

# JUMP, LOOP AND CALL INSTRUCTIONS

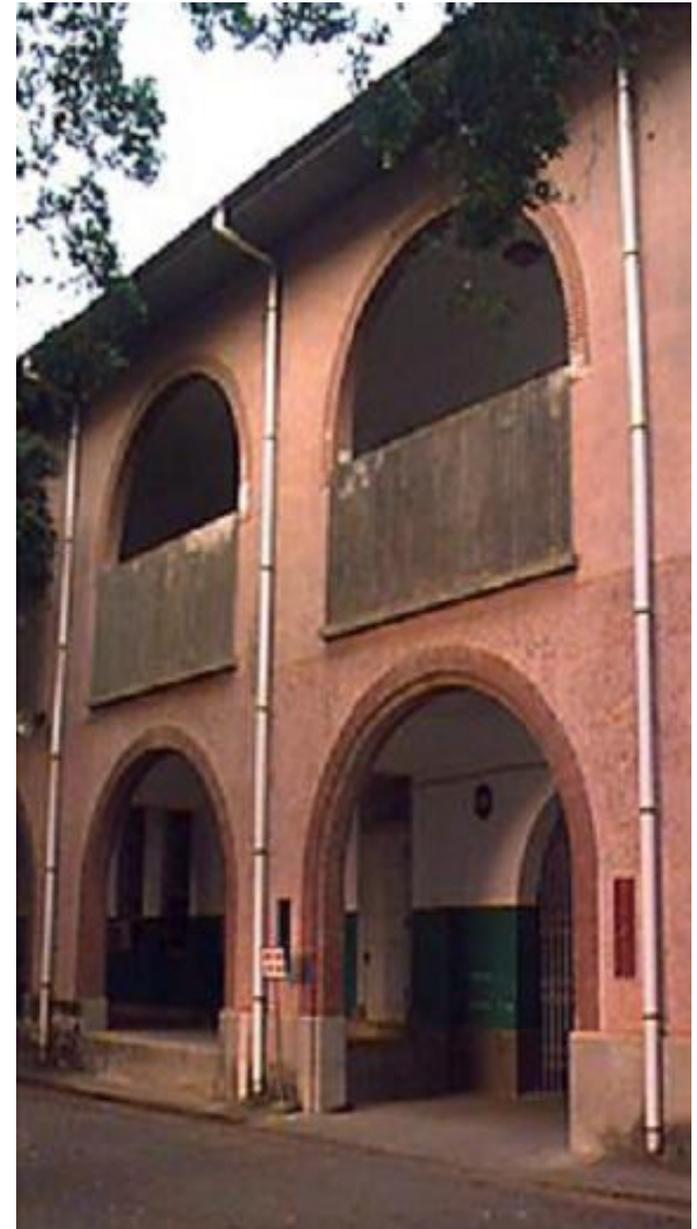
---

*The 8051 Microcontroller and Embedded Systems: Using Assembly and C*  
Mazidi, Mazidi and McKinlay

Chung-Ping Young  
楊中平

*Home Automation, Networking, and Entertainment Lab*

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



# LOOP AND JUMP INSTRUCTIONS

## Looping

A loop can be repeated a maximum of 255 times, if R2 is FFH

□ Repeating a sequence of instructions a certain number of times is called a *loop*

➤ Loop action is performed by  
DJNZ reg, Label

- The register is decremented
- If it is not zero, it jumps to the target address referred to by the label
- Prior to the start of loop the register is loaded with the counter for the number of repetitions
- Counter can be R0 – R7 or RAM location

```
;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
```



# LOOP AND JUMP INSTRUCTIONS

## Nested Loop

- If we want to repeat an action more times than 256, we use a loop inside a loop, which is called *nested loop*
  - We use multiple registers to hold the count

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times

```
MOV    A, #55H    ;A=55H
MOV    R3, #10    ;R3=10, outer loop count
NEXT:  MOV    R2, #70 ;R2=70, inner loop count
AGAIN: CPL    A      ;complement A register
        DJNZ  R2, AGAIN ;repeat it 70 times
        DJNZ  R3, NEXT
```



# LOOP AND JUMP INSTRUCTIONS

## Conditional Jumps

- Jump only if a certain condition is met

**JZ label ; jump if A=0**

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

Can be used only for register A,  
not any other register

Determine if R5 contains the value 0. If so, put 55H in it.

