

LESSON 4: LOGICAL INSTRUCTIONS

Objectives:

1. Study the operation of logical instructions.

- ANL A,<byte>
- ANL <byte>, A
- ANL <byte>, #data
- ORL A,<byte>
- ORL <byte>, A
- ORL <byte>, #data
- XRL A,<byte>
- XRL <byte>, A
- XRL <byte>, #data
- CLR A
- CPL A
- RL A
- RLC A
- RR A
- RRC A
- SWAP A

2. Use of monitor function call.

The logical instructions, ANL, ORL and XRL allow destination to be the Accumulator or direct byte. All of them perform logical function using bitwise between source and destination.

ANL A,<byte>
ANL <byte>, A
ANL <byte>, #data

Examples

1. Logically AND the Accumulator with memory location 35H. The result is placed in the Accumulator.

ANL A, 35H

The direct byte can be any locations of the on chip RAM, so it can be R0-R7 or any special function registers, e.g. P1, P2, P3.

2. Logically AND the memory location 35H with the Accumulator. The result is placed in memory location 35H.

ANL 35H, A

3. Logically AND the memory location 35H with 8-bit immediate value. The result is placed in memory location 35H.

ANL 35H,#00000001B

We can use logical AND for bit test. Suppose we want to know the specified bit in the Accumulator that was '0' or '1'. We can have MASK byte having bit position to be tested is '1'.

We have seen the example of sending ASCII letter to terminal using COUT subroutine.

```
COUT:    JNB TI,$
         CLR TI
         MOV SBUF,A
         RET
```

The subroutine checks TI bit whether one or zero? The TI bit is Transmit Interrupt flag. It will be '1' when hardware buffer is empty and ready to accept new data. The TI bit is bit 1 in SCON register.

MSB						LSB	
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SCON: Serial Control Register

Now here is the new version of COUT using bit test.

```
MASK1:   EQU 00000010B
MASK2:   EQU 11111101B

MY_COUT: MOV A,SCON
         ANL A, #MASK1    ; test bit 1 by ANDing with 1
         JZ  MY_COUT     ; jump back if TI=0
         ANL SCON,#MASK2 ; clear TI bit by ANDing with 0
         MOV SBUF,B      ; write B to SCON
         RET
```

We see that MASK1 has bit 1 is logic '1', the rest are '0'. The result in A after AND will affect to JZ instruction. If A equal 0, then the CPU will jump back to MY_COUT. If A is not equal 0, then CPU will continue next instruction.

MASK2 is used to CLEAR a specified bit without disturbs to the rest. The bit to be cleared in the mask byte must be '0'. We see that the MASK2 is 1111 1101B.

EXERCISE 4-1:

1. Write a program that uses MY_COUT to print "*" 128 times.
2. Write a program that clear bit 0 and bit 7 of external RAM from address 9000H-9100H. (use logical AND with MASK byte having bit position to be cleared is '0')

The method of using logical AND instruction for bit testing (with '1') and for clearing bit (with '0') is generic tool.

We can apply it for high-level language programming. See example below.

```
#define MASK 0x04 // P3.2 bit position
```

```
main()  
{  
    while(1)  
    {  
        while((P3&MASK) != 0)  
        P1 = 0x80; // turn LED off  
        P1 = ~0x80; // turn LED on  
    }  
}
```

The example was written in c language. The program tests P3.2 by ANDing P3 and 0x04. If P3.2 is '1' then port P1 will be written with 0x80. If P3.2 is '0' then P1 will be ANDed with complement of 0x80, making LED on.

ORL A,<byte>
ORL <byte>, A
ORL <byte>, #data

ORL instruction performs logical-OR operation between source and destination. The result is stored in destination byte.

Examples

1. Logically OR the Accumulator with memory location 40H.

```
ORL A, 40H
```

2. Logically OR the memory location 40H with Accumulator.

```
ORL 40H, A
```

3. Logically OR the memory location 40H with 8-bit immediate value.

```
ORL 40H, #00000001B
```

We can use ORL instruction to make a specified bit to be '1' by Oring it with MASK byte.

To set bit 0 of external RAM address 9000H to '1',

```
MASK3:    EQU 00000001B

          MOV DPTR,#9000H
          MOVX A,@DPTR
          ORL A,#MASK3
          MOVX @DPTR,A
```

EXERCISE 4-2:

1. Write a program that sets low nibble of external memory from 9000H to 9100H.

```
XRL A,<byte>
XRL <byte>, A
XRL <byte>, #data
```

XRL instruction performs logical Exclusive OR between source and destination. The result is stored in destination byte.

Examples

4. Logically EX-OR the Accumulator with memory location 40H.

```
XRL A, 40H
```

5. Logically EX-OR the memory location 40H with Accumulator.

```
XRL 40H, A
```

6. Logically EX-OR the memory location 40H with 8-bit immediate value.

```
XRL 40H, #00000001B
```

We can use XRL to toggle specified bit with logic '1'. See example below,

```
MASK:    EQU 80H    ; for P1.7 toggle.
```

```
MOV P1, 80H
TOGGLE: XRL P1,#MASK
        JMP TOGGLE
```

Another useful of using Exclusive OR is data encryption and decryption.

