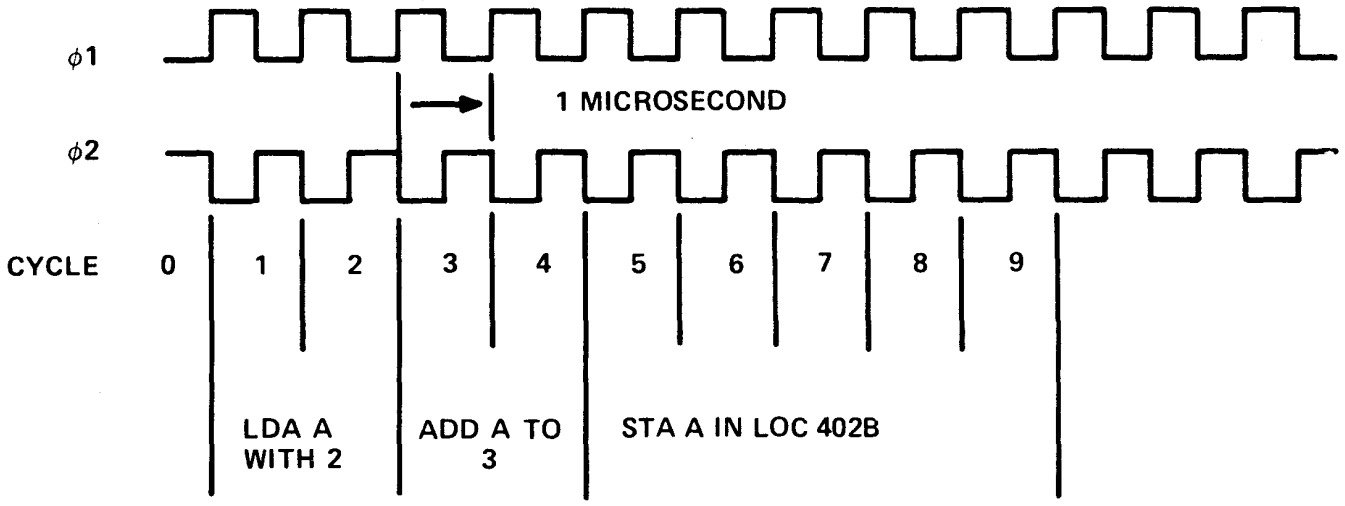
```
100  NAM ABS
110  OPT MEM
120  ORG 0
130 W RMB 1
140 Y RMB 1
150 Z RMB 1
160  ORG $0500
170  LDA A W
180  BGE Z1     IS W POSITIVE
190  NEG A      W WAS NEG. MAKE POS.
200 Z1 LDA B Y
210  BGE Z2     IS Y POSITIVE
220  NEG B      Y WAS NEG. MAKE POS.
230 Z2 SBA      SUBTRACT Y FROM W
240  BGT Z3     IS Z POSITIVE
250  CLR A      RESULT WAS ZERO OR NEG.
260 Z3 STA A Z  ANSWER
270  MON
```

Assembled Program

```
00100                          NAM    ABS
00110                          OPT    MEM
00120 0000                     ORG    0
00130 0000 0001     W          RMB    1
00140 0001 0001     Y          RMB    1
00150 0002 0001     Z          RMB    1
00160 0500                     ORG    $0500
00170 0500 96 00               LDA A  W
00180 0502 2C 01               BGE    Z1        IS W POSITIVE
00190 0504 40                  NEG A            W WAS NEG. MAKE POS.
00200 0505 D6 01    Z1         LDA B  Y
00210 0507 2C 01               BGE    Z2        IS Y POSITIVE
00220 0509 50                  NEG B            Y WAS NEG, MAKE POS.
00230 050A 10       Z2         SBA              SUBTRACT Y FROM W
00240 050B 2E 01               BGT    Z3        IS Z POSITIVE
00250 050D 4F                  CLR A            RESULT WAS ZERO OR NEG.
00260 050E 97 02    Z3         STA A  Z         ANSWER
00270                          MON
```


X → A
YES
A ≥ 0
NO
$\overline{A}$ → A
Y → B
YES
B ≥ 0
NO
$\overline{B}$ → B
A-B → A
YES
A > 0
NO
0 → A
A → Z

CYCLE BY CYCLE DESCRIPTION OF SAMPLE PROGRAM



ROM ADDRESS	ROM CONTENT	INSTRUCTION
0018	86	LDA A #2
0019	02	
001A	8B	ADD A #3
001B	03	
001C	F6	STA A $402B
001D	40	
001E	2B	

INDICATES IMMEDIATE MODE OF ADDRESSING

$ INDICATES A HEX NUMBER

NOTE: ADDRESS 402B MUST BE A RAM, PIA, OR ACIA.

DESCRIPTION OF PROGRAM: The A register is loaded with the number 2. Then the number 3 is added to the 2 in the A register with the result of 5 left in the A register. The 5 in the A register is then stored in location 402B.

