OUTLINE

• Loop and Jump instructions

• Call instructions

• Time delay
**LOOP: DJNZ**

- **DJNZ (decrease jump not zero)**
  - DJNZ *reg, label*
    - Decrement the reg by 1 (‘D’)
    - If reg is **not zero** (‘NZ’), jump (‘J’) to the line specified by label; otherwise exit the loop.
  - Example (Demo loop)
    - MOV A, #0
    - MOV R2, #4
    - AGAIN: ADD A, #03
    - DJNZ R2, AGAIN ; reg: R2, label: AGAIN
    - MOV R5, A

<table>
<thead>
<tr>
<th>Loop</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The maximum value that can be held in R2 is ; thus the loop can be repeated for a maximum of times.
**LOOP: DJNZ**

- **Loop inside a loop**
  - More loops can be achieved by embedding one loop inside another
  - Example
    ```
    MOV R1, #0H
    MOV A, #55H
    MOV R3, #3H
    LOOP2: MOV R2, #2H
    LOOP1: CPL A                  ; complement A register
            INC R1                  ; Increment R1 by 1
            DJNZ R2, LOOP1          ; jump back to LOOP1 if R2-1 is not zero
    DJNZ R3, LOOP2                ; jump back to LOOP2 if R3-1 is not zero
    ```
  - Flow chart
  - There are totally 2 x 3 = 6 loops
LOOP: CONDITIONAL JUMP INSTRUCTIONS

- **JZ (jump if A = 0)**
  - **JZ label**
  - Example:
    
    ```
    MOV A, #0FEH
    ADD A, #1H
    JZ OVER
    ADD A, #1H
    JZ OVER
    ADD A, #1H
    JZ OVER
    OVER:
    MOV R0, #0H
    ```
  - Note: JZ can only be used to test register A

- **JNZ (jump if A ≠ 0)**
  - **JNZ label**
  - Example: write a program to determine if R5 contains 0. If so, put 55H in it; otherwise do nothing (NOP)
    
    ```
    MOV A, R5
    JNZ NEXT
    MOV R5, #55H
    NEXT:
    NOP ; no operation
    ```
LOOP: CONDITIONAL JUMP INSTRUCTIONS

- JNC (jump if no carry)
  - JNC label
  - Jump to label if no carry (CY = 0)
  - Example: find the sum of 79H, F5H, and E2H. Put the sum in registers R0 (low byte) and R5 (high byte) (Demo)

```assembly
MOV A, #0
MOV R0, A
MOV R5, A

;-----------------------
MOV A, #79H ; A = A + 79H = 79H
ADD A, #0F5H ; A = 79H + F5H
JNC N2 ; if CY == 0, jump to the next summation
INC R5 ; if CY == 1, increment R5 to record the carry

N2: ADD A, #0E2H ; A = A + E2H
JNC N3 ; if CY == 0, jump to the end
INC R5 ; if CY == 1, increment R5 to record the carry

N3: MOV R0, A
NOP
```
• All conditional jumps are short jump
  – Short jump: the address of the target must be within -128 to +127 bytes of the current PC (program counter)
  – E.g.

<table>
<thead>
<tr>
<th>Addr</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>7400</td>
</tr>
<tr>
<td>0002</td>
<td>6003</td>
</tr>
<tr>
<td>0004</td>
<td>FA</td>
</tr>
<tr>
<td>0005</td>
<td>740A</td>
</tr>
<tr>
<td>0007</td>
<td>7A00</td>
</tr>
</tbody>
</table>

;Target (0007) – PC at JZ (0004) = 7 – 4 = 3

• Why -128 – 127?
  – The length of short jump instructions is 2 bytes
  – Opcode of JZ: 01100000 xxxxxxxxxx (offset)
  – Offset = address of target – PC at JZ (JZ addr+2, in the example, it is 4)
  – The offset is limited to 8 bits (-128 – 127)