EXERCISE 4-3:

1. Test program below with command jump. What is the code that prints on screen?

```
$mod51
#include(mypaulm2.equ)

        cseg at 8000h
        jmp main

        org 8100h

main:   mov r7,#8
        mov dptr,#mycode

loop:  movx a,@dptr
        call cout
        inc dptr
        djnz r7,loop
        jmp monitor

mycode: db '01234569' ; 8-byte code

        end
```

2. We will encrypt the code with our key. Suppose we have key byte equals 99H.

```
$mod51
#include(mypaulm2.equ)

        cseg at 8000h
        jmp main

        org 8100h

main:   mov r7,#8
        mov dptr,#mycode

loop:  movx a,@dptr
```

```
xrl a,#99h  
call cout  
inc dptr  
djnz r7,loop  
jmp monitor
```

mycode: db '01234569' ; 8-byte code

end

Before sending byte to terminal the program EX-OR it with 99H.

3. What is the code that prints on screen now? Does it the same as previous one?

Now let check what is the byte stream after encryption with 99H?

```
$mod51  
$include(mypaulm2.equ)
```

```
cseg at 8000h  
jmp main
```

```
org 8100h
```

```
main: mov r7,#8  
mov dptr,#mycode
```

```
loop: movx a,@dptr  
xrl a,#99h  
call phex  
mov a,#' '  
call cout  
inc dptr  
djnz r7,loop  
jmp monitor
```

mycode: db '01234569' ; 8-byte code

end

4. What is the HEX code that prints on screen now?
5. You can change the key to any values, let's try and see what will happen?

Here is the answer with 99H key, A9 A8 AB AA AD AC AF A0.

Now our code was encrypted with our key 99H. To decrypt it, we can use our key 99H EX-OR it.

6. Write a program that decrypts the code A9 A8 AB AA AD AC AF A0. Remember our key is 99H. Print the original code back on screen.

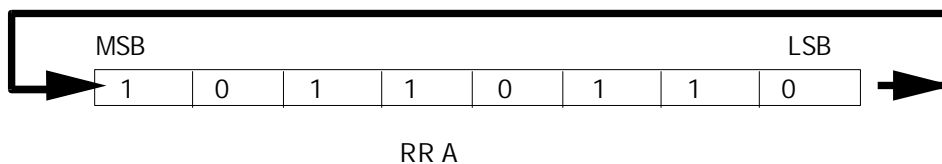
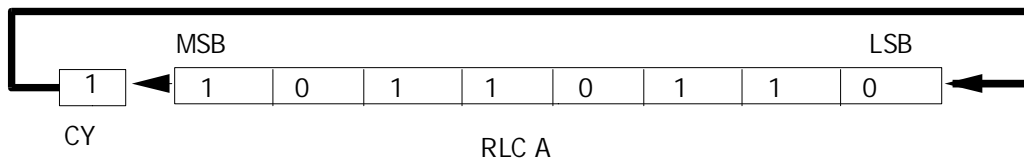
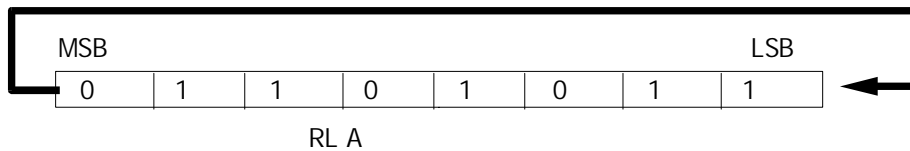
To make it complicate, the key can then be a stream of byte sequence.

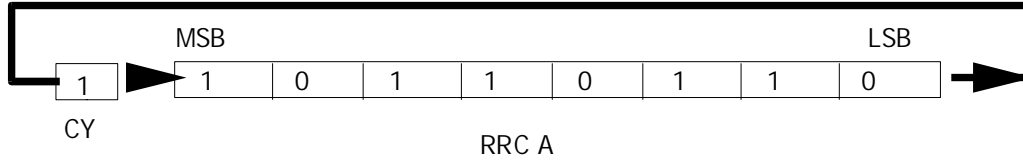
CLR A
CPL A

CLR A clears the Accumulator, all bits set to 0.
CPL A logically complements each bit of the Accumulator.

RL A
RLC A
RR A
RRC A

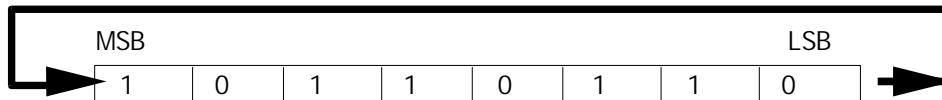
RL A rotates eight bits in the Accumulator one bit to left. RLC A rotates eight bits and carry flag one bit to left. Similarly for RR A and RRC A, but it rotates one bit to right. See below.





SWAP A

SWAP A interchanges the low and high order nibbles. The same as rotation operation with 4-bit rotate.



SWAP A (4-bit rotation)

EXERCISE 4-4:

1. Edit and test the program with single step, answer the following questions.

```

cseg at 8000h
    jmp main

    org 8100h

main: mov a,#0AAh
      rl a          ; A = ?
      clr c        ; carry flag = ?
      rlc a       ; A = ?

      rr a        ; A = ?
      setb c     ; carry = ?
      rrc a      ; A = ?
      swap a     ; A = ?
      end

```

2. Write a program that swaps nibble of external memory from address 9000H-9100H.