Module 5

Arithmetic, Logic Instructions and Programs

ADD Instruction

- Add the source operand to register A and put the result in A.

  ADD A, source

  A + source → A

```
MOV   A,#25H   ;load 25H into A
ADD   A,#34H   ;add 34H to A, now A=59H
```

  - The destination operand is always in A.
  - The instruction could change CY,AC,OV and P.

ADDC

ADD with carry

ADDC A, source

Write a program to add two 16-bit numbers. The numbers are 3CE7H and 3B8DH. Place that sum in R7 and R6; R6 should have the lower byte.

Solution:

```
CLR   C         ;make CY=0
MOV   A,#0E7H   ;load the low byte now A=E7H
ADD   A,#8DH    ;add the low byte,A=74H and CY=1
MOV   R6,A      ;save the low byte in R6
MOV   A,#3CH    ;load the high byte
ADDC  A,#3BH    ;add with the carry
MOV   R7,A      ;save the high byte of the sum
```

INC,DEC

- Supports the following modes
  - `INC A`
  - `INC DATA ADDR`
  - `INC Rn`
  - `INC @R0`
  - `INC @R1`

  - `DEC A`
  - `DEC DATA ADDR`
  - `DEC Rn`
  - `DEC @R0`
  - `DEC @R1`

CY is not affected even if value FF is incremented to 00.

Exercise

- Assume an 8 byte unsigned number x located at 30h-37h
- Assume 8 byte y located at 40h-47h
- Calculate z = x+y and place it at 30-38h

Solution

```
org 0h
mov r2,#08h
; 8 numbers to add
mov r0,#30h
; point to first number
mov r1,#40h
; point to second number
clr c
; clear carry
```

```
again: mov a,@r0
addc a,@r1
mov @r0,a
inc r0
inc r1
djnz r2,again
mov @r0,#00h
jnc son
inc @r0
son: sjmp son
end
```

Packed/Unpacked BCD

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- In Unpacked BCD, a byte is used to contain a BCD number.
  - 0000 0101 is unpacked BCD for 5
  - 0000 1001 is unpacked BCD for 9
- In Packed BCD, a byte has two BCD numbers.
  - 0101 1001 is packed BCD for 59.
  - It is twice as efficient in storing data.

Addition of BCD Numbers

- ADD and ADDC are just for hexadecimal!
- To calculate 17+18=35.

```
MOV   A,#17H
ADD  A,#28H
```

$$\begin{array}{r} 1\ 7 \\ +\ 1\ 8 \\ \hline 2\ F \end{array}$$

- The programmer must add 6 to the low digit such that a carry occurs.

$$\begin{array}{r} 2\ F \\ +\ 0\ 6 \\ \hline 3\ 5 \end{array}$$

If lower nibble>9 or AC=1, add 6 to lower nibble.

If higher nibble>9 or CY=1, add 6 to higher nibble.

DA A (Decimal Adjust A)

- Decimal adjust for addition
- DA instruction will add 6 to the lower nibble or higher nibble if needed.

DA A

```
MOV A,#47H
MOV B, #55H
ADD A,B
DA A
```

$$\begin{array}{r} 47 \\ +\ 55 \\ \hline 9C \\ +\ 06 \\ \hline A2 \\ +\ 60 \\ \hline 1\ \ 02 \end{array}$$

CY

SUBB

- Subtraction with borrow

  SUBB A, source ; A = A – source – CY

```
CLR   C
MOV   A,#3FH
MOV   R3,#23H
SUBB  A,R3
```

```
   3 F
-  2 3
------
   1 C
```

- Take the 2s complement of the subtrahend
- Add it to the minuend (A)
- Invert the carry

```
    0011 1111
+   1101 1101
-------------
1   0001 1100
```

CF = 0 (third step)

- If CY = 0 after SUBB the result is positive
- If CY = 1, the result is negative and the destination has the 2's complement of the result.

Analyze the following program:

```
      CLR   C
      MOV   A,#4CH   ;load A =4CH
      SUBB  A,#6EH  ;subtract 6E from A
      JNC   NEXT    ;jump not carry (CY=0)
      CPL   A       ;get 2's complement by
      INC   A       ;  yourself
NEXT: MOV   R1,A    ;save A in R1
```

Solution:

```
   4C   0100 1100            0100 1100
 - 6E   0110 1110          + 1001 0010
 -----                     -----------
  -22                      0 1101 1110(A)
CY=1, the result is negative.
Get the 2's complement of A =0010 0010=22.
```

Example

Analyze the following program:

```
CLR   C
MOV   A,#62H         27 62        0110 0010  62H
SUBB  A,#96H       - 12 96      + 0110 1010
                   -------      -----------
MOV   R7,A           14 CC      0 1100 1100
MOV   A,#27H
SUBB  A,#12H                      0010 0110
MOV   R6,A                      + 1110 1110
                                -----------
                                1 0001 0100
```

Solution:

After the SUBB, A = 62H-96H = CCH and CY=1 indicating there is a borrow.