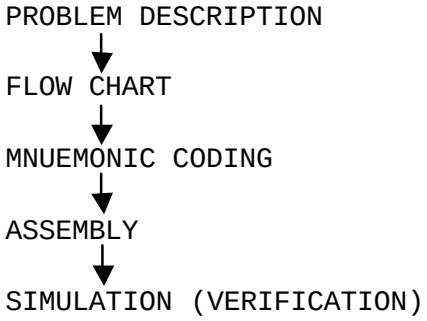
Cycle By Cycle Description of Sample Program

Cycle	Description
0	The program counter is assumed to be set at 0018.
1	The program counter is gated onto the Address Bus (A0-A15) and the read/write (R/W) line is put in a high state corresponding to a read condition. This results in ROM address 0018 be accessed and the contents of this address (86) being loaded into the instruction register (IR). The program counter is then incremented and becomes 0019.
2	The byte "86" in the IR is decoded and interpreted to be a load A immediate (LDA A IMM) instruction. Simultaneously, the program counter is gated onto the Address Bus and the R/W line is set high corresponding to a read condition. This accesses ROM address 0019 with the contents of this address (02) being put on the Data Bus (D0-D7). Since the instruction was decoded to be a LDA A immediate, the "02" is loaded into the A register. The program counter is then incremented and becomes 001A.
3	The sequence in (1) is repeated except ROM address 001A is accessed resulting in 8B being loaded into the instruction register. The program counter is incremented to 001B.
4.	The sequences in (2) is repeated except the instruction is decoded to be an ADD A immediate. Thus, the data "03" is added to the A register giving a result in the A register of "05". The program counter is incremented to 001C.
5	The sequences in (1) is repeated which results in F6 being loaded into the instruction register. The program counter is incremented to 001D.
6	The instruction register is decoded and determined to be a STA A extended. This causes the MPU to interpret the next two sequential locations in memory (00lD & 001E) as a 16 bit address with 001D the most significant and 001E the L.S. half of the address. Simultaneously, the number in ROM address 001D is read by the MPU and saved the program counter is incremented to 001E.
7	The contents of ROM address 001E (2B) is read by the MPU and saved. The MPU now has a full 16 bit address saved of 402B.
8	The extended address of 402E is gated onto the address bus register.
9	Address 402B is accessed and the R/W line is put in a low state, corresponding to a write. The data in the A register is then gated onto the data bus and stored in location 402B.

Sample Program: Multiply Routine #1

This handout documents the procedures followed to solve a typical problem using the M6800 software and software aids. The problem was the multiplication of two unsigned eight bits numbers.

The objective was to show the general method involved in using the Motorola Cross Assembler and Simulator to assist the programmer. The chart below illustrates the steps followed:



THIS PROGRAM IS FOR ILLUSTRATION ONLY.
IT IS NOT THE MOST EFFICIENT MULTIPLY
ROUTINE. IT IS SHOWN ONLY AS AN EXAMPLE
OF PROGRAMMING TECHNIQUES.

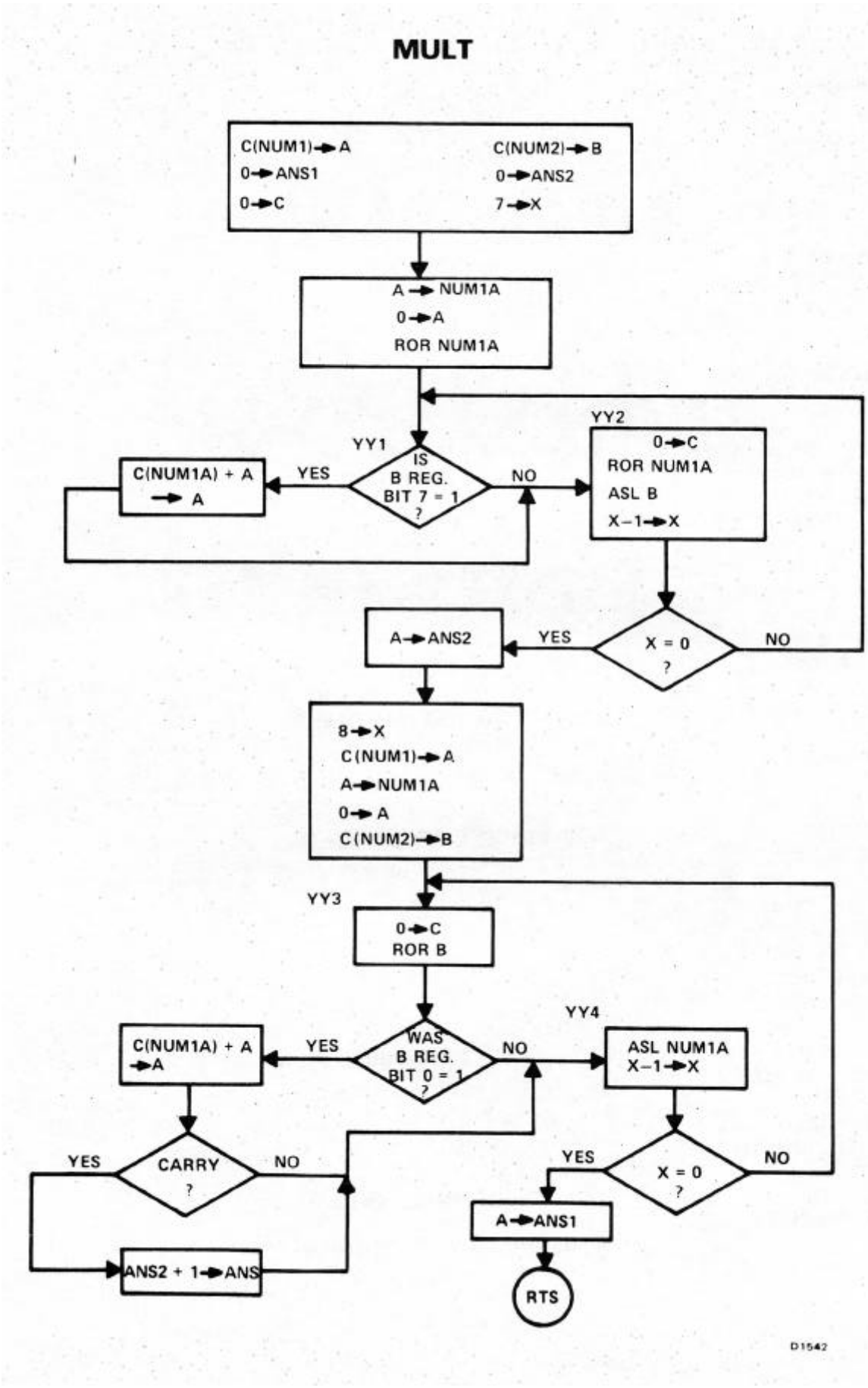
Subroutine "MULT" will multiply two unsigned 8 bit numbers (NUM1 & NUM2) and store the 16 bit result in locations ANS1 (Least significant 8 bits) and ANS2 (Most significant 8 bits). The algorithm used can be best explained by an example:

```
                    10000001          (Multiplicand) NUM1
                    11111111          (Multiplier) NUM2
                  ┌10000001
                  │1000001
                1 │000001
               10 │00001
              100 │0001
             1000 │001
            10000 │01
           100000 │1
          1000000 │01111111
           ANS2    ANS1
```

