LESSON 5: BOOLEAN AND JUMP INSTRUCTIONS

Objectives:

1. Study the operation of Boolean and jump instructions.

- ANL C,bit
- ANL C,/bit
- ORL C,bit
- ORL C,/bit
- MOV C,bit
- MOV bit,C
- CLR C
- SETB C
- CPL C
- CPL bit
- JC rel
- JNC rel
- JB bit,rel
- JNB bit,rel
- JBC bit,rel
- JZ rel
- JNZ rel
- DJNZ <byte>, rel
- CJNE A,<byte>,rel
- CJNE <byte>,#data,rel
- JMP addr
- JMP @A+DPTR
- CALL addr
- RET
- RETI

Boolean instructions perform bit operations between source and destination bits. The Carry flag is mainly used to be source and destination bits. All of bit addressable locations can be used with Boolean operations. The onchip RAM space from byte address 20H-2FH is reserved for general purpose bit variable storage. Most of special function registers from 80H-FFH also are bit addressable.

ANL C,bit
ANL C,/bit

Logical AND carry flag with bit variable, the result is stored in carry flag, the bit variable is unchanged. A slash (/) preceding the bit variable COMPLEMENTS the bit before AND with carry flag. The source bit is not changed.

Examples

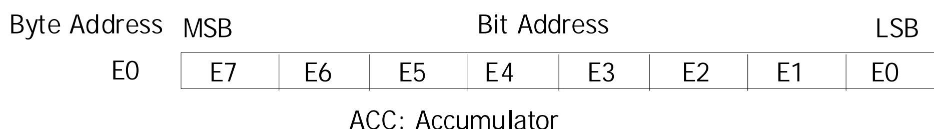
1. Logically AND Carry flag with bit 0 of the Accumulator. The result is placed in the Carry flag.

ANL C, ACC.0

2. Logically AND Carry flag with complement of bit 0 of the Accumulator. The result is placed in the Carry flag.

ANL C,/ACC.0

The Accumulator has bit address as shown below. Bit address ACC.0 is then E0H and ACC.7 is E7H.



Byte Address	MSB			Bit Address				LSB
E0	E7	E6	E5	E4	E3	E2	E1	E0

ACC: Accumulator

ORL C,bit
ORL C,/bit

Logical OR carry flag with bit variable, the result is stored in carry flag, the bit variable is unchanged. A slash (/) preceding the bit variable COMPLEMENTS the bit before AND with carry flag. The source bit is not changed.

Examples

1. Logically OR Carry flag with bit 7 of the Accumulator. The result is placed in the Carry flag.

ORL C, ACC.0

2. Logically OR Carry flag with complement of bit 0 of the Accumulator. The result is placed in the Carry flag.

ORL C,/ACC.0

MOV C,bit
MOV bit,C

We can copy content of bit variable to Carry flag and copy the content of Carry flag to bit variable as well.

Examples

1. Copy bit 0 of the Accumulator to Carry flag.

    MOV C, ACC.0

2. Copy Carry flag to ACC.7.

    MOV ACC.7,C

EXERCISE 5-1:

1. Test a program below with single step, answer the questions.

```
   org 8100h

main: mov c,2fh.0   ; C = ?
      mov acc.0,c   ; ACC.0 = ?

      mov c,2fh.1   ; C = ?
      anl c,acc.1   ; C = ?
      mov 2fh.1,c   ; Bit address 2fh.1 = ?

      mov c,acc.2   ; C = ?
      orl c,2fh.2   ; C = ?
      mov acc.2,c   ; ACC.2 = ?

      mov c,127 ; C = ?
      anl c,127 ; C = ?
      mov acc.7,c   ; ACC.7 = ?
```

CLR C
SETB C
CPL C
CPL bit

CLR C clears Carry flag.
SETB C sets Carry flag to ‘1’.
CPL C complements Carry flag.
CPL Bit complements bit variables.

Examples

1. Complement P1.7.

    CPL P1.7

JC rel
JNC rel

JC is conditionally jump instruction with Carry flag checking. If carry flag set, the current program counter will be added with relative byte or offset byte.
Examples

1. The program below uses Label "c_is_set" as the destination loaction.

```
     org 8100h

main: mov c,P3.2
     jc c_is_set
     sjmp main

c_is_set:

     cpl c
     sjmp main
     end
```

Let us see the list file what is the relative byte with JC instruction.

```
8100                67           org 8100h
                    68
8100 A2B2           69    main:  mov c,P3.2
8102 4002           70           jc c_is_set
8104 80FA           71           sjmp main
8106                72    c_is_set:
                    73
8106 B3             74           cpl c
8107 80F7           75           sjmp main
                    76           end
```

The machine code of JC is 40H and the relative byte or offset byte is **02**.
The label "c_is_set" is loacted at address 8016. When CPU fetches machine code of JC instruction, 40 and 02, the Program Counter will be 8104. If carry flag is set, the jump condition is then true, the Program Counter 8104 will be added with offset byte 02. The result in Program Counter will be 8106, where is the lable "c_is_set".

Similarly for short JUMP, sjmp main, but in backward direction, the relative byte is FA.

EXERCISE 5-2:

1. Show how the offset byte FA is used to compute the destination location, F000.
2. The same problem but now for the byte F7.

The offset or relative bytes actually are computed by the Assembler. Programmer just entered the label for the destination location.

3. Single step and press S1 button to make P3.2 low and learn how JC forces CPU jump to the specific location with flag condition.