• If we want to jump further than -128 or 127, we need to use more bits to represent the jump offset.
**LOOP: SHORT JUMP**

**Example**
- Find the offset of the forward jump instructions

<table>
<thead>
<tr>
<th>Line</th>
<th>Addr</th>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0000</td>
<td></td>
<td>ORG</td>
<td>0000</td>
</tr>
<tr>
<td>02</td>
<td>0000</td>
<td>7800</td>
<td>MOV</td>
<td>R0,#0</td>
</tr>
<tr>
<td>03</td>
<td>0002</td>
<td>7455</td>
<td>MOV</td>
<td>A,#55H</td>
</tr>
<tr>
<td>04</td>
<td>0004</td>
<td>60 XX</td>
<td>JZ</td>
<td>NEXT</td>
</tr>
<tr>
<td>05</td>
<td>0006</td>
<td>08</td>
<td>INC</td>
<td>R0</td>
</tr>
<tr>
<td>06</td>
<td>0007</td>
<td>04</td>
<td>AGAIN:</td>
<td>INC A</td>
</tr>
<tr>
<td>07</td>
<td>0008</td>
<td>04</td>
<td>INC A</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>0009</td>
<td>2477</td>
<td>NEXT:</td>
<td>ADD A,#77h</td>
</tr>
<tr>
<td>09</td>
<td>000B</td>
<td>50 XX</td>
<td>JNC</td>
<td>OVER</td>
</tr>
<tr>
<td>10</td>
<td>000D</td>
<td>E4</td>
<td>CLR A</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>000E</td>
<td>F8</td>
<td>MOV R0,A</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>000F</td>
<td>F9</td>
<td>MOV R1,A</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0010</td>
<td>FA</td>
<td>MOV R2,A</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0011</td>
<td>FB</td>
<td>MOV R3,A</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0012</td>
<td>2B</td>
<td>OVER:</td>
<td>ADD A,R3</td>
</tr>
<tr>
<td>16</td>
<td>0013</td>
<td>50F2</td>
<td>JNC AGAIN</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0015</td>
<td>80 FE</td>
<td>HERE:</td>
<td>SJMP HERE</td>
</tr>
<tr>
<td>18</td>
<td>0017</td>
<td></td>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>
LOOP: UNCONDITIONAL JUMP

- **LJMP (long jump)**
  - **LJMP label**
  - Jump to anywhere in the program
  - Opcode (3 bytes)
    - 00000010 A15-A8 A7-A0
    - The 2\textsuperscript{nd} and 3\textsuperscript{rd} bytes represent the absolute address in ROM
    - Recall: PC has 16 bits => ROM address range is 0000 - FFFFH => 16 bits are enough to label any address in ROM
  - Example
    ```
    ORG 0H
    LJMP FARAWAY ; opcode 02F000H
    ORG 0F000H
    FARAWAY: MOV A, 55H
    ```
**LOOP: UNCONDITIONAL JUMP**

- **SJMP (short jump)**
  - SJMP *label*
  - Jump to an address within -128 – 127 of current PC
  - Opcode (2 byte)
    
    \[
    \text{10000000 xxxxxxxx (offset)}
    \]
  - The calculation of offset is the same as conditional jumps (JZ, JNZ, JNC)
  - Example: find the offset of the SJMP instructions (xx and yy in the comments)

```assembly
ORG 0H
  SJMP TAGT1 ; opcode 80xxH
  MOV A, #0 ; opcode 7400H
ORG 10
  TAGT1:  SJMP TAGT2 ; opcode 80yyH
          MOV A, #0
          ORG 35
  TAGT2:  MOV A, #55H
```

- What will happen if the target is out of the range of [-128, 127] of current PC?
OUTLINE

• Loop and Jump instructions

• Call instructions

• Time delay
CALL INSTRUCTIONS

• **Subroutine**
  – A section of code that can perform a specific task (e.g. introduce a certain amount of delay)
  – If a task needs to be performed frequently, it’s better to structure the corresponding codes as a subroutine
    • Save memory space
    • Better program structure
  – Subroutines are invoked by call instructions

