



Jump, Loop, and Call Instructions

Chapter Objectives

- Code 8051 Assembly language instructions using loops
- Code 8051 Assembly language conditional jump instructions
- Explain conditions that determine each conditional jump instructions
- Code long & short instructions for unconditional long and short jumps
- Calculate target addresses for jump instructions
- Code 8051 subroutines



Jump, Loop, and Call Instructions

Chapter Objectives...

- Describe precautions in using the stack in subroutines
- Discuss crystal frequency versus machine cycle
- Code 8051 programs to generate a time delay



Jump, Loop, and Call Instructions

Looping in the 8051

- Repeating a sequence of instructions a certain number of times is called a “*loop*”.
- The loop is one of the most widely used actions that any microprocessor or microcontroller performs.
- In the 8051, the loop action is performed by the instruction “DJNZ” reg label.
- The register is decremented; if is not zero, it jumps to the target address referred to by the label.
- Prior to the start of the loop the register is loaded with the counter for the number of repetitions.



Jump, Loop, and Call Instructions

Looping in the 8051...

Example 3-1.

(Functional Requirements)

Write a program to

a) Clear ACC, then

b) Add 3 to the accumulator ten times



Jump, Loop, and Call Instructions

Looping in the 8051...

Example 3-1(Solution).

;This program adds value 3 to the ACC ten times

```
MOV A, #0          ; A=0, clear ACC
```

```
MOV R2, #10       ; load counter R2 =10
```

```
AGAIN: ADD A, #03  ; add 03 to ACC
```

```
DJNZ R2, AGAIN    ;repeat until R2=0 (10 times)
```

```
MOV R5, A         ;save A in R5
```



Jump, Loop, and Call Instructions

Looping inside a loop

- What happens if we want to repeat a process several times, say 256?
- To accomplish this a loop inside a loop allows code repetition.
- Loop encapsulated (inside) another loop is called a “***nested***” loop

See Example 3-2.



Jump, Loop, and Call Instructions

Looping inside a loop

Example 3-1.

(Functional Requirements)

Write a program to

- a) Load the accumulator with the value #55
- b) Complement the ACC 700 times



Jump, Loop, and Call Instructions

Looping inside a loop

Example 3-2(Solution).

Since 700 is larger than 255(the maximum capacity of any register) 2 registers shall be used to hold the count.

```
MOV A, #55H      ; A=55H
```

```
MOV R3, #10     ; R3 =10; the outer loop count
```

```
NEXT: MOV R2, #70 ; R2 = 70, the inner loop
```

```
AGAIN: CPL A    ; complement A register
```

```
DJNZ R2, AGAIN ; repeat it 70 times (inner loop)
```

```
DJNZ R3, NEXT
```



Jump, Loop, and Call Instructions

Other conditional Jumps

Some instructions, such as **JZ (jump if A = zero)** and **JC (jump if carry)**, jump only if a certain condition is met.

Example of JZ

```
MOV  A, R0    ; A=R0
JZ   OVER     ; jump if A= 0
MOV  A, R1    ; A=R1
JZ   OVER
```

OVER:

Note: JZ instruction can only be used with the register A.

See **Table 3-1 (slide 15)** for additional jump instructions



Jump, Loop, and Call Instructions

Other conditional Jumps...

Example JNZ (jump if not zero)

(Functional Requirements)

- a) Write a program to determine if R5 contains the value 0.
- b) If so, put 55H in it



Jump, Loop, and Call Instructions

Other conditional Jumps...

Example JNZ (Solution)

```
MOV  A, R5      ; copy R5 to A
JNZ  NEXT      ; jump if A is not zero
MOV  R5, #55H
```

```
NEXT:  ...
```



Jump, Loop, and Call Instructions

Other conditional Jumps...

Example of JNC(jump if no carry, jumps if $CY = 0$)

(Functional Requirements)

- a) Find the sum of values 79H, F5H, and E2H.
- b) Put the sum in registers R0 (low byte) and R5 (high byte)



Jump, Loop, and Call Instructions

Other conditional Jumps...

Example JNC Solution

```
MOV A, #0           ; clear A (A=0)
MOV R5, A           ; clear R5
ADD A, #79H         ; A =0+79H =79H
JNC N_1             ; if no carry, add next number
INC R5              ; If CY=1, increment R5
N_1: ADD A, #0F5H    ; A =79+F5=6E and CY=1
      JNC N_2        ; jump if CY =0
      INC R5         ; If CY =1 then increment R5 (R5=1)
N_2: ADD A, #0E2H    ; A=6E+E2=50 and CY=1
      JNC OVER       ; jump if CY=0
      INC R5         ; If CY =1, increment 5
OVER: MOV R0, A      ; Now R0 =50H, and R5=02
```



Jump, Loop, and Call Instructions

Other conditional Jumps...

Table 3-1: 8051 Conditional Jump Instructions

Instruction	Action
JZ	Jump if A = 0
JNZ	Jump if A \neq 0
DJNZ	Decrement and jump if register \neq 0
CJNE A, data	Jump if A \neq data
CJNE reg, #A	Jump if byte \neq #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if Bit = 0
JBC	Jump if bit = 1 and clear bit



Jump, Loop, and Call Instructions

All conditional jumps are short jumps

FYI :

All conditional jumps are short jumps, meaning that the address of the target must be within -128 to +127 bytes of the contents of the program counter.



Jump, Loop, and Call Instructions

Unconditional Jump Instructions

- The unconditional jump is a jump in which control is transferred to the target without any preset requirements to the target location.
- In the 8051uC there are 2 unconditional jumps:
 - a) LJMP (long jump)
 - b) SJMP (short jump)
- LJMP is an unconditional long jump.
 - a) It is a 3byte instruction in which the 1st by is the opcode and the 2nd 3rd bytes represent the 16 bit address location of the target location.
 - b) The 2-byte target address allows a jump to any memory location from 0000H to FFFFH



Jump, Loop, and Call Instructions

Unconditional Jump Instructions...

- SJMP is a 2byte instruction,
 - a) The 1st byte is the opcode and the 2nd byte is the relative address of the target location.
 - b) The relative address range of 00 –FFH
 - c) Divided into the forward and backward jumps: (within -128 to +127 bytes of memory relative to the address of the current PC [program counter]).
- If the jump is forward, the target address can be within the a space of 127 bytes from the current PC.
- If the target address is backward, the target address can be within -128 bytes from the current PC.