JB bit,rel
JNB bit,rel
JBC bit,rel

Above instructions can be used with any bit addressable locations. JB jumps if bit is ‘1’. JNB jumps if bit is ‘0’. JBC jumps if bit is ‘1’ and clear it before jump has made.

JZ rel
JNZ rel
DJNZ <byte>, rel
CJNE A,<byte>,rel
CJNE <byte>,#data,rel

JZ jumps if Accumulator is ZERO. JNZ jumps if Accumulator is NOT ZERO.

DJNZ decrements byte, R0-R7 or direct byte and test it. If not ZERO then jump to address computed by Program Counter + Relative byte.

CJNE A compares Accumulator with direct byte, jumps if A is not equal the content of direct byte. The destination location was computed by Program Counter + Relative byte.

CJNE byte,#data compares direct byte with 8-bit constant. If not equal, jump to destination.

Examples

Now suppose we want to produce time base using hardware timer 0. We will use 16-bit timer to count the clock signal when timer is overflow, the timer flag TF0 will set. We can use it for code running with time.

EXERCISE 5-3:

1. Assemble the code below, download to the board and run with command jump.

```
$mod51

led    equ P1.7

     dseg at 50h

tick:  ds 1    ; tick is variable for counting number of
```

```
; timer flag when set


      cseg at 8000h
      jmp main

      org 8100h

main: orl tmod,#1    ; set timer0 to mode1
       setb tr0      ; run timer0


wait: jnb tf0,$      ; jump if TF0 is zero
      clr tf0        ; clear TF0

      orl th0,#0dch  ; reload timer 0 with 0DC00H
      inc tick       ; increment tick
      mov a,tick
      cjne a,#50,next1 ; test if tick = 50

      mov tick,#0    ; clear tick
      cpl led        ; task1 is Complement LED

next1:
      jmp wait       ; jump back to wait TF0 again

         end
```

2. What happen to the onboard LED?
3. Draw the logic level appeared at P1.7. What is period of logic '1' and logic '0'?

Timer0 is 16-bit counter. When enabled, the counter will increment by one every clock input. Timer flag0 or TF0 will set when timer0 is overflow. The 8051SBC's oscillator is 11.0592MHz clock. The input clock that supplies to timer0 counter is 11.0592MHz/12 = 921600Hz. So if we have timer0 with 0000, the overflow will occur at 65536th clock. Or the TF0 will set at 14Hz rate. So if the timer0 has been loaded with predefined initial value, we can produce the desired rate easily.

Suppose we want to have TF0 set every 1/100Hz, or 10ms. So the number of clock that needed to be loaded will be 921600Hz/100Hz = 9216 clocks.

Since counter0 is increment counting, so we can find the value to be loaded equal 65536-9216 = 56320 or 0DC00H.

We see that after TF0 was set, the counter0 will be 0000. We use ORL TH0 with 0DCH, it means the counter is reloaded with 0DC00H.

We will use 10ms time-base for our next program.

JMP addr
JMP @A+DPTR
CALL addr
RET
RETI

JMP unconditionally jumps to destination address. There are two types: SJMP and LJMP. SJMP uses offset byte to compute the destination address. The range is –128 to +127 relative to the current program counter. LJMP uses absolute address for the destination. With the ASM51, the assembler will produce appropriate JMP automatically.

JMP @A+DPTR, the destination address will be computed by adding content of Accumulator and DPTR. We ca use this JMP for JUMP table

CALL, is instruction that calls subroutine at specified address. It uses STACK memory for saving the return address. When call to subroutine, the subroutine must have RET instruction to get back to main program. 8051 provides two types: ACALL and LCALL. ACALL needs machine code only two bytes, but it can jump only within 2kB space. LCALL is absolute call. The machine code is three bytes.

RET is return to main program. The operation of RET, will retrieve two byte TOP of STACK to Program Counter. When used with CALL, it will return to instruction next to CALL instruction. RETI is the same except it also restores the interrupt logic to accept additional interrupts.

EXERCISE 5-4:

1. Assemble the code below, download to the board and run with command jump.

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

led     equ P1.7

    dseg at 50h

tick:  ds 1
sec100: ds 1
sec:    ds 1
min:    ds 1
hour:   ds 1


        cseg at 8000h
    jmp main
```

```
    org 8100h

;---- initialization code ----------------
main: orl tcon,#1; set timer0 to mode1
    setb tr0

wait: jnb tf0,$

    clr tf0
    orl th0,#0dch

;------ tasks that run every 10ms -------

    call blink_led
    call update_clock

    jmp wait

;*********** subroutines *****************
; blink LED every second

blink_led: inc tick
       mov a,tick
       cjne a,#50,exit1
       mov tick,#0
       cpl led
exit1:
       ret

update_clock:

    inc sec100
    mov a,sec100
    cjne a,#100,exit2
    mov sec100,#0

    mov a,sec
    add a,#1
    da a
    mov sec,a
    call print_sec
```

```
exit2:
    ret

print_sec: mov a,sec
    call phex
    call newline
    ret
    end
```

2. What happen on screen?
3. Modify the code to print TIME on screen, HH:MM:SS, e.g. 17:59:55

We can add many tasks that run every 10ms to the main loop easily.

Here is the basic timing of above program. The Timer0 produces 10ms with TF0 indicator. We split large program into small subroutines that execute every 10ms. Each task will occupy a very short period of time and run in round-robin manner. We will learn more programs that implement for real-time applications.