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

- 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

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

```
;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
```

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:   ...
```

- (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 R5,#0

```
        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
```

LOOP AND JUMP INSTRUCTIONS

Conditional Jumps (cont')

8051 conditional jump instructions

Instructions	Actions
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and Jump if A ≠ 0
CJNE A,byte	Jump if A ≠ byte
CJNE reg,#data	Jump if byte ≠ #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)

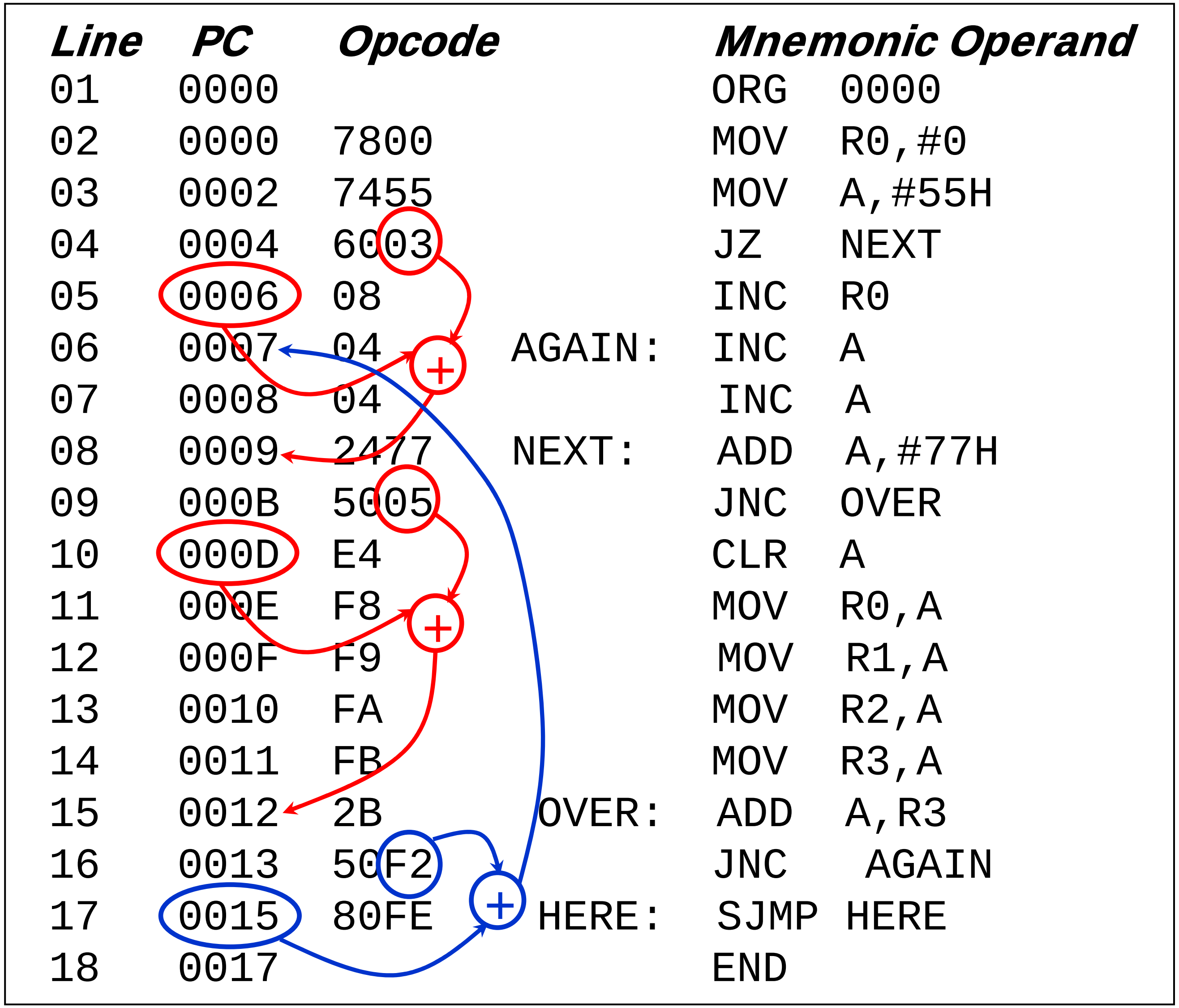
- 2-byte instruction
  - First byte is the opcode
  - Second byte is the relative target address
    - 00 to FFH (forward +127 and backward -128 bytes from the current PC)

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      AGAIN:  INC   A
07    0008  04              INC   A
08    0009  2477    NEXT:   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      OVER:   ADD   A,R3
16    0013  50F2            JNC   AGAIN
17    0015  80FE    HERE:   SJMP  HERE
18    0017                  END
```

- 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
```

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.

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

```
;---------- 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
```

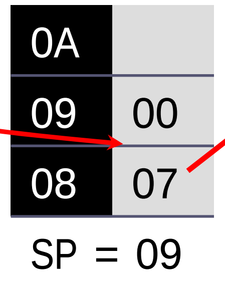
When R5 becomes 0, control falls to the `RET` which pops the address from the stack into the PC and resumes executing the instructions after the `CALL`.

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

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 7DFF             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



Low byte goes first and high byte is last

CALL INSTRUCTIONS

Use PUSH/POP in 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 22                RET            ;return to caller
22 031                                                    e
```

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

After first LCALL

0B		
0A		
09	00	PCH
08	0B	PCL

After PUSH 4

0B		
0A	99	R4
09	00	PCH
08	0B	PCL

After PUSH 5

0B	67	R5
0A	99	R4
09	00	PCH
08	0B	PCL

CALL INSTRUCTIONS

Calling Subroutines

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

HERE:   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
```