1 ANS2 is generated by shifting the multiplicand one bit to the right and then examining the most significant bit of the multiplier--if it is a "1", the multiplicand is added to ANS2. The multiplier is then shifted one bit to the left and the procedure (1) is repeated. This is done seven times to generate the seven terms of ANS2. No carry bit is possible from these additions.

2 ANSI is generated by examining the least significant bit of the multiplier-if it is a "1" the multiplicand is added to ANSI. The multiplicand is then shifted left one bit and the multiplier is shifted right one bit. The procedure (2) is repeated eight times to generate the eight terms of ANSI. If a carry occurs after any of these additions, one is added to ANS2.

MULT


C(NUM1)→A
0→ANS1
0→C
C(NUM2)→B
0→ANS2
7→X
A→NUM1A
0→A
ROR NUM1A
YY1
IS B REG. BIT 7 = 1 ?
YES
C(NUM1A) + A →A
NO
YY2
0→C
ROR NUM1A
ASL B
X−1→X
X = 0 ?
YES
A→ANS2
NO
8→X
C(NUM1)→A
A→NUM1A
0→A
C(NUM2)→B
YY3
0→C
ROR B
WAS B REG. BIT 0 = 1 ?
YES
C(NUM1A) + A →A
NO
YY4
ASL NUM1A
X−1→X
CARRY ?
YES
NO
ANS2 + 1→ANS
X = 0 ?
YES
NO
A→ANS1
RTS
D1542

SOURCE LISTING

```
LMULT     09:06   PHENIX 06/12/74

50  NAM AMULT
60  ORG 0
70  OPT MEM
95 *
96 *
100 ANS1 RMB 1
110 ANS2 RMB 1
120 NUM1 RMB 1
130 NUM2 RMB 1
140 NUM1A RMB 1
145 *
146 *
190  ORG 1000
195 *
196 *
200 MULT LDA A NUM1          ;NUM1=MULTIPLIER
210  LDA B NUM2              ;NUM2=MULTIPLICAND
220  CLR ANS1
230  CLR ANS2
240  CLC
250  LDX #7                  ;LOOP COUNT
260  STA A NUM1A
270  CLR A
280  ROR NUM1A
290 YY1 TST B                ;SET COND. CODES ACC.TO B
300  BPL YY2                 ;CHECK FOR A 1 IN BIT 7
310  ADD A NUM1A             ;OVERFLOW NOT POSSIBLE
320 YY2 CLC
330  ROR NUM1A
340  ASL B
350  DEX
360  BNE YY1                 ;CONTINUE UNTIL X=0
370  STA A NUM1A
380  LDX #8                  ;LOOP COUNT
390  LDA A NUM1
400  STA A NUM1A
410  CLR A
420  LDA B NUM2
430 YY3 CLC
440  ROR B
450  BCC YY4                 ;IF CARRY,INCREMENT ANS2
453  ADD A NUM1A
456  BCC YY4
460  INC ANS2
470 YY4 ASL NUM1A
480  DEX
490  BNE YY3                 ;CONTINUE UNTIL X=0
500  STA A ANS1
510  RTS                     ;FINISHED, EXIT TO MAIN
600  MON

READY
```

```
PAGE   1  AMULT   06/12/74   09:08.00                          ASSEMBLY LISTING

00050                          NAM    AMULT
00060 0000                     ORG    0
00070                          OPT    MEM
00095                  *
00096                  *
00100 0000 0001        ANS1    RMB    1
00110 0001 0001        ANS2    RMB    1
00120 0002 0001        NUM1    RMB    1
00130 0003 0001        NUM2    RMB    1
00140 0004 0001        NUM1A   RMB    1
00145                  *
00146                  *
00190 03E8                     ORG    1000
00195                  *
00196                  *
00200 03E8 96 02       MULT    LDA A  NUM1       ;NUM1=MULTIPLIER
00210 03EA D6 03               LDA B  NUM2       ;NUM2=MULTIPLICAND
00220 03EC 7F 0000             CLR    ANS1
00230 03EF 7F 0001             CLR    ANS2
00240 03F2 0C                  CLC
00250 03F3 CE 0007             LDX    #7         ;LOOP COUNT
00260 03F6 97 04               STA A  NUM1A
00270 03F8 4F                  CLR A
00280 03F9 76 0004             ROR    NUM1A
00290 03FC 5D          YY1     TST B             ;SET COND. CODES ACC.TO B
00300 03FD 2A 02               BPL    YY2        ;CHECK FOR A 1 IN BIT
00310 03FF 9B 04               ADD A  NUM1A      ;OVERFLOW NOT POSSIBLE
00320 0401 0C          YY2     CLC
00330 0402 76 0004             ROR    NUM1A
00340 0405 58                  ASL B
00350 0406 09                  DEX
00360 0407 26 F3               BNE    YY1        ;CONTINUE UNTIL X=0
00370 0409 97 04               STA A  NUM1A
00380 040B CE 0008             LDX    #8         ;LOOP COUNT
00390 040E 96 02               LDA A  NUM1
00400 0410 97 04               STA A  NUM1A
00410 0412 4F                  CLR A
00420 0413 D6 03               LDA B  NUM2
00430 0415 0C          YY3     CLC
00440 0416 56                  ROR B
00450 0417 24 07               BCC    YY4        ;IF CARRY,INCREMENT ANS2
00453 0419 9B 04               ADD A  NUM1A
00456 041B 24 03               BCC    YY4
00460 041D 7C 0001             INC    ANS2
00470 0420 78 0004     YY4     ASL    NUM1A
00480 0423 09                  DEX                ;
00490 0424 26 EF               BNE    YY3        ;CONTINUE UNTIL X=0
00500 0426 97 00               STA A  ANS1
00510 0428 39                  RTS                ;FINISHED, EXIT TO MAIN
00600                          MON

   SYMBOL TABLE:
 ANS1   0000   ANS2   0001   MULT   03E8   NUM1   0002   NUM1A  0004
 NUM2   0003   YY1    03FC   YY2    0401   YY3    0415   YY4    0420
STOP

RUNNING TIME: 69.4 SECS    I/O TIME: 26.1 SECS
```

