

Module 3

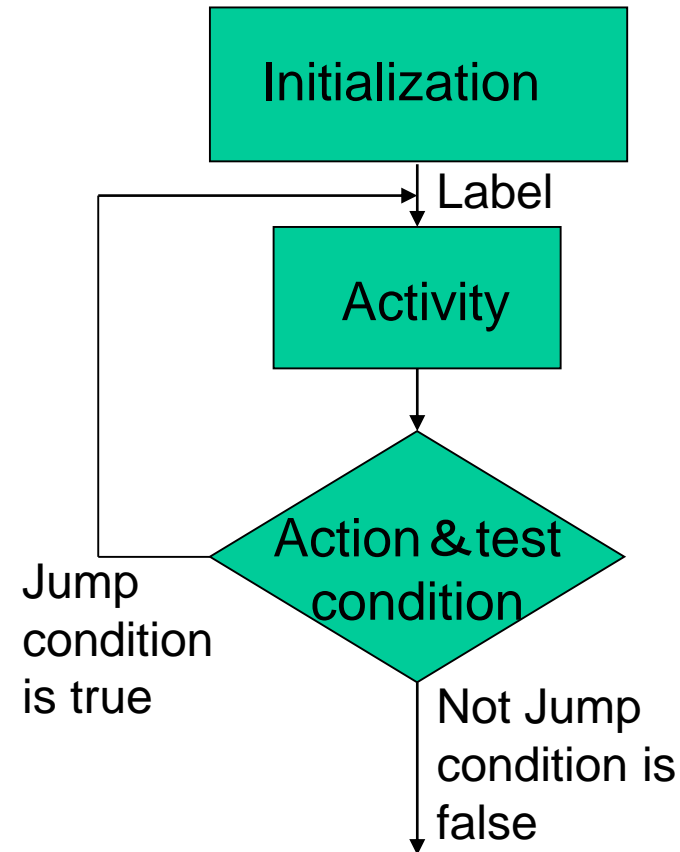
Jump, Loop, and Call (Branching) Instructions

Outline

- ❑ Conditional Jumps
- ❑ Subroutine Calls
- ❑ Unconditional Jumps
- ❑ Machine Cycles and Execution Times

Looping in the 8051 with Conditional Jump Instructions

- ❑ Repeating a sequence of instructions a certain number of times is called a **loop**.
 - Activity works several times.
 - It needs some **control transfer** instructions.
 - 8051 jump instructions
 - do activity first if exists
 - test the condition later
 - DJNZ, JZ, JNC



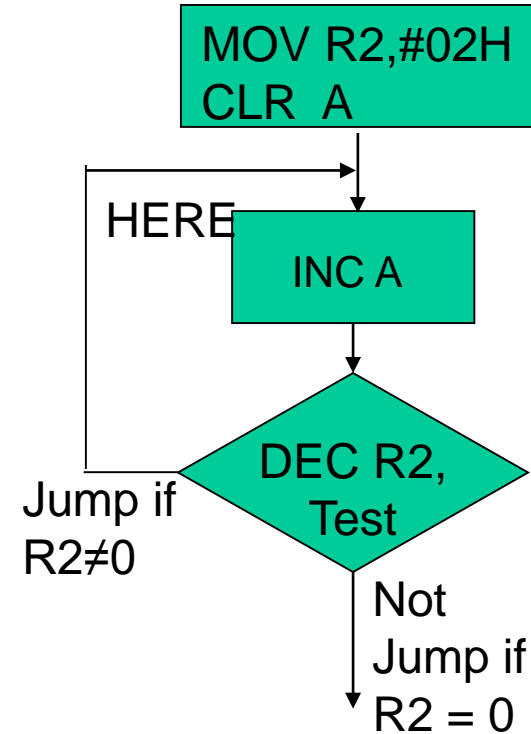
DJNZ (Decrement and Jump if Not Zero)

- Decrement and jump if not zero

DJNZ Rn, target

```
        MOV    R2 , #02H
        CLR    A
HERE:    INC    A
        DJNZ   R2 , HERE
```

- Rn is one of the registers R0 - R7.
- Target is a label. A label HERE is the ROM address of the instruction INC A.
- 2 times in the loop for R2=2,1,0 \Rightarrow A=2
- 2 times INC, DJNZ; 1 time jump



DJNZ - more

- Direct access mode

- DJNZ direct, target**

- ```
MOV 30H, #02 ;X=the value in RAM 30H
CLR A ;A=0
HERE: INC A ;increase A by 1
 DJNZ 30H, HERE ;Decrement X and
 ;Jump to HERE if X≠0
```

- Direct means “directly access the RAM with address”.

# Example

Write a program to

(a) clear ACC, then

(b) add 3 to the accumulator ten times.

(c) save the result in R5

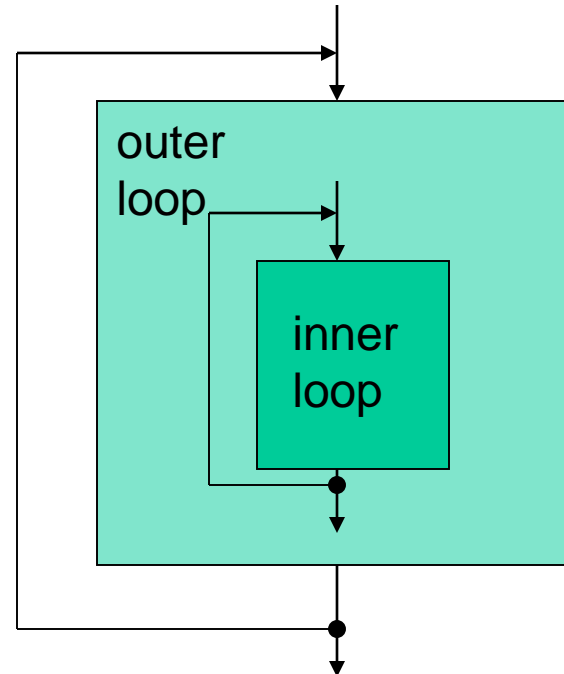
## Solution:

;This program adds value 3 to the ACC ten times

|                  |                                   |
|------------------|-----------------------------------|
| CLR A            | ;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<br>;times) |
| MOV R5,A         | ;save A in R5                     |

# Nested Loops

- ❑ A single loop is repeated at most 256 times (HOW?)
- ❑ If we want to repeat an action more times than 256, we use a loop inside a loop.
- ❑ This is called a nested loop.
- ❑ For example:
  - The inner loop is 256
  - The outer loop is 2
  - Total  $256 \times 2 = 512$



# Example

Write a program to

- (a) load the accumulator with the value 55H, and
- (b) complement the ACC 700 times.

