LOGIC AND COMPARE INSTRUCTIONS

AND

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

```
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

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.

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:  ...
```

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

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

`CJNE destination,source,rel. addr.`

- The actions of comparing and jumping are combined into a single instruction called `CJNE` (compare and jump if not equal)
  - The `CJNE` instruction compares two operands, and jumps if they are not equal
  - The destination operand can be in the accumulator or in one of the Rn registers
  - The source operand can be in a register, in memory, or immediate
    - The operands themselves remain unchanged
  - It changes the CY flag to indicate if the destination operand is larger or smaller

LOGIC AND COMPARE INSTRUCTIONS

Compare Instruction (cont')

```
          CJNE R5,#80,NOT_EQUAL ;check R5 for 80
          ...              ;R5 = 80
NOT_EQUAL:
          JNC  NEXT        ;jump if R5 > 80
          ...              ;R5 < 80
NEXT:     ...
```

Compare	Carry Flag
destination ≥ source	CY = 0
destination < source	CY = 1

CY flag is always checked for cases of greater or less than, but only after it is determined that they are not equal

- Notice in the CJNE instruction that any Rn register can be compared with an immediate value
  - There is no need for register A to be involved

LOGIC AND COMPARE INSTRUCTIONS

Compare Instruction (cont')

- The compare instruction is really a subtraction, except that the operands remain unchanged
  - Flags are changed according to the execution of the SUBB instruction

Write a program to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following.

If T = 75 then A = 75
If T < 75 then R1 = T
If T > 75 then R2 = T

Solution:

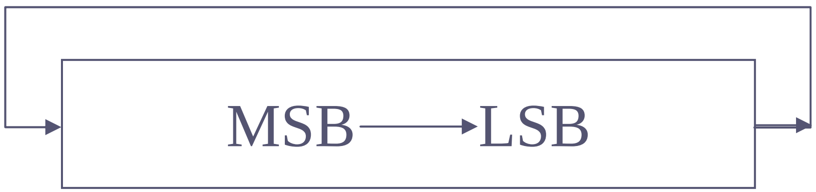
```
         MOV  P1,#0FFH   ;make P1 an input port
         MOV  A,P1       ;read P1 port
         CJNE A,#75,OVER ;jump if A is not 75
         SJMP EXIT       ;A=75, exit
OVER:    JNC  NEXT       ;if CY=0 then A>75
         MOV  R1,A       ;CY=1, A<75, save in R1
         SJMP EXIT       ; and exit
NEXT:    MOV  R2,A       ;A>75, save it in R2
EXIT:    ...
```

ROTATE INSTRUCTION AND DATA SERIALIZATION

Rotating Right and Left

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

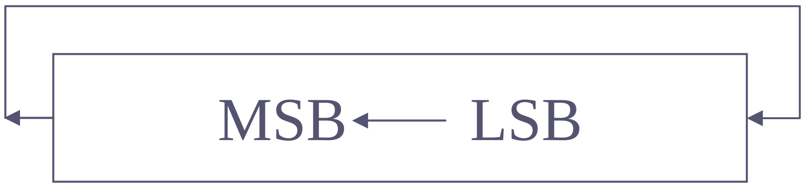
- In rotate right
  - The 8 bits of the accumulator are rotated right one bit, and
  - Bit D0 exits from the LSB and enters into MSB, D7




```
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
```

```
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
```

- 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

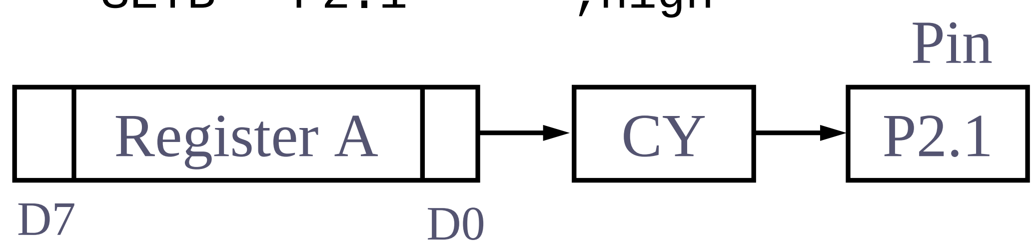
Department of Computer Science and Information Engineering
National Cheng Kung University, TAIWAN