MULTIPLY SUBROUTINE #2

This subroutine multiplies two eight bit unsigned binary numbers. The product of the two eight bit numbers is formed by shifting the multiplier one bit to the right and checking for a one or zero. If a one is present, the multiplicand is added to the product (answer).

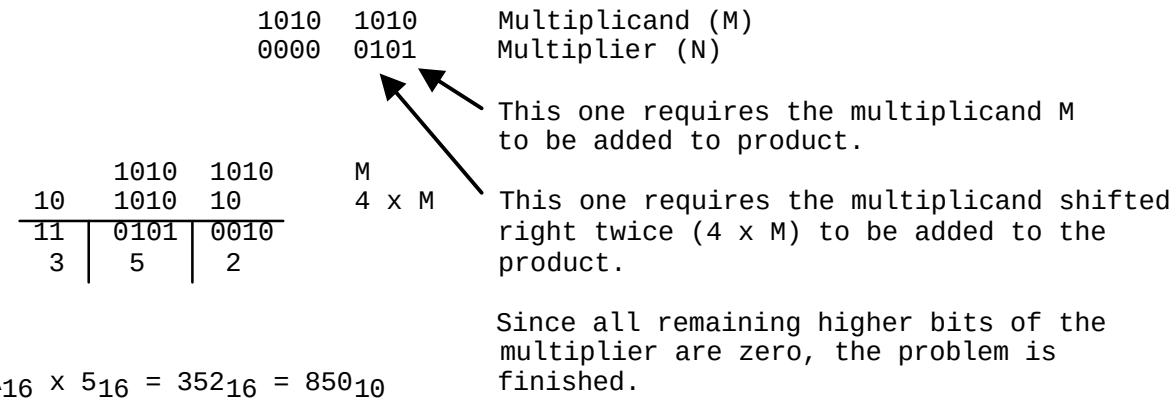
The multiplicand is then shifted one bit to the left. This has the effect of multiplying the multiplicand by two. The multiplier is again shifted one bit to the right and the shifted bit checked for a one or zero. If it is a one, the shifted multiplicand is added to the product. The process is repeated until the multiplier has no more ones remaining. When no more ones remain in the multiplier, the problem is finished and the product is the final product.