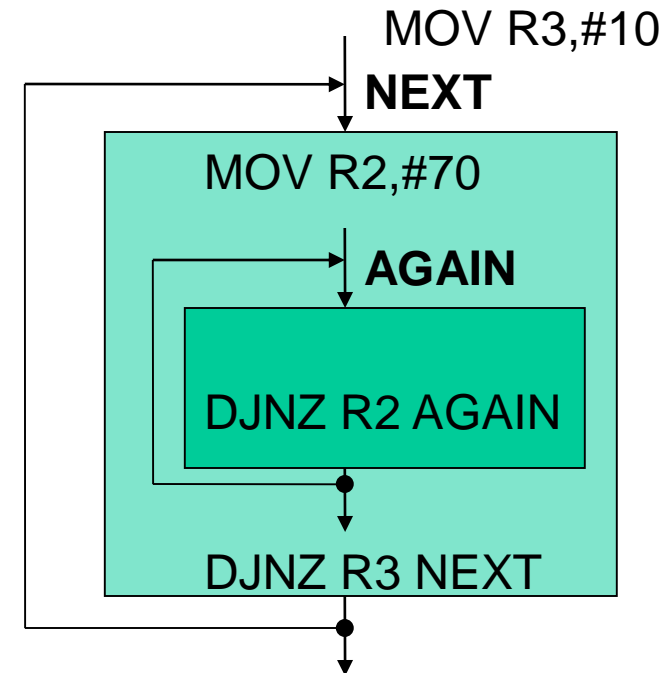
## Solution:

The following code shows how to use R2 and R3 for the count.

$$700 = 10 \times 70$$

Inner loop: R2=70

Outer loop: R3=10

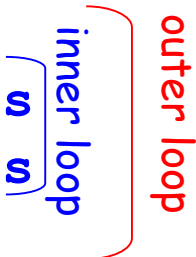




# Example

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

```
MOV A, #55H ; 1 time
MOV R3, #10 ; 1 time
NEXT: MOV R2, #70 ; 10 times
AGAIN: CPL A ; 10x70 times
 DJNZ R2, AGAIN; 10x70 times
 DJNZ R3, NEXT ; 10 times
```



# 8051 Conditional Jump Instructions

## Byte level

|      |                            |                                                     |   |    |
|------|----------------------------|-----------------------------------------------------|---|----|
| JZ   | rel                        | Jump if Accumulator is Zero                         | 2 | 24 |
| JNZ  | rel                        | Jump if Accumulator is Not Zero                     | 2 | 24 |
| CJNE | A,direct,rel               | Compare direct byte to Acc and Jump if Not Equal    | 3 | 24 |
| CJNE | A,#data,rel                | Compare immediate to Acc and Jump if Not Equal      | 3 | 24 |
| CJNE | R <sub>n</sub> ,#data,rel  | Compare immediate to register and Jump if Not Equal | 3 | 24 |
| CJNE | @R <sub>i</sub> ,#data,rel | Compare immediate to indirect and Jump if Not Equal | 3 | 24 |
| DJNZ | R <sub>n</sub> ,rel        | Decrement register and Jump if Not Zero             | 2 | 24 |
| DJNZ | direct,rel                 | Decrement direct byte and Jump if Not Zero          | 3 | 24 |

## Bit level

|     |         |                                       |   |    |
|-----|---------|---------------------------------------|---|----|
| JC  | rel     | Jump if Carry is set                  | 2 | 24 |
| JNC | rel     | Jump if Carry not set                 | 2 | 24 |
| JB  | bit,rel | Jump if direct Bit is set             | 3 | 24 |
| JNB | bit,rel | Jump if direct Bit is Not set         | 3 | 24 |
| JBC | bit,rel | Jump if direct Bit is set & clear bit | 3 | 24 |

# JZ

- Jump if A = zero

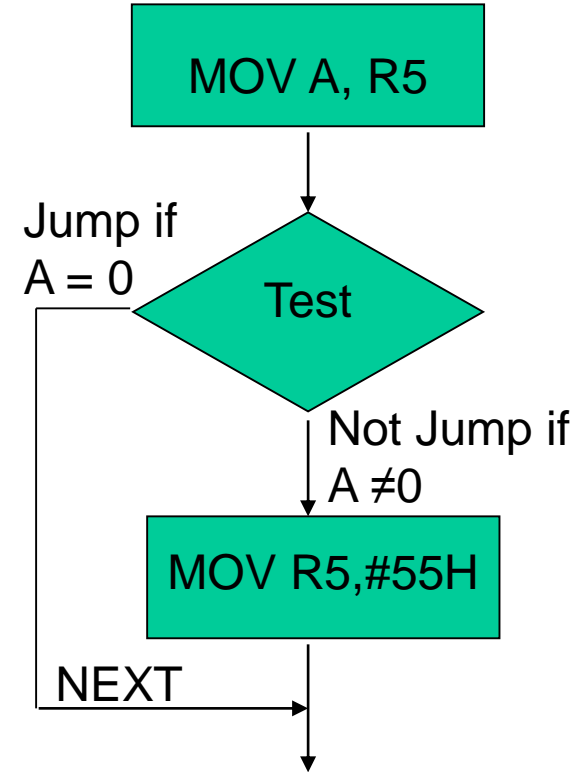
## JZ target

MOV A, R5

JZ NEXT

MOV R5, #55H

NEXT: ...