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



# LOOP AND JUMP INSTRUCTIONS

## Conditional Jumps (cont')

□ (cont')

**JNC label ;jump if no carry, CY=0**

- If CY = 0, the CPU starts to fetch and execute instruction from the address of the label
- If CY = 1, it will not jump but will execute the next instruction below JNC

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

```
MOV A,#0 ;A=0
MOV R5,A ;clear R5
ADD A,#79H ;A=0+79H=79H
; JNC N_1 ;if CY=0, 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,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
```

MOV R5,#0



# LOOP AND JUMP INSTRUCTIONS

## Conditional Jumps (cont')

### 8051 conditional jump instructions

Instructions	Actions
JZ	Jump if A = 0
JNZ	Jump if A $\neq$ 0
DJNZ	Decrement and Jump if A $\neq$ 0
CJNE A,byte	Jump if A $\neq$ byte
CJNE reg,#data	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

- All conditional jumps are short jumps
  - The address of the target must within -128 to +127 bytes of the contents of PC



# LOOP AND JUMP INSTRUCTIONS

## Unconditional Jumps

- ❑ The unconditional jump is a jump in which control is transferred unconditionally to the target location

### **LJMP** (long jump)

- 3-byte instruction
  - First byte is the opcode
  - Second and third bytes represent the 16-bit target address
    - Any memory location from 0000 to FFFFH

### **SJMP** (short jump)

- 2-byte instruction
  - First byte is the opcode
  - Second byte is the relative target address
    - 00 to FFH (forward +127 and backward -128 bytes from the current PC)



## LOOP AND JUMP INSTRUCTIONS

### Calculating Short Jump Address

- ❑ To calculate the target address of a short jump (`SJMP`, `JNC`, `JZ`, `DJNZ`, etc.)
  - The second byte is added to the PC of the instruction immediately below the jump
- ❑ If the target address is more than -128 to +127 bytes from the address below the short jump instruction
  - The assembler will generate an error stating the jump is out of range



# LOOP AND JUMP INSTRUCTIONS

## Calculating Short Jump Address (cont')

Line	PC	Opcode	Mnemonic	Operand
01	0000		ORG	0000
02	0000	7800	MOV	R0, #0
03	0002	7455	MOV	A, #55H
04	0004	6003	JZ	NEXT
05	0006	08	INC	R0
06	0007	04	INC	A
07	0008	04	INC	A
08	0009	2477	ADD	A, #77H
09	000B	5005	JNC	OVER
10	000D	E4	CLR	A
11	000E	F8	MOV	R0, A
12	000F	F9	MOV	R1, A
13	0010	FA	MOV	R2, A
14	0011	FB	MOV	R3, A
15	0012	2B	ADD	A, R3
16	0013	50F2	JNC	AGAIN
17	0015	80FE	SJMP	HERE
18	0017		END	



## CALL INSTRUCTIONS

- ❑ Call instruction is used to call subroutine
  - Subroutines are often used to perform tasks that need to be performed frequently
  - This makes a program more structured in addition to saving memory space

### **LCALL** (long call)

- 3-byte instruction
  - First byte is the opcode
  - Second and third bytes are used for address of target subroutine
    - Subroutine is located anywhere within 64K byte address space

### **ACALL** (absolute call)

- 2-byte instruction
  - 11 bits are used for address within 2K-byte range



CALL  
INSTRUCTIONS

LCALL

- ❑ When a subroutine is called, control is transferred to that subroutine, the processor
  - Saves on the stack the the address of the instruction immediately below the LCALL
  - Begins to fetch instructions form the new location
- ❑ After finishing execution of the subroutine
  - The instruction RET transfers control back to the caller
    - Every subroutine needs RET as the last instruction



# CALL INSTRUCTIONS

## LCALL (cont')

```
ORG 0
BACK: MOV A,#55H ;load A with 55H
      MOV P1,A ;send 55H to port 1
      LCALL DELAY ;time delay
      MOV A,#0AAH ;load A with AA (in hex)
      MOV P1,A ;send AAH to port 1
      LCALL DELAY
      SJMP BACK ;keep doing this indefinitely
```

The counter R5 is set to FFH; so loop is repeated 255 times.