Example

Multiply $170_{10} \times 5_{10} = 850_{10}$

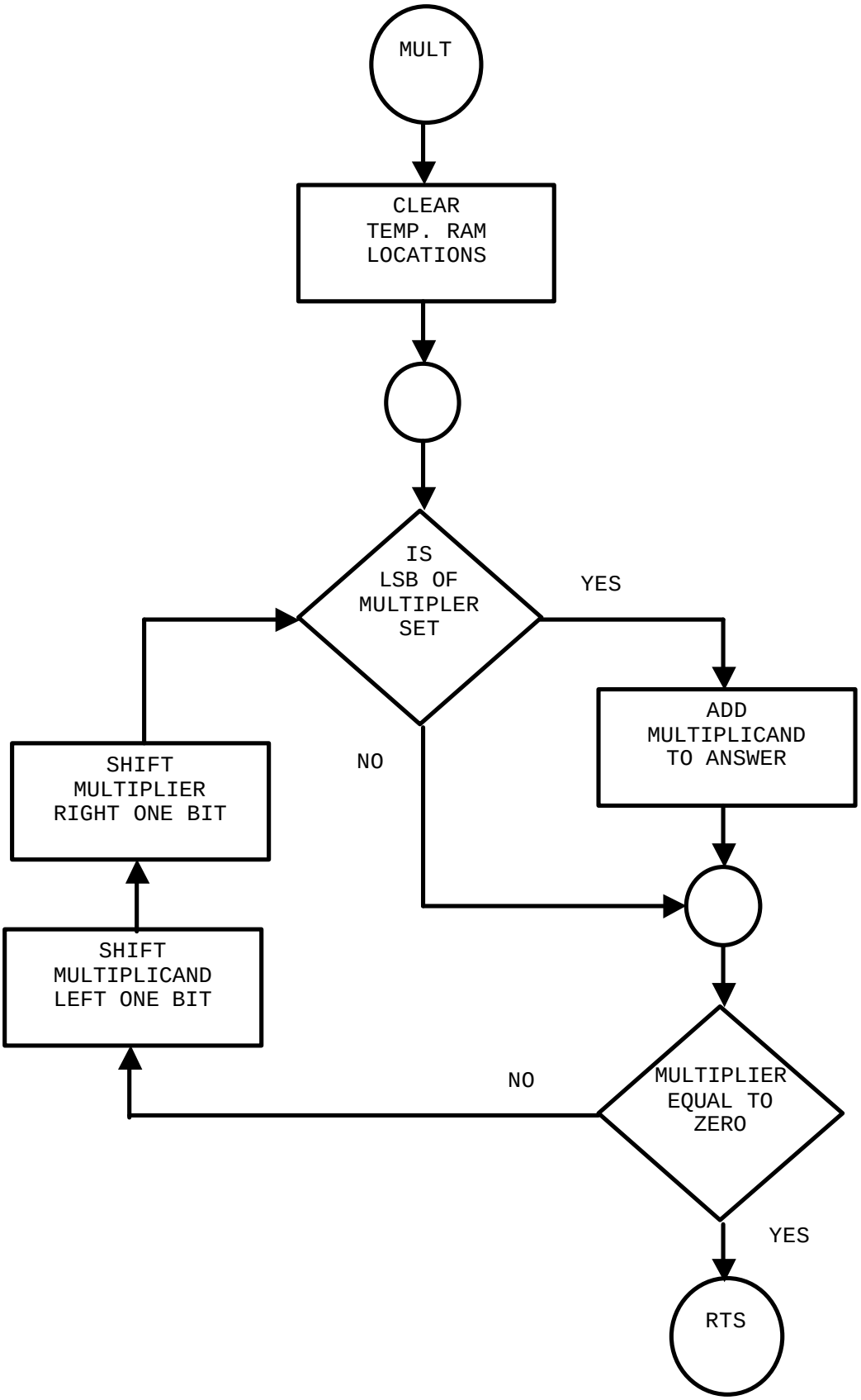
$170_{10} = AA_{16}$

$5 = 05_{16}$



$AA_{16} \times 5_{16} = 352_{16} = 850_{10}$

FLOW CHART OF MULTIPLY ROUTINE #2



CMULT 12/05/74

```
100  NAM CMULT
110  OPT MEM
120*************************************************************
130*REV002 12-5-74 BAINTER
140*
150* THIS SUBROUTINE MULTIPLIIES TWO 8 BIT BYTES.
160* THE MULTIPICAND IS STORED IN BITE NB1.
170* THE MULTIPLIER IS STORED IN BITE NB2.
180* THE RESULT IS STORED IN BYTE ANS2 AND ANS1.
190* ANS2. IS THE UPPER BITE OF THE RESULT.
200* ANS1. IS THE LOWER BITE OF THE RESULT.
210*************************************************************
220*
230  ORG 0
240*
250 NB1A RMB 1 *SHIFT MULTIPLICAND STORE.
260 NB1  RMB 1 *MULTIPLICAND
270 NB2  RMB 1 *MULTIPLIER
280 ANS2 RMB 1 *UPPER BYTE OF RESULT
290 ANS1 RMB 1 *LOWER BYTE OF RESULT
300*
410  ORG $10
320*
330 MULT CLR A
340  STA A ANS2
350  STA A ANS1
360  STA A NB1A
370  LDA A NB2      *NB2=MULTIPLIER
380  BRA LOOP1
390 LOOP2 ASL NB1   *SHIFT MULTIPLICAND LEFT.
400  ROL NB1A       *NB1A=UPPER BYTE OF MULTIPLICAND
410 LOOP1 LSR A     *SHIFT MULTIPLIER RIGHT
420  BCC NOADD      *SHIFT AND DON'T ADD
430  LDA B NB1      *ADD SHIFTED MULTIPLICAND-
440  ADD B ANS1     *TO ANS1 AND ANS2
450  STA B ANS1
460  LDA B NB1A
470  ADC B ANS2     *ANS2=UPPER BYTE OF RESULT
480  STA B ANS2
490  TST A
500 NOADD BNE LOOP2 *START SHIFTING AGAIN.
510 END RTS         *FINISHED!!!
520 MON
```

```
00100                           NAM    CMULT
00110                           OPT    MEM
00120                   ************************************************************
00130                   *REV002 12-5-74 BAINTER
00140                   *
00150                   * THIS SUBROUTINE MULTIPLIIES TWO 8 BIT BYTES.
00160                   * THE MULTIPICAND IS STORED IN BITE NB1.
00170                   * THE MULTIPLIER IS STORED IN BITE NB2.
00180                   * THE RESULT IS STORED IN BYTE ANS2 AND ANS1.
00190                   * ANS2. IS THE UPPER BITE OF THE RESULT.
00200                   * ANS1. IS THE LOWER BITE OF THE RESULT.
00210                   ************************************************************
00220                   *
00230  0000             NB1A    RMB    1         *SHIFT MULTIPLICAND STORE.
00240                   *
00250  0000             NB1     RMB    1         *MULTIPLICAND
00260  0001             NB2     RMB    1         *MULTIPLIER
00270  0002             ANS2    RMB    1         *UPPER BYTE OF RESULT
00280  0003             ANS1    RMB    1         *LOWER BYTE OF RESULT
00290  0004             *
00300                   *
00310  0010             MULT    CLR A
00320  0010 4F          STA A   ANS2
00330  0011 97 03       STA A   ANS1
00340  0013 97 04       STA A   NB1A
00350  0015 97 00       LDA A   NB2       *NB2=MULTIPLIER
00360  0017 96 02       BRA     LOOP1
00370  0019 20 06       ASL     NB1       *SHIFT MULTIPLICAND LEFT.
00380  001B 78 00 01    ROL     NB1A      *NB1A=UPPER BYTE OF MULTIPLICAND
00390  001E 79 00 00    LSR A             *SHIFT MULTIPLIER RIGHT
00400  0021 44          BCC     NOADD     *SHIFT AND DON'T ADD
00410  0022 24 0D       LDA B   NB1       *ADD SHIFTED MULTIPLICAND-
00420  0024 D6 01       ADD B   ANS1      *TO ANS1 AND ANS2
00430  0026 DB 04       STA B   ANS1
00440  0028 D7 04       LDA B   NB1A
00450  002A D6 00       ADC B   ANS2      *ANS2=UPPER BYTE OF RESULT
00460  002C D9 03       STA B   ANS2
00470  002E D7 03       TST A
00480  0030 4D          BNE     LOOP2     *START SHIFTING AGAIN.
00490  0031 26 E8       RTS               *FINISHED!!!
00400  0033 39          MON
00410  0033 39
00420
```