- This instruction examines the contents of the ACC and jumps if ACC has value 0.

# JNZ

- Jump if A is not zero

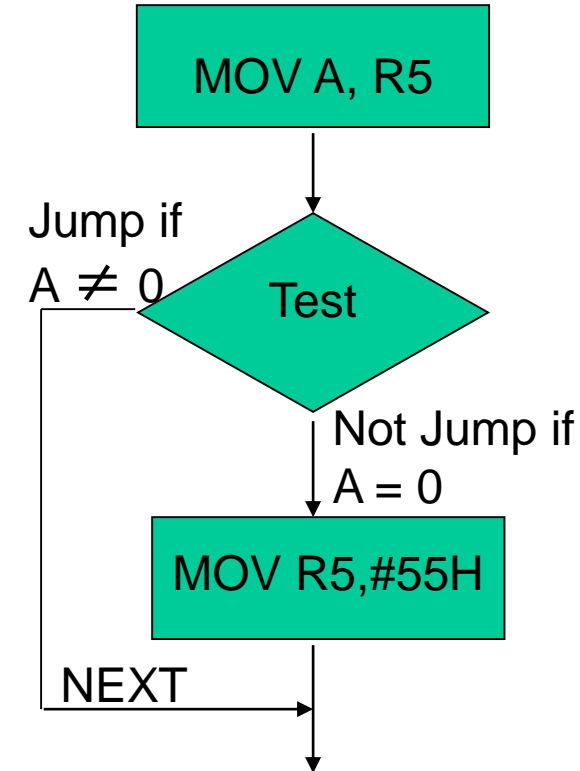
## JNZ target

MOV A,R5

JNZ NEXT

MOV R5,#55H

NEXT: . . .



- This instruction examines the contents of the ACC and jumps if ACC is not 0.

# JC, JNC

- Jump if (no) carry

**JNC target**

```
MOV A, #0FFH
```

```
ADD A, #01H
```

```
JNC NEXT
```

```
INC R5
```

**NEXT: ...**

- CY is PSW.7
- This instruction examines the CY flag, and if it is zero it will jump to the target address.

# Using BIT and EQU Directives

- ❑ Assign bit-addressable I/O and RAM locations

name BIT bit-address



bit-address → name

```
OVEN_HOT BIT P2.3
```

```
HERE: JNB OVEN_HOT, HERE
```

- ❑ We can also use the EQU directive to assign addresses.

# Example

- ❑ Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

## Solution:

**OVEN\_HOT BIT P2.3**

**BUZZER BIT P1.5**

**HERE : JNB OVEN\_HOT, HERE**

**SETB BUZZER**

**ACALL DELAY**

**CPL BUZZER**

**ACALL DELAY**

**SJMP HERE**

❑ A switch (SW) is connected to pin P1.7. Write a program to check the status of SW and perform the following:

(a) If SW=0, send "NO" to P2.

(a) If SW=1, send "YES" to P2.

```
SW EQU P1.7 ;bit address
MYDATA EQU P2 ;byte address

HERE: SETB SW ;make P1.7 an input
 MOV C, SW ;save P1.7 to carry
 JC OVER
 MOV MYDATA, #'N' ;SW=0
 MOV MYDATA, #'O'
 SJMP HERE
OVER: MOV MYDATA, #'Y' ;SW=1
 MOV MYDATA, #'E'
 MOV MYDATA, #'S'
 SJMP HERE
```



# Most important comparison instruction – CJNE (Compare and Jump if Not Equal)

CJNE destination, source,  
relative address

```
 MOV A, #55H
 CJNE A, #99H, NEXT
 ... ; do
here if A=99H
NEXT: ... ; jump
here if A≠99H
```

- The compare instruction is really a subtraction, except that the operands themselves remain unchanged.
- Flags are changed according to the execution of the SUBB instruction.

|      |                            |                                                     |   |    |
|------|----------------------------|-----------------------------------------------------|---|----|
| CJNE | A,direct,rel               | Compare direct byte to Acc and Jump if Not Equal    | 3 | 24 |
| CJNE | A,#data,rel                | Compare immediate to Acc and Jump if Not Equal      | 3 | 24 |
| CJNE | R <sub>n</sub> ,#data,rel  | Compare immediate to register and Jump if Not Equal | 3 | 24 |
| CJNE | @R <sub>i</sub> ,#data,rel | Compare immediate to indirect and Jump if Not Equal | 3 | 24 |

# CJNE

| <u>Compare</u>            | <u>Carry Flag</u> |
|---------------------------|-------------------|
| destination $\geq$ source | CY = 0            |
| destination $<$ source    | CY = 1            |

- This instruction affects the carry flag only.

```
CJNE R5, #80, NEXT
```

```
... ;do here if R5=80
```

```
NEXT: JNC LAR
```

```
... ;do here if R5<80
```

```
LAR: ... ;do here if R5>80
```

# Example

Assume that P1 is an input port connected to a temperature sensor. Write a program to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following.

|           |             |
|-----------|-------------|
| If T = 75 | then A = 75 |
| If T < 75 | then R1 = T |
| If T > 75 | then R2 = T |

## Solution:

```
MOV P1, #0FFH ;make P1 an input port
MOV A, P1 ;read P1 port
CJNE A, #75, OVER ;jump if A not equal to 75
SJMP EXIT ;A=75
OVER: JNC NEXT ;
MOV R1, A ;A<75, save A R1
SJMP EXIT ;
NEXT: MOV R2, A ;A>75, save A in R2
EXIT: ...
```

# Example

- ❑ Search a null-terminated string of ASCII codes and count the number of digit characters ("0"- "9"). The string is stored in memory beginning at internal ROM location 1000H. Put the count in the accumulator.