• **There are two call instructions in 8051**
  – LCALL (3 byte instruction)
    • 16 bits (2 bytes) are used to represent target address
    • Long call, the subroutine can be placed anywhere in the ROM
  – ACALL (2 byte instruction)
    • Absolute call
    • Only 11 bits are used to represent target address
      – The target address must be in the same 2KByte block of memory as the opcode following the ACALL instruction.
CALL: LCALL

- **LCALL**
  - Long call, 3-byte instruction
  - Opcode: 00010010 A15-A8 A7-A0 ; the last two bytes are used to represent target address
  - Can be used to call subroutines located anywhere within the 64KB of the ROM.
  - Example

```
ORG 0
BACK: MOV A, #55H
      MOV P1, A ; send 55H to port 1
      LCALL DELAY ; call the subroutine delay
      MOV A, #0AAH
      MOV P1, A ; send AAH to port 1
      LCALL DELAY
      SJMP BACK

;--------------begin of subroutine--------------
ORG 300H
DELAY: MOV R5, #0FFH
AGAIN: DJNZ R5, AGAIN
       RET ; return to caller
;--------------end of subroutine--------------
```

END
CALL: CALL AND STACK

• Call instructions and stack
  – After ‘LCALL’ is executed, PC is changed to the starting address of the subroutine
    • E.g. after LCALL, PC points to address 0300H
  – After the subroutine is done (‘RET’ is executed), PC goes back to the instruction that follows ‘LCALL’
    • E.g. after RET, PC points back to address 0007H (‘MOV A, #0AAH’)
  – How does the CPU know where the PC should point to after the subroutine? (DEMO)
    • Before loading the PC with the address of the subroutine (0300H), the CPU automatically push the address of the next instruction into stack.
    • After RET is executed, the CPU automatically pop the address back to PC.

```assembly
001 0000 ORG 0
002 0000 7455 BACK: MOV A, #55H ; load A with 55H
003 0002 F590 MOV P1, A ; send 55H to port 1
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 port 1
007 000B 120300 LCALL DELAY
008 000E 80F0 SJMP BACK ; keep doing this
009 0010 ; this is the delay subroutine
010 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
```
CALL: CALL AND STACK

• Call instructions and stack (Cont’d)
  – Each address is 16 bits (recall: PC is a 16-bit register)
    • Each PUSH can put in 8 bit → two PUSH instructions are used
    • Similarly, two POP instructions are used to restore the address to PC.
  – If you use stack in a subroutine, you MUST use EQUAL number of PUSH and POP
    • Unequal number of PUSH and POP will result in a wrong value being restored (DEMO LCALL)
    • When you exit a subroutine, the SP should always point to the return address of the subroutine
### CALL: CALL AND STACK

#### Example

- Analyze the contents of the stack and PC after LCALL and RET

<table>
<thead>
<tr>
<th>addr</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>7455</td>
<td>BACK: MOV A, #55H</td>
</tr>
<tr>
<td>0002</td>
<td>F590</td>
<td>MOV P1, A</td>
</tr>
<tr>
<td>0004</td>
<td>7C99</td>
<td>MOV R4, #99H</td>
</tr>
<tr>
<td>0006</td>
<td>7D67</td>
<td>MOV R5, #67H</td>
</tr>
<tr>
<td>0008</td>
<td>120300</td>
<td>LCALL TEST</td>
</tr>
<tr>
<td>000B</td>
<td>74AA</td>
<td>MOV A, #0AAH</td>
</tr>
<tr>
<td>000D</td>
<td>80F1</td>
<td>SJMP BACK</td>
</tr>
</tbody>
</table>

#### CODE

```
ORG 300H

0300  C004   TEST:  PUSH 4
0302  C005   PUSH 5
0304  D001   POP 1
0306  D002   POP2
0308  22     RET
```
CALL: ACALL