Upon executing "LCALL DELAY", the address of instruction below it, "MOV A,#0AAH" is pushed onto stack, and the 8051 starts to execute at 300H.

```
;----- this is delay subroutine -----
ORG 300H ;put DELAY at address 300H
DELAY: MOV R5,#0FFH ;R5=255 (FF in hex), counter
AGAIN: DJNZ R5,AGAIN ;stay here until R5 become 0
RET ;return to caller (when R5 =0)
END
```

The amount of time delay depends on the frequency of the 8051

When R5 becomes 0, control falls to the RET which pops the address from the stack into the PC and resumes executing the instructions after the CALL.



# CALL INSTRUCTIONS

## CALL Instruction and Stack

```
001 0000                ORG 0
002 0000 7455   BACK:  MOV  A,#55H   ;load A with 55H
003 0002 F590                MOV  P1,A     ;send 55H to p1
004 0004 120300             LCALL DELAY  ;time delay
005 0007 74AA                MOV  A,#0AAH ;load A with AAH
006 0009 F590                MOV  P1,A     ;send AAH to p1
007 000B 120300             LCALL DELAY
008 000E 80F0                SJMP  BACK   ;keep doing this
009 0010
010 0010 ;-----this is the delay subroutine-----
011 0300                ORG 300H
012 0300                DELAY:
013 0300 7DFE                MOV  R5,#0FFH ;R5=255
014 0302 DDFE   AGAIN:  DJNZ R5,AGAIN ;stay here
015 0304 22                RET          ;return to caller
016 0305                END          ;end of asm file
```

### Stack frame after the first LCALL

0A	
09	00
08	07

SP = 09

Low byte goes first  
and high byte is last



# CALL INSTRUCTIONS

Use PUSH/POP in Subroutine

Normally, the number of PUSH and POP instructions must always match in any called subroutine

```

01 0000                                ORG 0
02 0000 7455    BACK:  MOV  A,#55H    ;load A with 55H
03 0002 F590                                MOV  P1,A        ;send 55H to p1
04 0004 7C99                                MOV  R4,#99H
05 0006 7D67                                MOV  R5,#67H
06 0008 120300    LCALL DELAY    ;time delay
07 000B 74AA                                MOV  A,#0AAH    ;load A with AA
08 000D F590                                MOV  P1,A        ;send AAH to p1
09 000F 120300    LCALL DELAY
10 0012 80EC                                SJMP  BACK      ;keeping doing
                    this
11 0014 ;-----this is the delay subroutine-----
12 0300                                ORG 300H
13 0300 C004    DELAY:  PUSH 4          ;push R4
14 0302 C005                                PUSH 5          ;push R5
0304 7CFF                                MOV  R4,#0FFH ;R4=FFH
0306 7DFF    NEXT:  MOV  R5,#0FFH ;R5=FFH
0308 DDFE    AGAIN: DJNZ  R5,AGAIN
030A DCFA                                DJNZ  R4,NEXT
030C D005                                POP  5          ;POP into R5
030E D004                                POP  4          ;POP into R4
0310 0000                                ORG 0
22 0310                                ORG 0

```

After first LCALL			After PUSH 4			After PUSH 5		
0B			0B			0B	67	R5
0A			0A	99	R4	0A	99	R4
09	00	PCH	09	00	PCH	09	00	PCH
08	0B	PCL	08	0B	PCL	08	0B	PCL



# CALL INSTRUCTIONS

## Calling Subroutines

```
;MAIN program calling subroutines
      ORG 0
MAIN:  LCALL      SUBR_1
      LCALL      SUBR_2
      LCALL      SUBR_3

      SJMP      HERE
;-----end of MAIN

SUBR_1: ...
      ...
      RET
;-----end of subroutine1

SUBR_2: ...
      ...
      RET
;-----end of subroutine2

SUBR_3: ...
      ...
      RET
;-----end of subroutine3
      END                ;end of the asm file
```

It is common to have one main program and many subroutines that are called from the main program

This allows you to make each subroutine into a separate module

- Each module can be tested separately and then brought together with main program
- In a large program, the module can be assigned to different programmers