# Solution

```
 MOV DPTR,#1000H
 MOV R7,#0
REPEAT: CLR A
 MOVC A,@A+DPTR
 INC DPTR
 CJNE A,#'0',$+3 ; $ current PC
 JC UNTIL
 CJNE A, #'9'+1,$+3 ; $ current PC
 JNC UNTIL
THEN: INC R7
UNTIL: CJNE A,#0,REPEAT
 MOV A,R7
HERE: SJMP HERE
 END
```

# Example

Write a program to get the x value from P1 and send  $x^2$  to P2, continuously.

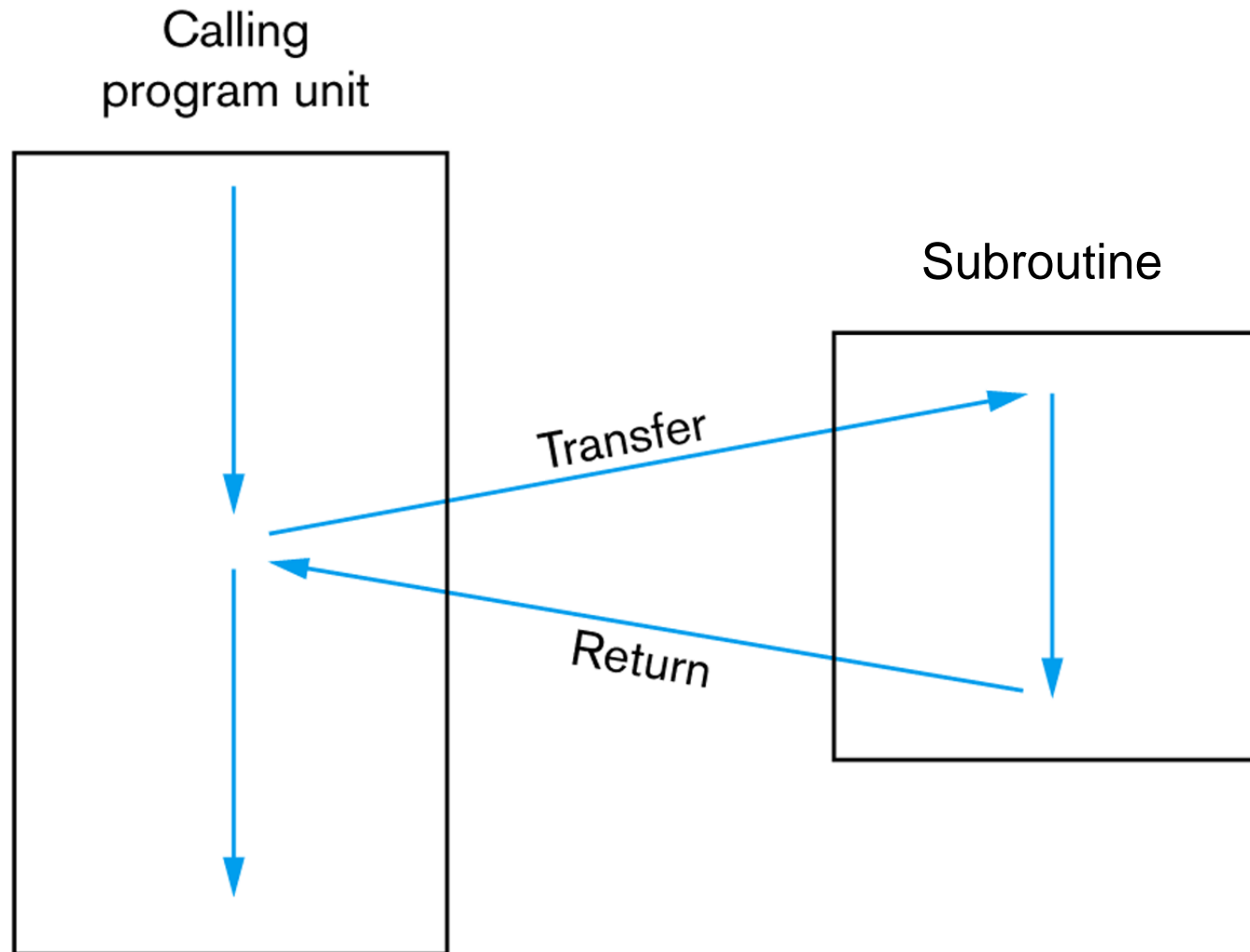
## Solution:

```
 ORG 0
 MOV DPTR, #XSQR_TABLE
 MOV A, #0FFH
 MOV P1, A ; P1 as INPUT PORT
BACK: MOV A, P1 ; Get x
 MOVC A, @A+DPTR ; Count the addr.
 MOV P2, A ; Issue it to P2
 SJMP BACK
 ORG 300H
XSQR_TABLE:
 DB 0, 1, 4, 9, 16, 25, 36, 49, 64, 81
 END
```

# Calls

- ❑ Another control transfer instruction is the CALL instruction, which is used to call a 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
- ❑ In the 8051 there are two instructions for call:
  - LCALL long call Examples 3-8 and 3-10
  - ACALL absolute call Examples 3-11 and 3-12

# Calls - more





# 8051 Main Program that calls 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 subroutine 1
;
SUBR_2:

 RET
;-----end of subroutine 2

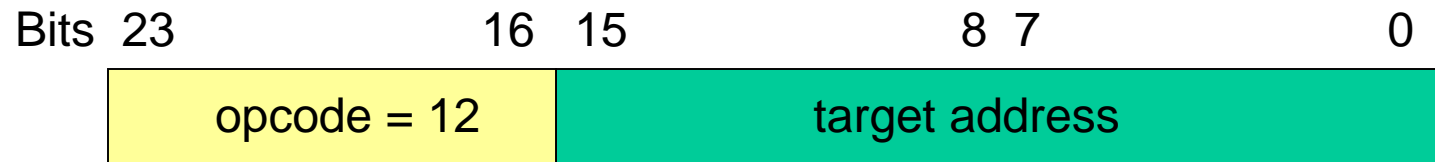
SUBR_3:

 RET
;-----end of subroutine 3
 END ;end of the asm file
```

| PROGRAM BRANCHING |        |                          |   |    |
|-------------------|--------|--------------------------|---|----|
| ACALL             | addr11 | Absolute Subroutine Call | 2 | 24 |
| LCALL             | addr16 | Long Subroutine Call     | 3 | 24 |
| RET               |        | Return from Subroutine   | 1 | 24 |

# LCALL (Long Call)

- ❑ A 3-byte instruction
  - The first byte is the opcode
  - The next two bytes are the target address
- ❑ LCALL is used to jump to any address location within the 64K byte code space of the 8051.



# LCALL

- ❑ Jump to a new address

LCALL 16-bit-target-addr.

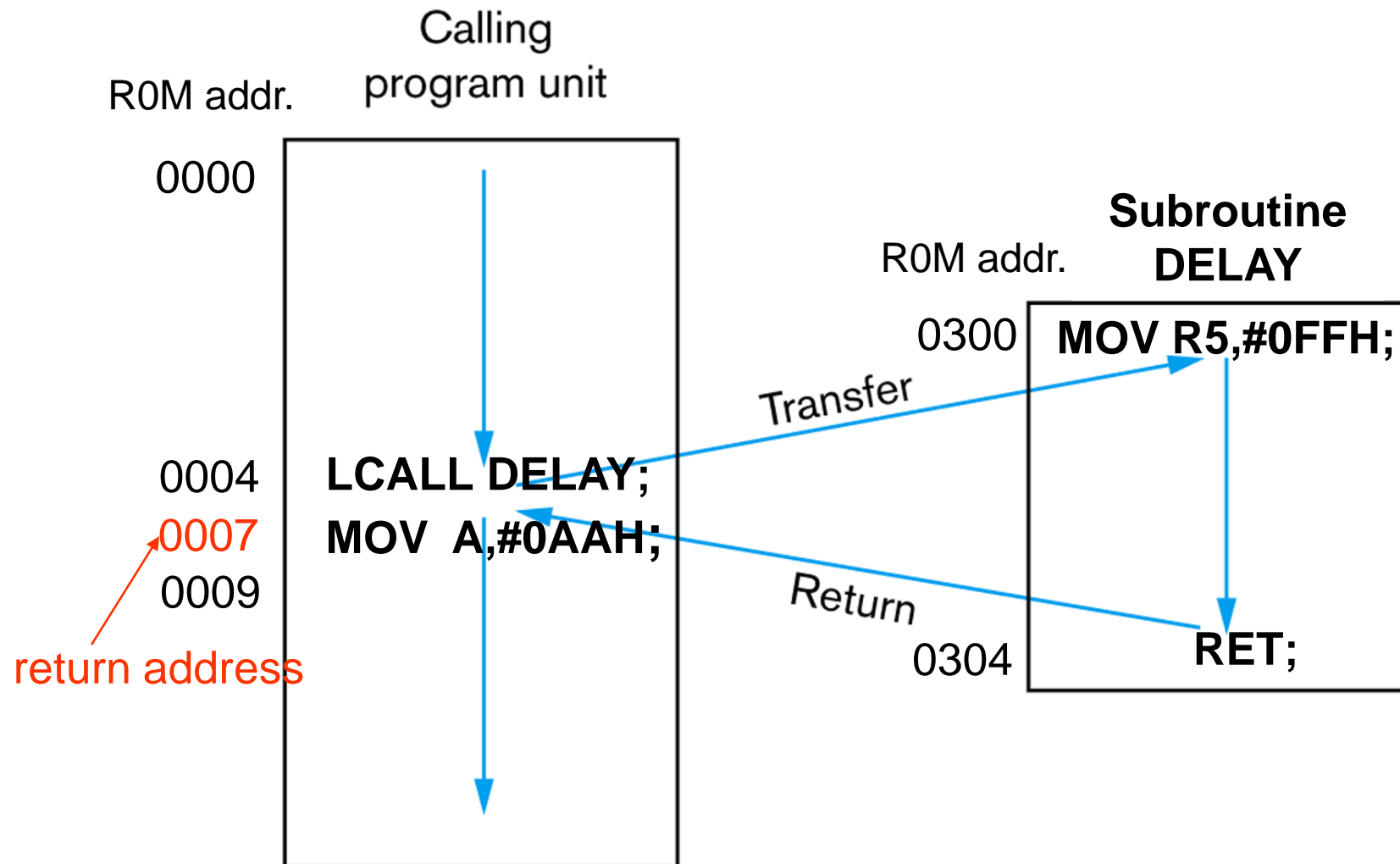
| Line | PC   | Opcode | Mnemonic | Operand  |
|------|------|--------|----------|----------|
| 04   | 0004 | 120300 | LCALL    | DELAY    |
| 05   | 0007 | 74AA   | MOV      | A, #0AAH |

|    |      |      |            |           |
|----|------|------|------------|-----------|
| 11 | 0300 |      | ORG        | 300H      |
| 12 | 0300 | 7DFF | DELAY: MOV | R5, #0FFH |
|    |      |      | ...        |           |
| 15 | 0304 | 22   | RET        |           |

subroutine

- The opcode of LCALL is 12.
- The target address is 0300.
- The return address is 0007

# LCALL - more



# Example

Write a program to toggle all the bits of port 1 by sending to it the values 55H and AAH continuously. Put a time delay in between each issuing of data to port 1.

## Solution:

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

# Example

Analyze the contents of the stack after the execution of the first LCALL.

## Solution:

When the first LCALL is executed, the address of the instruction “MOV A,#0AAH” is saved *on the stack*. Notice that *the low byte goes first* and *the high byte is last*. The last instruction of the called subroutine must be a RET instruction which directs the CPU to POP the top bytes of the stack into the PC and resume executing at address 07. The diagram shows the stack frame after the first LCALL.

|         |    |
|---------|----|
| 0A      |    |
| 09      | 00 |
| 08      | 07 |
| SP = 09 |    |

# The process of calling a subroutine

- ❑ After execution of the called subroutine, the 8051 must know where to come back to.
- ❑ The process of calling a subroutine :
  - A subroutine is called by CALL instructions.
  - The 8051 increments PC to point to the next instruction.
  - The 8051 pushes PC onto the stack.
  - The 8051 copies the target address to the PC.
  - The 8051 fetches instructions from the new location.
  - When the instruction RET is fetched, the subroutine ends.
  - The 8051 pops the return address from the stack.
  - The 8051 copies the return address to the PC.
  - The 8051 fetches instructions from the new location.

# Example

Analyze the stack for the first LCALL instruction in the following program.

