ARITHMETIC & LOGIC INSTRUCTIONS AND PROGRAMS

The 8051 Microcontroller and Embedded Systems: Using Assembly and C

Department of software Engineering,
Karachi Campus

ARITHMETIC INSTRUCTIONS

Addition of Unsigned Numbers

```
ADD  A,source  ;A = A + source
```

- The instruction ADD is used to add two operands
  - Destination operand is always in register A
  - Source operand can be a register, immediate data, or in memory
  - Memory-to-memory arithmetic operations are never allowed in 8051 Assembly language

Show how the flag register is affected by the following instruction.

```
MOV A,#0F5H ;A=F5 hex
ADD A,#0BH  ;A=F5+0B=00
```

Solution:

```
      F5H        1111 0101
  +   0BH    +   0000 1011
     100H        0000 0000
```

CY =1, since there is a carry out from D7
PF =1, because the number of 1s is zero (an even number), PF is set to 1.
AC =1, since there is a carry from D3 to D4

ARITHMETIC INSTRUCTIONS

ADDC and Addition of 16-Bit Numbers

- When adding two 16-bit data operands, the propagation of a carry from lower byte to higher byte is concerned

```
        1
        3C E7
 +      3B 8D
       78 74
```

When the first byte is added (E7+8D=74, CY=1).
The carry is propagated to the higher byte, which result in 3C + 3B + 1 =78 (all in hex)

Write a program to add two 16-bit numbers. Place the 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
MOV  R6, A      ;save the low byte sum in R6
MOV  A, #3CH    ;load the high byte
ADDC A, #3BH    ;add with the carry
MOV  R7, A      ;save the high byte sum
```

```
DA A ;decimal adjust for addition
```

- The DA instruction is provided to correct the aforementioned problem associated with BCD addition
  - The DA instruction will add 6 to the lower nibble or higher nibble if need

Example:

```
MOV A,#47H     ;A=47H first BCD operand
MOV B,#25H     ;B=25H second BCD operand
ADD A,B        ;hex(binary) addition(A=6CH)
DA  A          ;adjust for BCD addition
               (A=72H)
```

DA works only after an ADD, but not after INC

The "DA" instruction works only on A. In other word, while the source can be an operand of any addressing mode, the destination must be in register A in order for DA to work.

ARITHMETIC INSTRUCTIONS

DA Instruction (cont')

Summary of DA instruction

- After an ADD or ADDC instruction
  1. If the lower nibble (4 bits) is greater than 9, or if AC=1, add 0110 to the lower 4 bits
  2. If the upper nibble is greater than 9, or if CY=1, add 0110 to the upper 4 bits

Example:

	HEX	BCD	
	29	0010 1001	
+	18	0001 1000	
	41	0100 0001	AC=1
+	6	0110	
	47	0100 0111	

Since AC=1 after the addition, "DA A" will add 6 to the lower nibble.
The final result is in BCD format.

ARITHMETIC INSTRUCTIONS

Subtraction of Unsigned Numbers

- In many microprocessor there are two different instructions for subtraction: SUB and SUBB (subtract with borrow)
  - In the 8051 we have only SUBB
  - The 8051 uses adder circuitry to perform the subtraction

```
SUBB A,source ;A = A - source - CY
```

- To make SUB out of SUBB, we have to make CY=0 prior to the execution of the instruction
  - Notice that we use the CY flag for the borrow

ARITHMETIC INSTRUCTIONS

Subtraction of Unsigned Numbers (cont')

- SUBB when CY = 0
  1. Take the 2's complement of the subtrahend (source operand)
  2. Add it to the minuend (A)
  3. Invert the carry

```
        CLR     C
        MOV     A,#4C    ;load A with value 4CH
        SUBB    A,#6EH   ;subtract 6E from A
        JNC     NEXT     ;if CY=0 jump to NEXT
        CPL     A   ;if CY=1, take 1's complement
        INC     A   ;and increment to get 2's comp
NEXT:   MOV     R1,A     ;save A in R1
```

Solution:

```
    4C      0100 1100          0100 1100
-   6E      0110 1110          1001 0010
   -22                        01101 1110
```

① 2's complement

② +

③ Invert carry

CY =1

CY=0, the result is positive; CY=1, the result is negative and the destination has the 2's complement of the result

SUBB when CY = 1

- This instruction is used for multi-byte numbers and will take care of the borrow of the lower operand

A = 62H – 96H – 0 = CCH
CY = 1

```
        CLR     C
        MOV     A,#62H    ;A=62H
        SUBB    A,#96H    ;62H-96H=CCH with CY=1
        MOV     R7,A      ;save the result
        MOV     A,#27H    ;A=27H
        SUBB    A,#12H    ;27H-12H-1=14H
        MOV     R6,A      ;save the result
```

A = 27H - 12H - 1 = 14H
CY = 0

Solution:

We have 2762H - 1296H = 14CCH.

ARITHMETIC INSTRUCTIONS

Unsigned Multiplication

- The 8051 supports byte by byte multiplication only
  - The byte are assumed to be unsigned data

```
MUL AB  ;AxB, 16-bit result in B, A
```

```
        MOV     A,#25H      ;load 25H to reg.  A
        MOV     B,#65H      ;load 65H to reg.  B
        MUL     AB          ;25H * 65H = E99 where
                            ;B = OEH and A = 99H
```

Multiplication	Operand1	Operand2	Result
Byte x byte	A	B	B = high byte A = low byte

ARITHMETIC INSTRUCTIONS

Unsigned Division

- The 8051 supports byte over byte division only
  - The byte are assumed to be unsigned data

```
DIV  AB   ;divide A by B, A/B
```

```
MOV     A,#95    ;load 95 to reg.  A
MOV     B,#10    ;load 10 to reg.  B
MUL     AB       ;A = 09(quotient) and
                 ;B = 05(remainder)
```

Unsigned Division Summary (DIV AB)

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

CY is always 0
If B ≠ 0, OV = 0
If B = 0, OV = 1 indicates error

SIGNED ARITHMETIC INSTRUCTIONS

Signed 8-bit Operands

- D7 (MSB) is the sign and D0 to D6 are the magnitude of the number
  - If D7=0, the operand is positive, and if D7=1, it is negative

- Positive numbers are 0 to +127
- Negative number representation (2's complement)
  1. Write the magnitude of the number in 8-bit binary (no sign)
  2. Invert each bit
  3. Add 1 to it

SIGNED ARITHMETIC INSTRUCTIONS

Signed 8-bit Operands (cont')

Show how the 8051 would represent -34H

Solution:

1. 0011 0100 34H given in binary
2. 1100 1011 invert each bit
3. 1100 1100 add 1 (which is CC in hex)

Signed number representation of -34 in 2's complement is CCH

Decimal	Binary	Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
...	... ...	...
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
...	... ...	...
+127	0111 1111	7F

SIGNED ARITHMETIC INSTRUCTIONS

Overflow Problem

- If the result of an operation on signed numbers is too large for the register
  - An overflow has occurred and the programmer must be noticed

Examine the following code and analyze the result.

```
MOV    A,#+96       ;A=0110 0000 (A=60H)
MOV    R1,#+70      ;R1=0100 0110(R1=46H)
ADD    A,R1         ;A=1010 0110
                    ;A=A6H=-90,INVALID
```

Solution:

```
     +96    0110 0000
  +  +70    0100 0110
  -------   ---------
  +  166    1010 0110 and OV =1
```

According to the CPU, the result is -90, which is wrong. The CPU sets OV=1 to indicate the overflow

SIGNED ARITHMETIC INSTRUCTIONS

OV Flag

- In 8-bit signed number operations, OV is set to 1 if either occurs:
  1. There is a carry from D6 to D7, but no carry out of D7 (CY=0)
  2. There is a carry from D7 out (CY=1), but no carry from D6 to D7