SYMBOL TABLE:

ANS1	0004	ANS2	0003	END	0033	LOOP1	0021	LOOP2	001B
MULT	0010	NB1	0001	NB1A	0000	NB2	0002	NOADD	0031

PROGRAM STOP AT 0

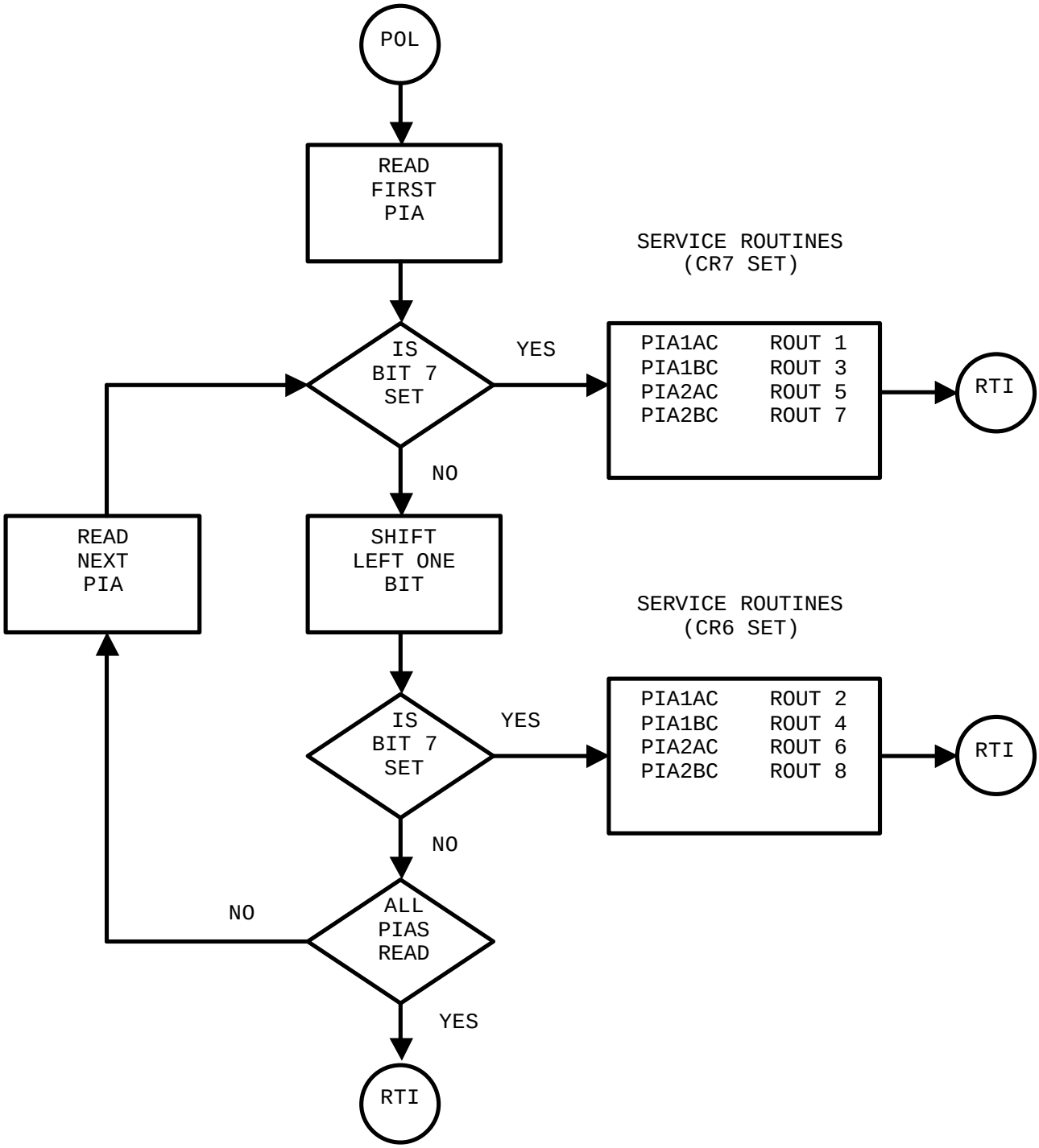
USED 20.24 UNITS

PIA POLING ROUTINE #1

The following routine illustrates one of the various techniques of determining which PIA has generated an interrupt. Recall that each PIA has an A side and a B side which may cause the IRQ line to go low thus generating an interrupt. All the PIA interrupt lines are tied together and connected to the one interrupt input pin (IRQ) of the MPU. Consequently, when an interrupt is generated, some bit 6 or bit 7 of a PIA is set. The only way to determine where the interrupt came from is to poll bit 6 and bit 7 of each PIA control register to see if it is a "1" (thus an interrupt).

This routine polls the control registers of two PIA's. It reads the contents of each control register and executes the BMI instruction which effectively checks to see if bit 7 is set. If bit 7 is not set, a ROL A instruction is executed which shifts bit 6 into bit 7 thus permitting use of the BMI instruction again. Once a set control bit is detected, it branches to a subroutine to service that particular interrupt. After servicing the interrupt, an RTI instruction is executed which causes the processor to return to whatever it was doing before the interrupt.