Since CY = 1, when SUBB is executed the second time A = 27H-12H-1 =14H. Therefore, we have 2762H-1296H =14CCH.

Multiplication/Division of Unsigned Numbers

MUL AB

Multiplication	Operand 1	Operand 2	Result
byte × byte	A	B	A = low byte, B = high byte

DIV AB

Division	Numerator	Denominator	Quotient	Remainder
byte / byte	A	B	A	B

(If B = 0, then OV = 1 indicating an error)

An application for DIV

- There are times when an analog-to-digital converter is connected to a port.
- The ADC represents some quantity such as temperature or pressure.
- The 8-bit ADC provides data in hex.
- This hex data must be converted to decimal for display. We divide it by 10 repeatedly until the quotient is less than 10.
- Ex: FDh is read from the port

  FDh= OAh * 19h + 03h

  19h = 0Ah * 2 + 05H

  FDh = 253d

Example

Write a program to get hex data in the range of 00 – FFH from port 1 and convert it to decimal. Save the digits in R7, R6 and R5, where the least significant digit is in R7.

```
 MOV A,#0FFH
MOV P1,A        ;make P1 an input port
MOV A,P1        ;read data from P1
MOV B,#10       ;B=0A hex (10 dec)
DIV AB          ;divide by 10
MOV R7,B        ;
MOV B,#10       ;P1 has max value 255
DIV AB          ;3 bytes to save 3 decimals
MOV R6,B        ;Twice DIV are used
MOV R5,A        ;
```

16-Bit Multiplication

- First number in R6 and R7 (MSB R6)
- Second number in R4 and R5 (MSB R4)
- The result will be stored in R0,R1,R2,R3 (MSB R0)

	Byte 4	Byte 3	Byte 2	Byte 1
*			R6	R7
*			R4	R5
=	**R0**	**R1**	**R2**	**R3**

- Multiply R5 by R7, leaving the 16-bit result in R2 and R3.
- Multiply R5 by R6, adding the 16-bit result to R1 and R2.
- Multiply R4 by R7, adding the 16-bit result to R1 and R2.
- Multiply R4 by R6, adding the 16-bit result to R0 and R1.

16 bit multiplication

.	Byte 4	Byte 3	Byte 2	Byte 1
			62	**30**
*			**43**	**2E**
=			08	A0
		11	9C	
		0C	90	
	19	A6		
=	**19**	**C4**	**34**	**A0**

For 16-bit division visit http://www.8052.com/div16.phtml

Code

Step 1. Multiply R5 by R7, leaving the 16-bit result in R2 and R3.

```
MOV A,R5 ;Move the R5 into the Accumulator
MOV B,R7 ;Move R7 into B
MUL AB ;Multiply the two values
MOV R2,B ;Move B (the high-byte) into R2
MOV R3,A ;Move A (the low-byte) into R3
```

Step 2. Multiply R5 by R6, adding the 16-bit result to R1 and R2.

```
MOV A,R5 ;Move R5 back into the Accumulator
MOV B,R6 ;Move R6 into B
MUL AB ;Multiply the two values
ADD A,R2 ;Add the low-byte into the value already in R2
MOV R2,A ;Move the resulting value back into R2
MOV A,B ;Move the high-byte into the accumulator
ADDC A,#00h ;Add zero (plus the carry, if any)
MOV R1,A ;Move the resulting answer into R1
```

16 bit multiplication

	Byte 4	Byte 3	Byte 2	Byte 1
			R6 ← **62**	R7 ← **30**
*			R4 ← **43**	R5 ← **2E**
=			08	A0
		11	9C	
		11	A4	A0
		0C	90	
		1E	34	A0
	19	A6		
=	**19**	**C4**	**34**	**A0**
	R0	R1	R2	R3

Step 3. Multiply R4 by R7, adding the 16-bit result to R1 and R2.

```
MOV A,R4 ;Move R4 into the Accumulator
MOV B,R7 ;Move R7 into B
MUL AB ;Multiply the two values
ADD A,R2 ;Add the low-byte into the value already in R2
MOV R2,A ;Move the resulting value back into R2
MOV A,B ;Move the high-byte into the accumulator
ADDC A,R1 ;Add the current value of R1 (plus any carry)
MOV R1,A ;Move the resulting answer into R1.
MOV A,#00h ;Load the accumulator with zero
ADDC A,#00h ;Add zero (plus the carry, if any)
MOV R0,A ;Move the resulting answer to R1.
```