- 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
```



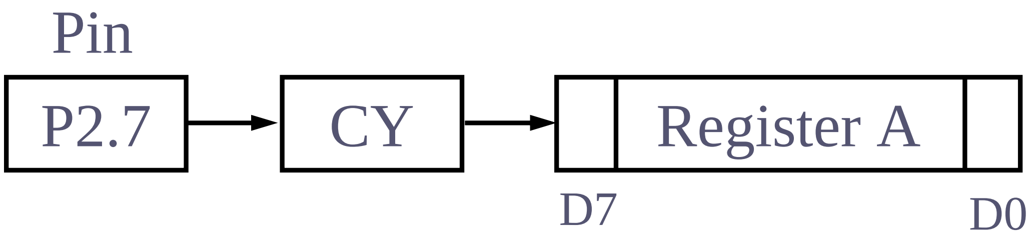
ROTATE INSTRUCTION AND DATA SERIALIZATION

Serializing Data (cont')

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
```



- 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
```

```
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

(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
```

ASCII code and BCD 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

- The DS5000T microcontrollers have a real-time clock (RTC)
  - The RTC provides the time of day (hour, minute, second) and the date (year, month, day) continuously, regardless of whether the power is on or off
- However this data is provided in packed BCD
  - To be displayed on an LCD or printed by the printer, it must be in ACSII format

Packed BCD		Unpacked BCD		ASCII
29H 0010 1001	→	02H & 09H 0000 0010 & 0000 1001	→	32H & 39H 0011 0010 & 0011 1001

ASCII to Packed BCD Conversion

- To convert ASCII to packed BCD
  - It is first converted to unpacked BCD (to get rid of the 3)
  - Combined to make packed BCD

key	ASCII	Unpacked BCD	Packed BCD
4	34	0000 0100	
7	37	0000 0111	0100 0111 or 47H

```
        MOV     A, #'4'    ;A=34H, hex for '4'
        MOV     R1,#'7'    ;R1=37H,hex for '7'
        ANL     A, #0FH    ;mask upper nibble (A=04)
        ANL     R1,#0FH    ;mask upper nibble (R1=07)
        SWAP    A          ;A=40H
        ORL     A, R1      ;A=47H, packed BCD
```

BCD AND ASCII APPLICATION PROGRAMS

ASCII to Packed BCD Conversion (cont')

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.

```
MOV    A,#29H   ;A=29H, packed BCD
MOV    R2,A     ;keep a copy of BCD data
ANL    A,#0FH   ;mask the upper nibble (A=09)
ORL    A,#30H   ;make it an ASCII, A=39H('9')
MOV    R6,A     ;save it
MOV    A,R2     ;A=29H, get the original
data
ANL    A,#0F0H  ;mask the lower nibble
RR     A        ;rotate right
RR     A        ;rotate right      SWAP A
RR     A        ;rotate right
RR     A        ;rotate right
ORL    A,#30H   ;A=32H, ASCII char. '2'
MOV    R2,A     ;save ASCII char in R2
```

BCD AND ASCII APPLICATION PROGRAMS

Using a Look-up Table for ASCII

Assume that the lower three bits of P1 are connected to three switches. Write a program to send the following ASCII characters to P2 based on the status of the switches.

000	'0'
001	'1'
010	'2'
011	'3'
100	'4'
101	'5'
110	'6'
111	'7'

Solution:

```
        MOV     DPTR,#MYTABLE
        MOV     A,P1         ;get SW status
        ANL     A,#07H       ;mask all but lower 3
        MOVC    A,@A+DPTR ;get data from table
        MOV     P2,A         ;display value
        SJMP    $            ;stay here
;------------------
        ORG     400H
MYTABLE DB      '0','1','2','3','4','5','6','7'
        END
```

- To ensure the integrity of the ROM contents, every system must perform the checksum calculation
  - The process of checksum will detect any corruption of the contents of ROM
  - The checksum process uses what is called a *checksum byte*
    - The checksum byte is an extra byte that is tagged to the end of series of bytes of data

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      The checksum is calculated by first adding the
+    62H      bytes. The sum is 118H, and dropping the carry,
+    3FH      we get 18H. The checksum byte is the 2's
+    52H      complement of 18H, which is E8H
    ----
    118H
```

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

```
     25H
+    62H      Adding the series of bytes including the checksum
+    3FH      byte must result in zero. This indicates that all the
+    52H      bytes are unchanged and no byte is corrupted.
+    E8H
    ----
    200H (dropping the carries)
```

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

```
     25H
+    22H      Adding the series of bytes including the checksum
+    3FH      byte shows that the result is not zero, which indicates
+    52H      that one or more bytes have been corrupted.
+    E8H
    ----
    1C0H (dropping the carry, we get C0H)
```

- Many ADC (analog-to-digital converter) chips provide output data in binary (hex)
  - To display the data on an LCD or PC screen, we need to convert it to ASCII
    - Convert 8-bit binary (hex) data to decimal digits, 000 – 255
    - Convert the decimal digits to ASCII digits, 30H – 39H