Flow Chart for PIA #1 Poling Routine



Source Program for PIA #1 Poling Routine

```
EDU         12:09EST     02/07/75
100  NAM POLL
110  OPT MEM
120 PIA1AC EQU $4005
130 PIA1BC EQU $4007
140 PIA2AC EQU $4009
150 PIA2BC EQU $4008
200  ORG $100
210 POLL LDA A PIA1AC
220  BMI ROUT1
230  ROL A
240  BMI ROUT2
250  LDA A PIA1BC
260  BMI ROUT3
270  ROL A
280  BMI ROUT4
290  LDA A PIA2AC
300  BMI ROUT5
310  ROL A
320  BMI ROUT6
330  LDA A PIA2BC
340  BMI ROUT7
350  ROL A
360  BMI ROUT8
370  RTI
380 ROUT1 NOP *THIS IS PIA1AC CA1 SERVICE ROUTINE
390  RTI
400 ROUT2 NOP *THIS IS PIA1AC CA2 SERVICE ROUTINE
410  RTI
420 ROUT3 NOP *THIS IS PIA1BC CB1 SERVICE ROUTINE
430  RTI
440 ROUT4 NOP *THIS IS PIA1BC CB2 SERVICE ROUTINE
450  RTI
460 ROUT5 NOP *THIS IS PIA2AC CA1 SERVICE ROUTINE
470  RTI
480 ROUT6 NOP *THIS IS PIA2AC CA2 SERVICE ROUTINE
490  RTI
500 ROUT7 NOP *THIS IS PIA2BC CB1 SERVICE ROUTINE
510  RTI
520 ROUT8 NOP *THIS IS PIA2BC CB2 SERVICE ROUTINE
530  RTI
540  MON
```

Assembled Program for PIA #1 Poling Routine

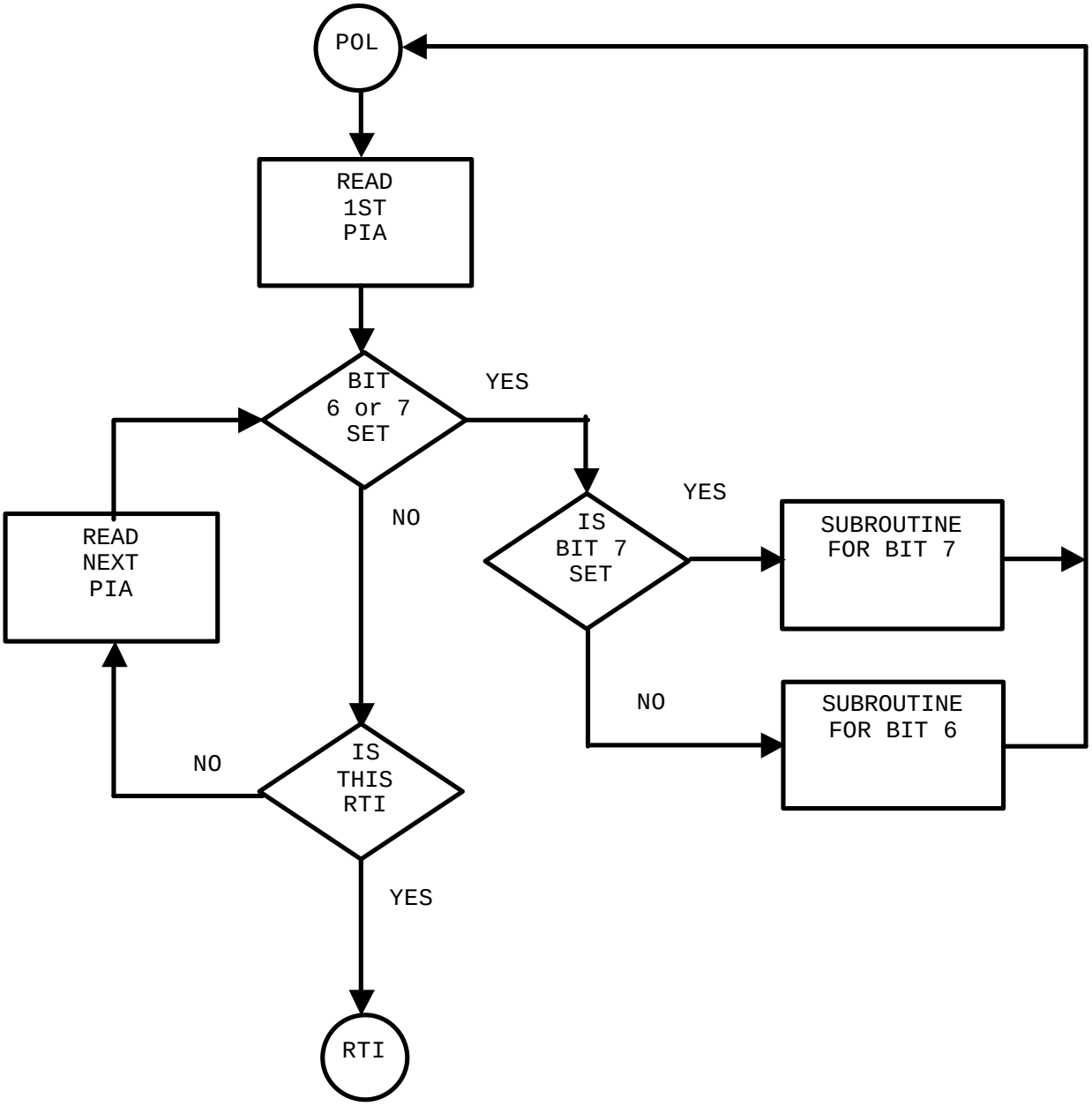
```
00100                        NAM    POLL
00110                        OPT    MEM
00120 4005          PIA1AC   EQU    $4005
00130 4007          PIA1BC   EQU    $4007
00140 4009          PIA2AC   EQU    $4009
00050 4008          PIA2BC   EQU    $4008
00200 0100                   ORG    $100
00210 0100 B6 4005  POLL     LDA A  PIA1AC
00220 0103 2B 1C             BMI    ROUT1
00230 0105 49                ROL A
00240 0106 2B 1B             BMI    ROUT2
00250 0108 B6 4007           LDA A  PIA1BC
00260 010B 2B 18             BMI    ROUT3
00270 010D 49                ROL A
00280 010E 2B 17             BMI    ROUT4
00290 0110 B6 4009           LDA A  PIA2AC
00300 0113 2B 14             BMI    ROUT5
00310 0115 49                ROL A
00320 0116 2B 13             BMI    ROUT6
00330 0118 B6 4008           LDA A  PIA2BC
00340 011B 2B 10             BMI    ROUT7
00350 011D 49                ROL A
00360 011E 2B 0F             BMI    ROUT8
00370 0120 3B                RTI
00380 0121 01       ROUT1    NOP           *THIS IS PIA1AC CA1 SERVICE ROUTINE
00390 0122 3B                RTI
00400 0123 01       ROUT2    NOP           *THIS IS PIA1AC CA2 SERVICE ROUTINE
00410 0124 3B                RTI
00420 0125 01       ROUT3    NOP           *THIS IS PIA1BC CB1 SERVICE ROUTINE
00430 0126 3B                RTI
00440 0127 01       ROUT4    NOP           *THIS IS PIA1BC CB2 SERVICE ROUTINE
00450 0128 3B                RTI
00460 0129 01       ROUT5    NOP           *THIS IS PIA2AC CA1 SERVICE ROUTINE
00470 012A 3B                RTI
00480 012B 01       ROUT6    NOP           *THIS IS PIA2AC CA2 SERVICE ROUTINE
00490 012C 3B                RTI
00500 012D 01       ROUT7    NOP           *THIS IS PIA2BC CB1 SERVICE ROUTINE
00510 012E 3B                RTI
00520 012F 01       ROUT8    NOP           *THIS IS PIA2BC CB2 SERVICE ROUTINE
00530 0130 3B                RTI
00540                        MON
```