Step 4. Multiply R4 by R6, adding the 16-bit result to R0 and R1.

```
MOV A,R4 ;Move R4 back into the Accumulator
MOV B,R6 ;Move R6 into B
MUL AB ;Multiply the two values
ADD A,R1 ;Add the low-byte into the value already in R1
MOV R1,A ;Move the resulting value back into R1
MOV A,B ;Move the high-byte into the accumulator
ADDC A,R0 ;Add it to the value already in R0 (plus any carry)
MOV R0,A ;Move the resulting answer back to R0
```

ANL, ORL

ANL destination-byte,source-byte

```
MOV   A,#35H   ;0011 0101
ANL   A,#0FH   ;0000 1111  => A=0000 0101
```

- No effect on any of the flags.
  - ANL is often used to mask (set to 0) certain bits of an operands.

ORL destination-byte,source-byte

```
MOV   A,#35H   ;0011 0101
ORL   A,#0FH   ;0000 1111  => A=0011 1111
```

- No effect on any of the flags.
- ORL is often used to set certain bits of an operand to 1.

XRL,CPL

XRL destination-byte,source-byte

```
MOV   A,#35H   ;0011 0101
XRL   A,#0FH   ;0000 1111   => A=0011 1010
```

- No effect on any of the flags.
- XRL is often used (1) to clear a register, (2) to see if two registers have the same value or (3) to toggle bits of an operand.
  - XRL A,A ; clears A

CPL A

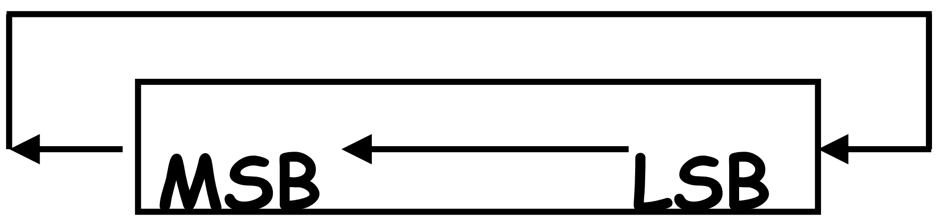
```
MOV   A,#55H   ;0101 0101
CPL   A        ;1010 1010
ADD A,#1         ; two's complement
```

- No effect on any of the flags.
- This is also called 1's complement.

RL (Rotate A Left)

RL A

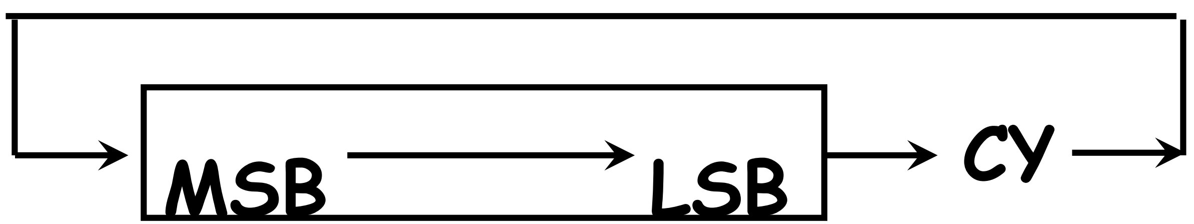
```
MOV   A,#36H     ;A=0011 0110
RL    A          ;A=0110 1100
RL    A          ;A=1101 1000
RL    A          ;A=1011 0001
RL    A          ;A=0110 0011
```



RRC (Rotate right thru carry)

RRC A

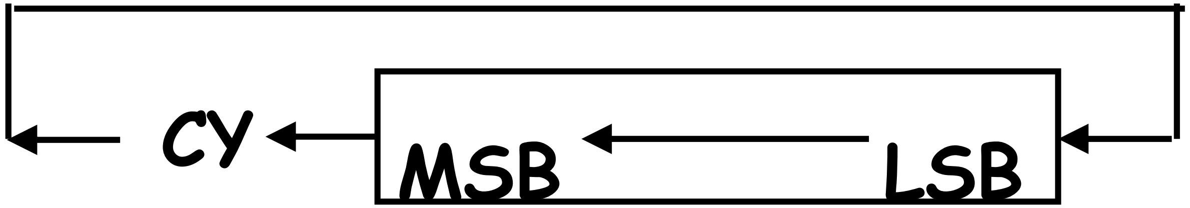
```
MOV  A,#36H     ;A=0011 0110, CY=0
RRC  A          ;A=0001 1011, CY=0
RRC  A          ;A=0000 1101, CY=1
RRC  A          ;A=1000 0110, CY=1
RRC  A          ;A=1100 0011, CY=0
```



RLC (Rotate left carry)

RLC A

```
MOV   A,#36H        ;A=0011 0110, CY=1
RLC   A             ;A=0110 1101, CY=0
RLC   A             ;A=1101 1010, CY=0
RLC   A             ;A=1011 0100, CY=1
RLC   A             ;A=1001 1001, CY=1
```



Example

- Write a program that finds the number of 1s in a given byte.