```
MOV  A,#-128    ;A=1000 0000(A=80H)
MOV  R4,#-2     ;R4=1111 1110(R4=FEH)
ADD  A,R4       ;A=0111 1110(A=7EH=+126,INVALID)
     -128              1000 0000
  +    -2              1111 1110
     -130              0111 1110 and OV=1
```

OV = 1
The result +126 is wrong

SIGNED ARITHMETIC INSTRUCTIONS

OV Flag (cont')

```
MOV  A,#-2      ;A=1111 1110(A=FEH)
MOV  R1,#-5     ;R1=1111 1011(R1=FBH)
ADD  A,R1       ;A=1111 1001(A=F9H=-7,
                ;Correct, OV=0)
     -2              1111 1110
  +  -5              1111 1011
  -------            ---------
     -7              1111 1001 and OV=0
```

OV = 0
The result -7 is correct

```
MOV  A,#+7      ;A=0000 0111(A=07H)
MOV  R1,#+18    ;R1=0001 0010(R1=12H)
ADD  A,R1       ;A=0001 1001(A=19H=+25,
                ;Correct,OV=0)
      7              0000 0111
   + 18              0001 0010
  ------             ---------
     25              0001 1001 and OV=0
```

OV = 0
The result +25 is correct

SIGNED ARITHMETIC INSTRUCTIONS

OV Flag (cont')

- In unsigned number addition, we must monitor the status of CY (carry)
  - Use JNC or JC instructions
- In signed number addition, the OV (overflow) flag must be monitored by the programmer
  - `JB PSW.2` or `JNB PSW.2`

SIGNED ARITHMETIC INSTRUCTIONS

2's Complement

- To make the 2's complement of a number

```
        CPL     A        ;1's complement (invert)
        ADD     A,#1     ;add 1 to make 2's comp.
```

LOGIC AND COMPARE INSTRUCTIONS

AND

```
ANL destination,source
                ;dest = dest AND source
```

- This instruction will perform a logic AND on the two operands and place the result in the destination
  - The destination is normally the accumulator
  - The source operand can be a register, in memory, or immediate

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Show the results of the following.

```
        MOV   A,#35H    ;A = 35H
        ANL   A,#0FH    ;A = A AND 0FH

    35H       0 0 1 1 0 1 0 1
    0FH       0 0 0 0 1 1 1 1
    05H       0 0 0 0 0 1 0 1
```

ANL is often used to mask (set to 0) certain bits of an operand

LOGIC AND COMPARE INSTRUCTIONS

OR

```
ORL destination,source
                ;dest = dest OR source
```

- The destination and source operands are ORed and the result is placed in the destination
  - The destination is normally the accumulator
  - The source operand can be a register, in memory, or immediate

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Show the results of the following.

```
        MOV   A,#04H    ;A = 04
        ORL   A,#68H    ;A = 6C

04H       0 0 0 0 0 1 0 0
68H       0 1 1 0 1 0 0 0
6CH       0 1 1 0 1 1 0 0
```

ORL instruction can be used to set certain bits of an operand to 1

LOGIC AND COMPARE INSTRUCTIONS

XOR

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

```
XRL destination,source
            ;dest = dest XOR source
```

- This instruction will perform XOR operation on the two operands and place the result in the destination
  - The destination is normally the accumulator
  - The source operand can be a register, in memory, or immediate

Show the results of the following.

```
        MOV   A,#54H
        XRL   A,#78H

54H          0 1 0 1 0 1 0 0
78H          0 1 1 1 1 0 0 0
2CH          0 0 1 0 1 1 0 0
```

XRL instruction can be used to toggle certain bits of an operand

LOGIC AND COMPARE INSTRUCTIONS

XOR (cont')

The XRL instruction can be used to clear the contents of a register by XORing it with itself. Show how `XRL A,A` clears A, assuming that AH = 45H.