```

01 0000 ORG 0
02 0000 7455 BACK: MOV A,#55H ;load A with 55H
03 0002 F590 MOV P1,A ;send 55H to port1
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 port 1
09 000F 120300 LCALL DELAY
10 0012 80EC SJMP BACK ;keep 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
15 0304 7CFF MOV R4,#0FFH ;R4=FFH
16 0306 7DFF NEXT: MOV R5,#0FFH ;R5=255
17 0308 DDFE AGAIN: DJNZ R5,AGAIN
18 030A DCFA DJNZ R4,NEXT
19 030C D005 POP 5 ;POP INTO R5
20 030E D004 POP 4 ;POP INTO R4
21 0310 22 RET ;return to caller
22 0311 END ;end of asm file

```



# Example

## Solution:

The stack keeps track of where the CPU should return after completing the subroutine. For this reason, the number of PUSH and POP instructions must always match in any called subroutine.

After the first  
LCALL

|    |    |     |
|----|----|-----|
| 0B |    |     |
| 0A |    |     |
| 09 | 00 | PCH |
| 08 | 0B | PCL |

SP=09

After PUSH 4

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

SP=0A

After PUSH 5

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

SP=0B

# Example

## Solution:

- Also notice that for the PUSH and POP instructions we must specify the direct address of the register being pushed or popped.

After the POP 5

|    |    |     |
|----|----|-----|
| 0B | 67 |     |
| 0A | 99 | R4  |
| 09 | 00 | PCH |
| 08 | 0B | PCL |

SP=0A

After POP 4

|    |    |     |
|----|----|-----|
| 0B | 67 |     |
| 0A | 99 |     |
| 09 | 00 | PCH |
| 08 | 0B | PCL |

SP=09

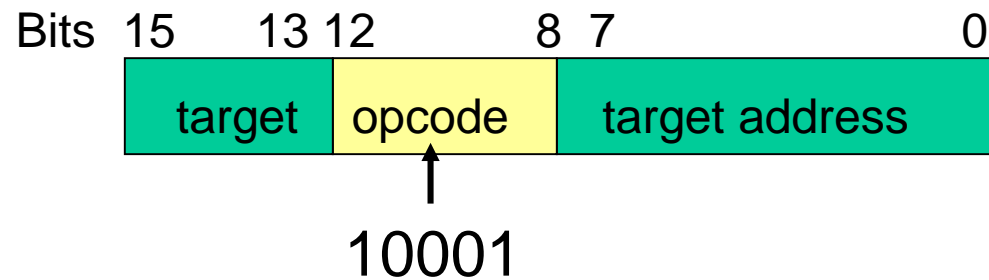
After RET

|    |    |  |
|----|----|--|
| 0B | 67 |  |
| 0A | 99 |  |
| 09 | 00 |  |
| 08 | 0B |  |

SP=07

# ACALL

- a 2-byte instruction
  - The target address must be within 2K bytes of program memory.
  - Using ACALL can save memory space compared to using LCALL.
  - Also see AJMP (Absolute Jump)



# ACALL - more

- Jump to a new address

ACALL target

| Line | PC   | Opcode | Mnemonic   | Operand   |
|------|------|--------|------------|-----------|
| 04   | 0004 | 7100   | ACALL      | DELAY     |
| 05   | 0006 | 74 AA  | MOV        | A, #0AAH  |
| ...  |      |        |            |           |
| 11   | 0300 |        | ORG        | 300H      |
| 12   | 0300 | 7DFF   | DELAY: MOV | R5, #0FFH |
| ...  |      |        |            |           |
| 15   | 0304 | 22     | RET        |           |


Calculation of the opcode (by assembler):

- The opcode of ACALL is 10001B (5 bits).
- The target address is 0300H=0000 0011 0000 0000B (16 bits).
- The binary code is 0111 0001 0000 0000 B=7100H

# ACALL - more

Calculation of the target address (by 8051)

❑ ACALL DELAY → 7100H: **011** 1 0001 **0000 0000**



Last 11 bits of the target address: **011 0000 0000**

❑ PC points to the address of the next instruction

❑ Take the first 5 bits from PC

PC: 0006H : **0000 0** 0000 0000 0110



0000 0011 0000 0000: 0300H

❑ First 5 bits of the address of the following instruction and the target address must be the same

# Another Approach to Read from the Internal ROM: MOVC A,@A+PC

- Look-up table SQUR has the squares of values 0 to 9 and R3 has a value in the range 0 to 9. Write a program to fetch the squares.