## CALL INSTRUCTIONS

### ACALL

- ❑ The only difference between `ACALL` and `LCALL` is
  - The target address for `LCALL` can be anywhere within the 64K byte address
  - The target address of `ACALL` must be within a 2K-byte range
- ❑ The use of `ACALL` instead of `LCALL` can save a number of bytes of program ROM space



# CALL INSTRUCTIONS

## ACALL (cont')

```
ORG 0
BACK: MOV A,#55H ;load A with 55H
      MOV P1,A ;send 55H to port 1
      LCALL DELAY ;time delay
      MOV A,#0AAH ;load A with AA (in hex)
      MOV P1,A ;send AAH to port 1
      LCALL DELAY
      SJMP BACK ;keep doing this indefinitely
      ...
      END ;end of asm file
```

### A rewritten program which is more efficiently

```
ORG 0
      MOV A,#55H ;load A with 55H
BACK: MOV P1,A ;send 55H to port 1
      ACALL DELAY ;time delay
      CPL A ;complement reg A
      SJMP BACK ;keep doing this indefinitely
      ...
      END ;end of asm file
```



## TIME DELAY FOR VARIOUS 8051 CHIPS

- ❑ CPU executing an instruction takes a certain number of clock cycles
  - These are referred as to as *machine cycles*
- ❑ The length of machine cycle depends on the frequency of the crystal oscillator connected to 8051
- ❑ In original 8051, one machine cycle lasts 12 oscillator periods

Find the period of the machine cycle for 11.0592 MHz crystal frequency

**Solution:**

$$11.0592/12 = 921.6 \text{ kHz};$$

$$\text{machine cycle is } 1/921.6 \text{ kHz} = 1.085 \mu\text{s}$$



## TIME DELAY FOR VARIOUS 8051 CHIPS (cont')

For 8051 system of 11.0592 MHz, find how long it takes to execute each instruction.

- (a) MOV R3,#55 (b) DEC R3 (c) DJNZ R2 target  
(d) LJMP (e) SJMP (f) NOP (g) MUL AB

**Solution:**

	<i>Machine cycles</i>	<i>Time to execute</i>
(a)	1	$1 \times 1.085 \mu s = 1.085 \mu s$
(b)	1	$1 \times 1.085 \mu s = 1.085 \mu s$
(c)	2	$2 \times 1.085 \mu s = 2.17 \mu s$
(d)	2	$2 \times 1.085 \mu s = 2.17 \mu s$
(e)	2	$2 \times 1.085 \mu s = 2.17 \mu s$
(f)	1	$1 \times 1.085 \mu s = 1.085 \mu s$
(g)	4	$4 \times 1.085 \mu s = 4.34 \mu s$