**ACALL**

- Absolute call, 2-byte instruction
- 11-bits are used to represent the target address
  - Opcode: A10A9A8 10001 A7-A0
  - **A15-A11** of the target address **must be the same** as that of the opcode following the ACALL instruction => in the same 2KB block
- The ONLY difference between ACALL and LCALL is the limit on target address
- Using ACALL will save 1 byte of memory space.
• Loop and Jump instructions

• Call instructions

• Time delay
• **Terminology**
  - Clock
    • A crystal oscillator is connected to 8051 to provide clock source for 8051.
    • Typical clock frequency (f): 11.0592 MHz, 16 MHz, 20 MHz.
    • **Oscillator period (T):**
  - Machine cycle
    • A basic operation performed by CPU to execute an instruction.
    • Original 8051
      - 1 machine cycle = 12 oscillator periods
    • DS89C450
      - 1 machine cycle = 1 oscillator period
    • Different instructions require different number of machine cycles
      - E.g. original 8051
        1 machine cycle: ADD, MOV R3, A
        2 machine cycles: MOV 08, R2
        4 machine cycles: MUL, DIV
    • Machine cycles can be found at Table A-1 in Appendix A (p.554).
    • It takes different amount of time to execute different instructions.
Example

For an 8051 system with 1 machine cycle = 12 oscillator periods. If the clock frequency is 11.0592 MHz,
(1) What is the duration of 1 machine cycle?
(2) find how long it takes to execute each of the following instructions

- MOV R3, A
- MOV 12H, R1
- NOP
- DJNZ R2, AGAIN

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Machine Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV Rn, A</td>
<td>1</td>
</tr>
<tr>
<td>MOV direct, Rn</td>
<td>2</td>
</tr>
<tr>
<td>NOP</td>
<td>1</td>
</tr>
<tr>
<td>DJNZ Rn, target</td>
<td>2</td>
</tr>
</tbody>
</table>
**Example:**

- For an 8051 system with 1 machine cycle = 12 oscillator periods. If the clock frequency is 11.0592 MHz. Find the delay incurred by the subroutine.

    ORG 300H
    DELAY: MOV R3, #200 ; 1 machine cycle
    HERE:  DJNZ R3, HERE ; 2 machine cycle
    RET     ; 2 machine cycle
**DELAY: DS89C450**

- **DS89C450**
  - 1 machine cycle = 1 oscillator clock period
  - The machine cycles for all instructions can be found in the user guide of DS89C4x0

- **Example:**
  - A 89C450 is connected to an oscillator with frequency 11.0592MHz. Find how long it takes to execute the following instruction
    (a) MOV R3, #55    (b) DJNZ R2, target

<table>
<thead>
<tr>
<th>Instruction</th>
<th>8051</th>
<th>DS89C4x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV R3,#value</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DEC Rx</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DJNZ</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>LJMP</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>SJMP</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>NOP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MUL AB</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
• **Example**
  - A DS89C450 is connected to a 11.0592 MHz XTAL. Find the time delay in the following subroutine

```
DELAY:    ; machine cycles
          ; 2
  MOV R2, #200  ; 2
AGAIN:   ; 2
  MOV R3, #250  ; 2
HERE:    ; 1
  NOP  ; 1
  NOP  ; 1
DJNZ R3, HERE  ; 4
DJNZ R2, AGAIN  ; 4
RET  ; 3
```
**Example**

Write a program to toggle all the bits of P1 every 200 ms (55H => AAH => 55H …) with DS89C450 and 11.0592 MHz XTAL.

```
MOV R1, #9  ; 2
A1:  MOV R2, #242  ; 2
A2:  MOV R3, #255  ; 2
A3:  DJNZ R3, A3  ; 4
     DJNZ R2, A2  ; 4
     DJNZ R1, A1  ; 4
```