```
PROC: MOV A,R3
```

```
INC A ; to bypass the RET
```

```
MOVC A,@A+PC
```

```
RET
```

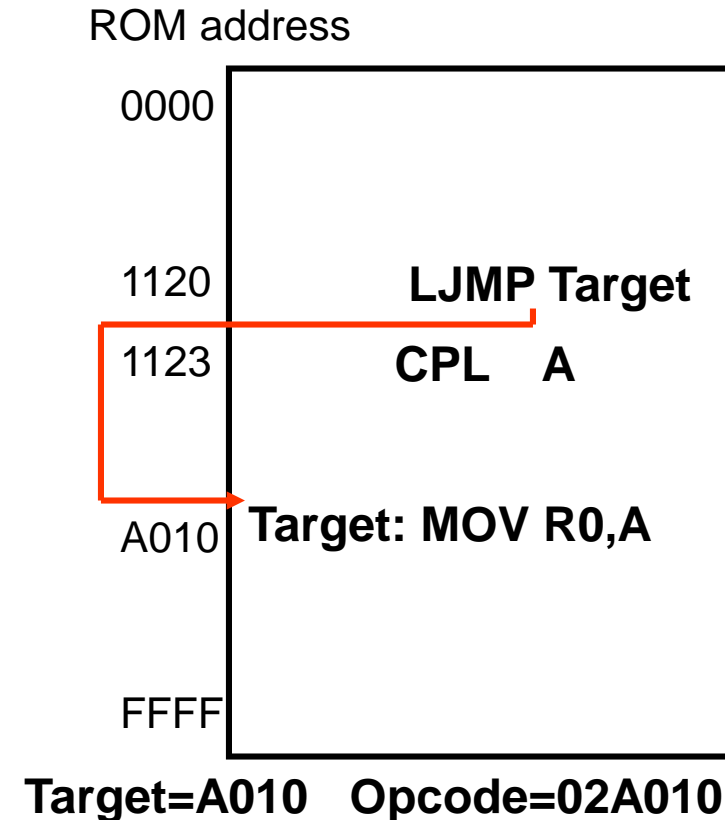
```
SQUR:DB 0,1,4,9,16,25,36,49,64,81
```

This method is preferable to MOVC A,@A+DPTR if we do not want to divide the code space into two separate areas

# Unconditional Jumps LJMP (Long Jump)

## LJMP target

- ❑ A 3-byte instruction
  - The first byte is the opcode
  - The next two bytes are the target address (real address)
- ❑ LJMP is used to jump to any address location within the 64K byte code space of the 8051.

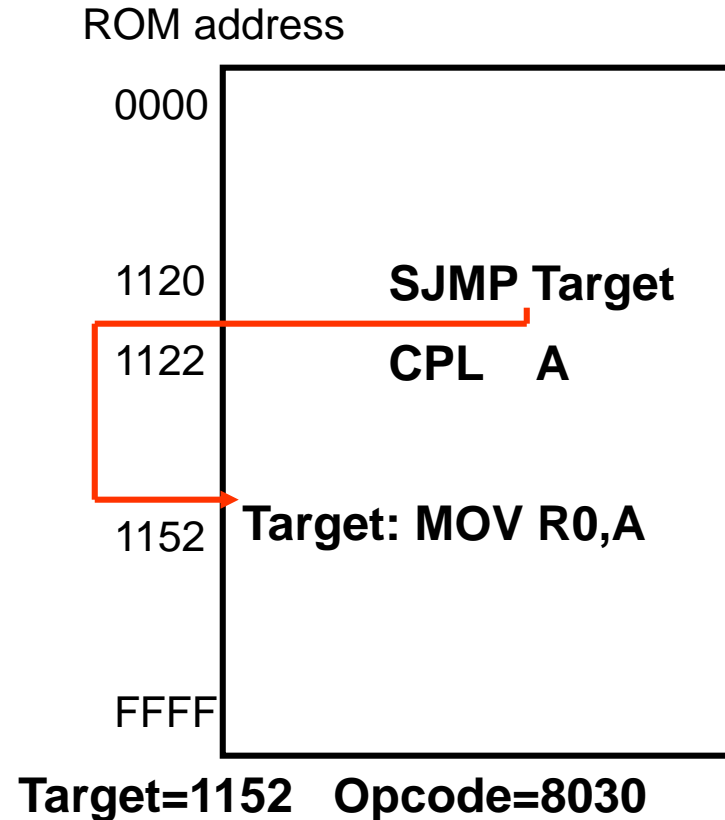


- ❑ Not all 8051 members have that much on-chip program ROM

# SJMP (Short Jump)

## SJMP target

- A 2-byte instruction
  - The first byte is the opcode
  - The second byte is the signed number displacement, which is added to the PC to get the target address.
  - The address is referred to as a *relative address* since the target address is relative to the PC.



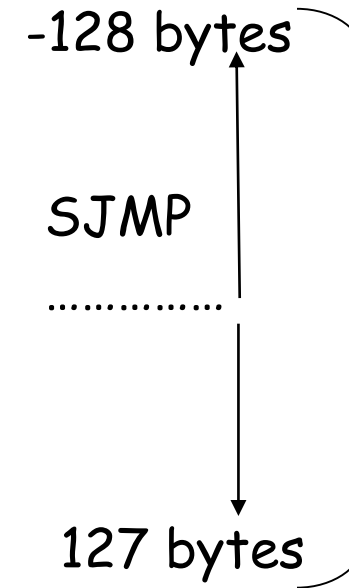


# SJMP - more

- ❑ Jump to a new address  
(8 bit relative-address is used)

| Line | PC   | Opcode | Mnemonic   | Operand |
|------|------|--------|------------|---------|
| 17   | 0015 | 80FE   | HERE: SJMP | HERE    |
| 18   | 0017 |        | END        |         |

- Then opcode of “SJMP” is 80.
- FE is the relative offset
- ❑ A jump can be performed within the -128 to +127 bytes from the address of the following instruction.
- ❑ All conditional jumps are short jumps



# Relative Address

- ❑ The target address must be within **-128 to +127** bytes of the address of the following instruction.
  - Forward jump: 0 ~ 127 (0 ~ 7FH)
  - Backward jump: -1 ~ -128 (FFH ~ 80H)
- ❑ The 8051 Assembler changes the target label into the relative offset to PC and saves the offset in the instructions.
- ❑ Calculation of the relative offset (by assembler)
  - **Relative offset = Target address - PC**
- ❑ Calculation of the target address (by 8051):
  - **Real target address = PC + relative address**
    - **Ex1**: PC=1001H, relative address=40H  
⇒ target address = 1001H+40H=1041H
    - **Ex2**: PC=1001H, relative address=FEH (-2<sub>10</sub>)  
⇒ target address = 1001H+**FF**FEH=0FFFH  
(for backward jump, take the high byte as FF)

# Example

- Using the following list file, verify the **jump forward and backward** address calculation.