```
45H      0 1 0 0 0 1 0 1
45H      0 1 0 0 0 1 0 1
00H      0 0 0 0 0 0 0 0
```

Read and test P1 to see whether it has the value 45H. If it does, send 99H to P2; otherwise, it stays cleared.

XRL can be used to see if two registers have the same value

Solution:

```
        MOV P2,#00      ;clear P2
        MOV P1,#0FFH    ;make P1 an input port
        MOV R3,#45H     ;R3=45H
        MOV  A,P1       ;read P1
        XRL  A,R3
        JNZ EXIT        ;jump if A is not 0
        MOV P2,#99H
EXIT: ...
```

If both registers have the same value, 00 is placed in A. JNZ and JZ test the contents of the accumulator.

LOGIC AND COMPARE INSTRUCTIONS

Complement Accumulator

```
CPL A ;complements the register A
```

- This is called 1's complement

```
        MOV A, #55H
        CPL A            ;now A=AAH
                         ;0101 0101(55H)
                         ;becomes 1010 1010(AAH)
```

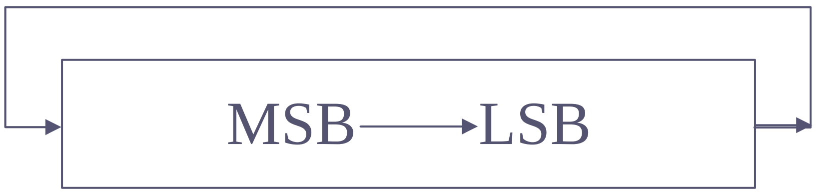
- To get the 2's complement, all we have to do is to to add 1 to the 1's complement

ROTATE INSTRUCTION AND DATA SERIALIZATION

Rotating Right and Left

```
RR A          ;rotate right A
```

- In rotate right
  - The 8 bits of the accumulator are rotated right one bit, and
  - Bit D0 exits from the LSB and enters into MSB, D7




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

ROTATE INSTRUCTION AND DATA SERIALIZATION

Rotating Right and Left (cont')

```
RL A          ;rotate left A
```

- In rotate left
  - The 8 bits of the accumulator are rotated left one bit, and
  - Bit D7 exits from the MSB and enters into LSB, D0

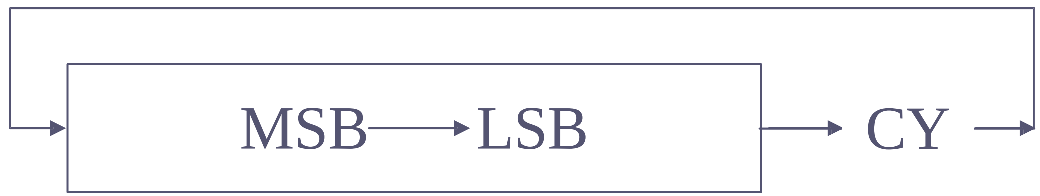

```
MOV A,#72H     ;A = 0111 0010
RL  A          ;A = 1110 0100
RL  A          ;A = 1100 1001
```

ROTATE INSTRUCTION AND DATA SERIALIZATION

Rotating through Carry

```
RRC A    ;rotate right through carry
```

- In RRC A
  - Bits are rotated from left to right
  - They exit the LSB to the carry flag, and the carry flag enters the MSB




```
CLR C          ;make CY = 0
MOV A,#26H     ;A = 0010 0110
RRC A          ;A = 0001 0011    CY = 0
RRC A          ;A = 0000 1001    CY = 1
RRC A          ;A = 1000 0100    CY = 1
```

ROTATE INSTRUCTION AND DATA SERIALIZATION

Rotating through Carry (cont')

```
RLC A    ;rotate left through carry
```

- In RLC A
  - Bits are shifted from right to left
  - They exit the MSB and enter the carry flag, and the carry flag enters the LSB

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