Solution:

```
        MOV R1,#0     ;R1 keeps the number of 1s
        MOV R7,#8     ;counter=08 rotate 8 times
        MOV A,#97H    ;A=10010111H
AGAIN:  RLC A         ;rotate it through CY once
        JNC NEXT      ;check for CY
        INC R1        ;if CY=1 then R1++
NEXT:   DJNZ R7,AGAIN
```

Exam Question

Q1) (35 points) Assume A:R1 holds a two byte binary number; Accumulator (A) holds the high part and R1 holds the low part of the binary number. Write a subroutine with the name CALC that calculates round(number/4) and overwrites it into A:R1. For example, if the number is 602d, then A:R1 = 02h:5Ah. For this example, round(602d/4)=151d=97h and upon return from the subroutine A:R1 will read 00h:97h.

DIV AB is not to be used!

```
CALC: CLR C  ; insert a zero
      RRC A  ; right rotate once
      PUSH ACC  ; save acc
      MOV A,R1  ;
      RRC A  ;
      MOV R1,A  ; right rotate R2 once
      POP ACC  ; recover A , now we have divided by 2
      CLR C ; go for another divide
      RRC A
      PUSH ACC
      MOV A,R1
      RRC A
      MOV R1,A
      POP ACC
      JNC SKIP   ; the final carry determines whether we have to increment or not
      INC R1
      CJNE R1,#00h,SKIP
      INC A
SKIP: RET
```

SWAP

SWAP A

```
MOV  A,#72H      ;A=72H
SWAP A           ;A=27H
```

- Only works for the accumulator

before:	D7 – D4	D3 – D0
after:	D3 – D0	D7 – D4

before:	0111	0010
after:	0010	0111

XCH

- The **XCH** instruction loads the accumulator with the byte value of the specified operand while simultaneously storing the previous contents of the accumulator in the specified operand.
  - **XCH A, @Ri**
  - **XCH A, direct**
  - **XCH A, Rn**

Conversion from BCD and ASCII

- There is a real time clock (RTC) in many new microcontrollers.
  - Ex: DS5000T has RTC
- RTC keeps the time (hour, minute, second) and date (year, month, day) even when the power is off.
- This data is provided in packed BCD.
- For this data to be displayed (ex: on an LCD), it must be in ASCII format.
- We show below instructions in the conversion of BCD and ASCII

ASCII code for digits 0-9

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

Packed BCD

Packed BCD to ASCII

- To convert packed BCD to ASCII

  Step 1. It must be converted to unpacked BCD first.

```
MOV A,#29H   ;It means 29₁₀
ANL A,#0FH   ;get the lower nibble
```

  Step 2. The unpacked BCD is tagged with 30H

```
ORL A,#30H   ;make it an ASCII,A=39H '9'
```

  Step 3. Similar procedure for the significant digit

ASCII to Packed BCD

- To convert ASCII to packed BCD

  Step 1. It must be converted to unpacked BCD first.

```
MOV A,#'2'    ;A=32H
ANL A,#0FH    ;get the lower nibble
MOV B,#'9'  ;
ANL B,#0FH ;get the lower nibble
```

  Step 2. Combine them to the packed BCD

```
SWAP A          ;become upper nibble A=20H
ORL A,B       ;packed BCD,A=29H
```

Example

Assume that register A has packed BCD, write a program to convert packed BCD to two ASCII numbers and place them in R2 and R6.

Solution:

```
MOV A,#29H    ;packed BCD
MOV R3,A      ; save A in R3
ANL A,#0FH    ;Lower nibble: A=09H
ORL A,#30H    ;make it an ASCII, A=39H ('9')
MOV R6,A      ;R6=39H ASCII char
MOV A,R3    ; retrieve A
ANL A,#0F0H   ;upper nibble: A=20H
SWAP A        ;A=02H, equals to "RR A" 4 times
ORL A,#30H    ;A=32H,ASCII char.'2'
MOV R2,A      ;R2=32H ASCII char
```