# TIME DELAY FOR VARIOUS 8051 CHIPS

## Delay Calculation

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
                MOV  A, #55H
AGAIN:          MOV  P1, A
                ACALL DELAY
                CPL  A
                SJMP AGAIN
;---time delay-----
DELAY:          MOV  R3, #200
HERE:          DJNZ R3, HERE
                RET
```

A simple way to short jump to itself in order to keep the microcontroller busy

HERE: SJMP HERE

We can use the following:

SJMP \$

### Solution:

	<i>Machine cycle</i>
DELAY: MOV R3, #200	1
HERE: DJNZ R3, HERE	2
RET	2

Therefore,  $[(200 \times 2) + 1 + 2] \times 1.085 \mu s = 436.255 \mu s$ .



## TIME DELAY FOR VARIOUS 8051 CHIPS

### Increasing Delay Using NOP

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

	<i>Machine Cycle</i>
DELAY: MOV R3, #250	1
HERE: NOP	1
NOP	1
NOP	1
NOP	1
DJNZ R3, HERE	2
RET	2

#### **Solution:**

The time delay inside HERE loop is

$$[250(1+1+1+1+2)] \times 1.085 \mu s = 1627.5 \mu s.$$

Adding the two instructions outside loop we

$$\text{have } 1627.5 \mu s + 3 \times 1.085 \mu s = 1630.755 \mu s$$



# TIME DELAY FOR VARIOUS 8051 CHIPS

## Large Delay Using Nested Loop

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

	<i>Machine Cycle</i>
DELAY: MOV R2, #200	1
AGAIN: MOV R3, #250	1
HERE: NOP	1
NOP	1
DJNZ R3, HERE	2
DJNZ R2, AGAIN	2
RET	2

Notice in nested loop, as in all other time delay loops, the time is approximate since we have ignored the first and last instructions in the subroutine.

### Solution:

For HERE loop, we have  $(4 \times 250) \times 1.085 \mu s = 1085 \mu s$ .  
For AGAIN loop repeats HERE loop 200 times, so we have  $200 \times 1085 \mu s = 217000 \mu s$ . But "MOV R3, #250" and "DJNZ R2, AGAIN" at the start and end of the AGAIN loop add  $(3 \times 200 \times 1.805) = 651 \mu s$ .  
As a result we have  $217000 + 651 = 217651 \mu s$ .



## TIME DELAY FOR VARIOUS 8051 CHIPS

### Delay Calculation for Other 8051

- ❑ Two factors can affect the accuracy of the delay
  - Crystal frequency
    - The duration of the clock period of the machine cycle is a function of this crystal frequency
  - 8051 design
    - The original machine cycle duration was set at 12 clocks
    - Advances in both IC technology and CPU design in recent years have made the 1-clock machine cycle a common feature

#### Clocks per machine cycle for various 8051 versions

Chip/Maker	Clocks per Machine Cycle
AT89C51 Atmel	12
P89C54X2 Philips	6
DS5000 Dallas Semi	4
DS89C420/30/40/50 Dallas Semi	1



## TIME DELAY FOR VARIOUS 8051 CHIPS

### Delay Calculation for Other 8051 (cont')

Find the period of the machine cycle (MC) for various versions of 8051, if XTAL=11.0592 MHz.

(a) AT89C51 (b) P89C54X2 (c) DS5000 (d) DS89C4x0

**Solution:**

(a)  $11.0592\text{MHz}/12 = 921.6\text{kHz};$

MC is  $1/921.6\text{kHz} = 1.085\mu\text{s} = 1085\text{ns}$

(b)  $11.0592\text{MHz}/6 = 1.8432\text{MHz};$

MC is  $1/1.8432\text{MHz} = 0.5425\mu\text{s} = 542\text{ns}$

(c)  $11.0592\text{MHz}/4 = 2.7648\text{MHz};$

MC is  $1/2.7648\text{MHz} = 0.36\mu\text{s} = 360\text{ns}$

(d)  $11.0592\text{MHz}/1 = 11.0592\text{MHz};$

MC is  $1/11.0592\text{MHz} = 0.0904\mu\text{s} = 90\text{ns}$



# TIME DELAY FOR VARIOUS 8051 CHIPS

## Delay Calculation for Other 8051 (cont')

Instruction	8051	DSC89C4x0
MOV R3, #55	1	2
DEC R3	1	1
DJNZ R2 target	2	4
LJMP	2	3
SJMP	2	3
NOP	1	1
MUL AB	4	9

For an AT8051 and DSC89C4x0 system of 11.0592 MHz, find how long it takes to execute each instruction.

- (a) MOV R3, #55 (b) DEC R3 (c) DJNZ R2 target  
(d) LJMP (e) SJMP (f) NOP (g) MUL AB

### Solution:

	<i>AT8051</i>	<i>DS89C4x0</i>
(a)	$1 \times 1085\text{ns} = 1085\text{ns}$	$2 \times 90\text{ns} = 180\text{ns}$
(b)	$1 \times 1085\text{ns} = 1085\text{ns}$	$1 \times 90\text{ns} = 90\text{ns}$
(c)	$2 \times 1085\text{ns} = 2170\text{ns}$	$4 \times 90\text{ns} = 360\text{ns}$
(d)	$2 \times 1085\text{ns} = 2170\text{ns}$	$3 \times 90\text{ns} = 270\text{ns}$
(e)	$2 \times 1085\text{ns} = 2170\text{ns}$	$3 \times 90\text{ns} = 270\text{ns}$
(f)	$1 \times 1085\text{ns} = 1085\text{ns}$	$1 \times 90\text{ns} = 90\text{ns}$
(g)	$4 \times 1085\text{ns} = 4340\text{ns}$	$9 \times 90\text{ns} = 810\text{ns}$