```
        MOV     R1,#0
        MOV     R7,#8       ;count=08
        MOV     A,#97H
AGAIN:  RLC     A
        JNC     NEXT        ;check for CY
        INC     R1          ;if CY=1 add to count
NEXT:   DJNZ    R7,AGAIN
```

ROTATE INSTRUCTION AND DATA SERIALIZATION

Serializing Data

- Serializing data is a way of sending a byte of data one bit at a time through a single pin of microcontroller
  - Using the serial port, discussed in Chapter 10
  - To transfer data one bit at a time and control the sequence of data and spaces in between them

ROTATE INSTRUCTION AND DATA SERIALIZATION

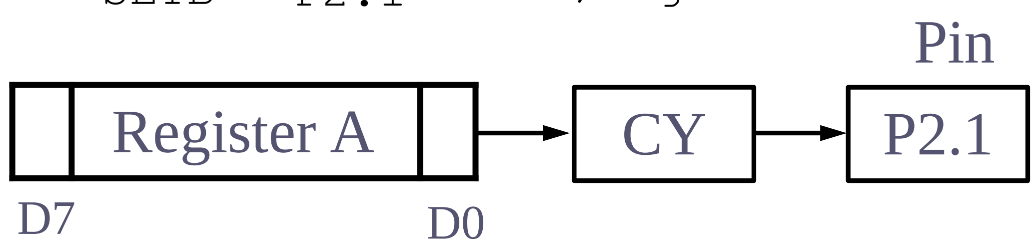
Serializing Data (cont')

- Transfer a byte of data serially by
  - Moving CY to any pin of ports P0 – P3
  - Using rotate instruction

Write a program to transfer value 41H serially (one bit at a time) via pin P2.1. Put two highs at the start and end of the data. Send the byte LSB first.

Solution:

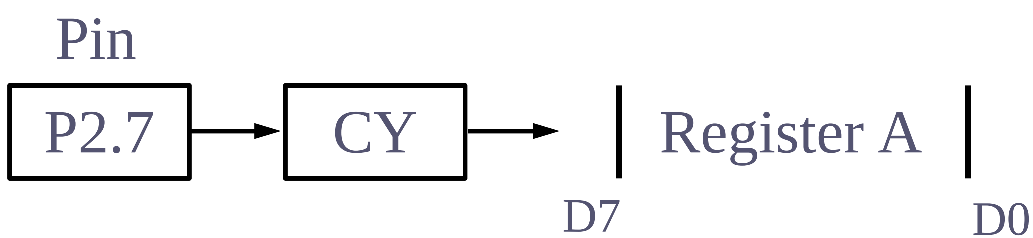
```
       MOV    A,#41H
       SETB   P2.1       ;high
       SETB   P2.1       ;high
       MOV    R5,#8
AGAIN: RRC    A
       MOV    P2.1,C     ;send CY to P2.1
       DJNZ   R5,HERE
       SETB   P2.1       ;high
       SETB   P2.1       ;high
```



Write a program to bring in a byte of data serially one bit at a time via pin P2.7 and save it in register R2. The byte comes in with the LSB first.

Solution:

```
        MOV     R5,#8
AGAIN:  MOV     C,P2.7      ;bring in bit
        RRC     A
        DJNZ    R5,HERE
        MOV     R2,A        ;save it
```



ROTATE INSTRUCTION AND DATA SERIALIZATION

Single-bit Operations with CY

- There are several instructions by which the CY flag can be manipulated directly

Instruction	Function
`SETB C`	Make CY = 1
`CLR C`	Clear carry bit (CY = 0)
`CPL C`	Complement carry bit
`MOV b,C`	Copy carry status to bit location (CY = b)
`MOV C,b`	Copy bit location status to carry (b = CY)
`JNC target`	Jump to target if CY = 0
`JC target`	Jump to target if CY = 1
`ANL C,bit`	AND CY with bit and save it on CY
`ANL C,/bit`	AND CY with inverted bit and save it on CY
`ORL C,bit`	OR CY with bit and save it on CY
`ORL C,/bit`	OR CY with inverted bit and save it on CY

ROTATE INSTRUCTION AND DATA SERIALIZATION

Single-bit Operations with CY

(cont')

Assume that bit P2.2 is used to control an outdoor light and bit P2.5 a light inside a building. Show how to turn on the outside light and turn off the inside one.

Solution:

```
        SETB    C           ;CY = 1
        ORL     C,P2.2      ;CY = P2.2 ORed w/ CY
        MOV     P2.2,C      ;turn it on if not on
        CLR     C           ;CY = 0
        ANL     C,P2.5      ;CY = P2.5 ANDed w/ CY
        MOV     P2.5,C     ;turn it off if not off
```

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

Solution:

```
        MOV     R1,#0       ;R1 keeps number of 1s
        MOV     R7,#8   ;counter, rotate 8 times
        MOV     A,#97H ;find number of 1s in 97H
AGAIN: RLC      A           ;rotate it thru CY
        JNC     NEXT        ;check CY
        INC     R1          ;if CY=1, inc count
NEXT:   DJNZ    R7,AGAIN    ;go thru 8 times
```

ROTATE INSTRUCTION AND DATA SERIALIZATION

SWAP

```
SWAP A
```

- It swaps the lower nibble and the higher nibble
  - In other words, the lower 4 bits are put into the higher 4 bits and the higher 4 bits are put into the lower 4 bits
- SWAP works only on the accumulator (A)

before :	D7-D4	D3-D0
after :	D3-D0	D7-D4

ROTATE INSTRUCTION AND DATA SERIALIZATION

SWAP (cont')

(a) Find the contents of register A in the following code.

(b) In the absence of a SWAP instruction, how would you exchange the nibbles? Write a simple program to show the process.

Solution:

```
(a)
        MOV     A,#72H      ;A = 72H
        SWAP    A           ;A = 27H
(b)
        MOV     A,#72H      ;A = 0111 0010
        RL      A           ;A = 0111 0010
        RL      A           ;A = 0111 0010
        RL      A           ;A = 0111 0010
        RL      A           ;A = 0111 0010
```

BCD AND ASCII APPLICATION PROGRAMS

Checksum Byte in ROM (cont')

- To calculate the checksum byte of a series of bytes of data
  - Add the bytes together and drop the carries
  - Take the 2's complement of the total sum, and it becomes the last byte of the series
- To perform the checksum operation, add all the bytes, including the checksum byte
  - The result must be zero
  - If it is not zero, one or more bytes of data have been changed

BCD AND ASCII APPLICATION PROGRAMS

Checksum Byte in ROM (cont')

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H.(a) Find the checksum byte, (b) perform the checksum operation to ensure data integrity, and (c) if the second byte 62H has been changed to 22H, show how checksum detects the error.

Solution:

(a) Find the checksum byte.

```
      25H
+     62H
+     3FH
+     52H
     ----
     118H
```

The checksum is calculated by first adding the bytes. The sum is 118H, and dropping the carry, we get 18H. The checksum byte is the 2's complement of 18H, which is E8H

(b) Perform the checksum operation to ensure data integrity.

```
      25H
+     62H
+     3FH
+     52H
+     E8H
     ----
     200H (dropping the carries)
```

Adding the series of bytes including the checksum byte must result in zero. This indicates that all the bytes are unchanged and no byte is corrupted.

(c) If the second byte 62H has been changed to 22H, show how checksum detects the error.

```
      25H
+     22H
+     3FH
+     52H
+     E8H
     ----
     1C0H (dropping the carry, we get C0H)
```

Adding the series of bytes including the checksum byte shows that the result is not zero, which indicates that one or more bytes have been corrupted.