PIA POLING ROUTINE #2

The routine presented on the following pages describes a way of determining where an interrupt came from out of a possible 16. (4 PIA's). Recall each PIA has an A side and a B side. Each side of each PIA has a control register of which bit 6 and/or bit 7 may get set if an interrupt came in on the interrupt lines (CA1, CA2, CB1, and CB2). As mentioned above, this is a way of poling the control registers for the interrupts. There are many other ways of accomplishing this task.

This routine, called "POL" will read the control register of each PIA, starting with the first PIA and determine if bit 6 or bit 7 is set, thus indicating an interrupt. When an interrupt has been detected via bit 6 or bit 7 of the control register, the MPU will branch to a subroutine designated to service that particular interrupt. On completion of servicing an interrupt, the MPU starts the poling sequence again with the first PIA. Only after all control registers have been poled, and no interrupts detected, does the MPU return to the program it was executing before it was interrupted. A branch to POL (BRA POL) instruction must be the last instruction of each servicing routine.

Flow Chart for PIA #2 Poling Routine



Source Program for PIA #2 Poling Routine

```
100  NAM PIA
105  OPT MEM
110  ORG 0
130 SETX RMB 2
140  SPC 4
150  ORG $2004
160 PIA1AD RMB 1
165 PIA1AC RMB 1
170 PIA1BD RMB 1
175 PIA1BC RMB 1
180 PIA2AD RMB 1
185 PIA2AC RMB 1
190 PIA2BD RMB 1
195 PIA2BC RMB 1
200  ORG $2010
210 PIA3AD RMB 1
215 PIA3AC RMB 1
220 PIA3BD RMB 1
225 PIA3BC RMB 1
230  ORG $2020
240 PIA4AD RMB 1
245 PIA4AC RMB 1
250 PIA4BD RMB 1
255 PIA4BC RMB 1
260  SPC 4
270  ORG $1000
280 * $1000 THRU $102D ARE THE SERVICING ROUTINES
390 * FOR THE 4 PIAS
300  JMP ROUT1
310  JMP ROUT2
320  JMP ROUT3
330  JMP ROUT4
340  JMP ROUT5
350  JMP ROUT6
360  JMP ROUT7
370  JMP ROUT8
380  JMP ROUT9
390  JMP ROUT10
400  JMP ROUT11
410  JMP ROUT12
420  JMP ROUT13
430  JMP ROUT14
440  JMP ROUT15
450  JMP ROUT16
```

Source Program for PIA #2 Poling Routine

```
470 * THIS IS A SAMPLE ROUTINE FOR
480 * POLLING PIA INTERUPTS
490  SPC3
500 POL LDA A #$10
510  STA A SETX
520  CLR B
525  LDX #0
530  LDA A PIA1AC
540  AND A #%11000000
550  BNE INTER
560  ADD B #6
570  LDA A PIA1BC
580  AND A #%11000000
590  BNE INTER
600  ADD B #6
610  LDA A PIA2AC
620  AND A #%11000000
630  BNE INTER
640  ADD B #6
650  LDA A PIA2BC
660  AND A #%11000000
670  BNE INTER
680  ADD B #6
690  LDA A PIA3AC
700  AND A #%11000000
710  BNE INTER
720  ADD B #6
730  LDA A PIA3BC
740  AND A #%11000000
750  BNE INTER
760  ADD B #6
770  LDA A PIA4AC
780  AND A #%11000000
790  BNE INTER
800  ADD B #6
810  LDA A PIA4BC
820  AND A #%11000000
830  BNE INTER
840  RTI
845  SPC3
850 INTER STA B SETX+1
860  LDX SETX
865  TST A
870  BMI SERVE
880  ADD B #3
885  STA B SETX+1
890  LDX SETX
900 SERVE JMP 0,X JUMP TO A SERVICE ROUTINE
901 *             BASED ON THE VALUE OF X
910  MON
```