| <i>Line</i> | <i>PC</i> | <i>Opcode</i> |        | <i>Mnemonic</i> | <i>Operand</i> |
|-------------|-----------|---------------|--------|-----------------|----------------|
| 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             |                |

# AJMP (Absolute Jump)

- AJMP has the same use and feel as ACALL

|      |         |                                    |   |    |
|------|---------|------------------------------------|---|----|
|      |         |                                    |   |    |
| AJMP | addr11  | Absolute Jump                      | 2 | 24 |
| LJMP | addr16  | Long Jump                          | 3 | 24 |
| SJMP | rel     | Short Jump (relative addr)         | 2 | 24 |
| JMP  | @A+DPTR | Jump indirect relative to the DPTR | 1 | 24 |

# Exercise

- Write a portion of a program to check whether the read byte from P1 is one of the prime numbers strictly less than 100.

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10  |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20  |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30  |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40  |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50  |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60  |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70  |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80  |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90  |
| 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 |

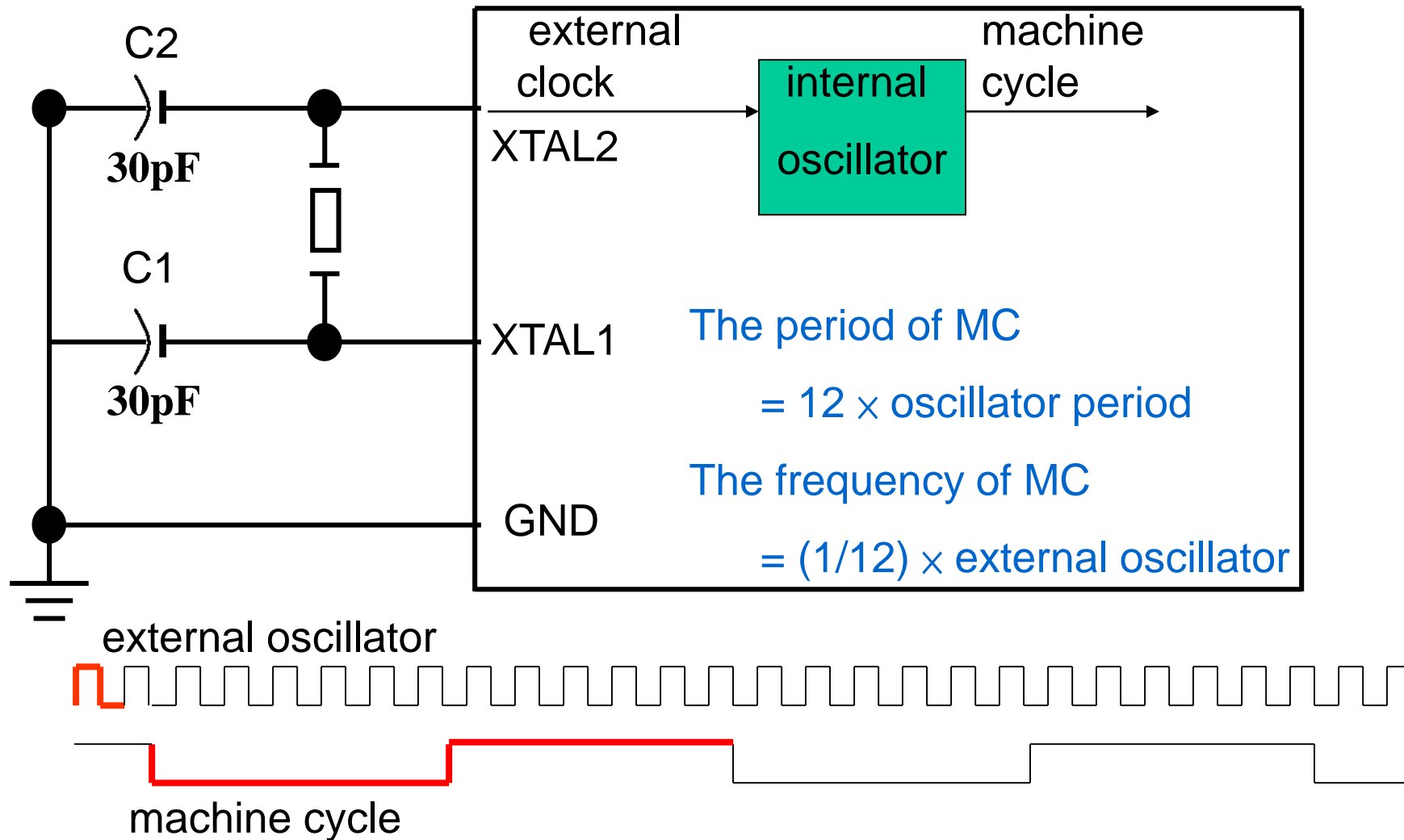
# Machine Cycle

- ❑ In the 8051 family, the clock cycles provided by the external clock make up the *machine cycles*.
- ❑ To execute an instruction takes a certain number of machine cycles.
  - Ex: RET needs 2 machine cycles
- ❑ The 8051 has an on-chip oscillator which generates machine cycles.
- ❑ The 8051 requires an external clock, i.e., a quartz crystal oscillator, to run the on-chip oscillator

# Machine Cycle

- ❑ The relationship between two oscillators
  - The length of the machine cycle is 12 of the oscillator period.
  - The frequency of the machine cycle is 1/12 of the crystal frequency.
- ❑ The frequency of the external crystal can vary from 4 MHz to 30MHz.
- ❑ Very often the 11.0592MHz crystal oscillator is used to make the 8051-based system compatible with the serial port of the IBM PC. See Chapter 10.

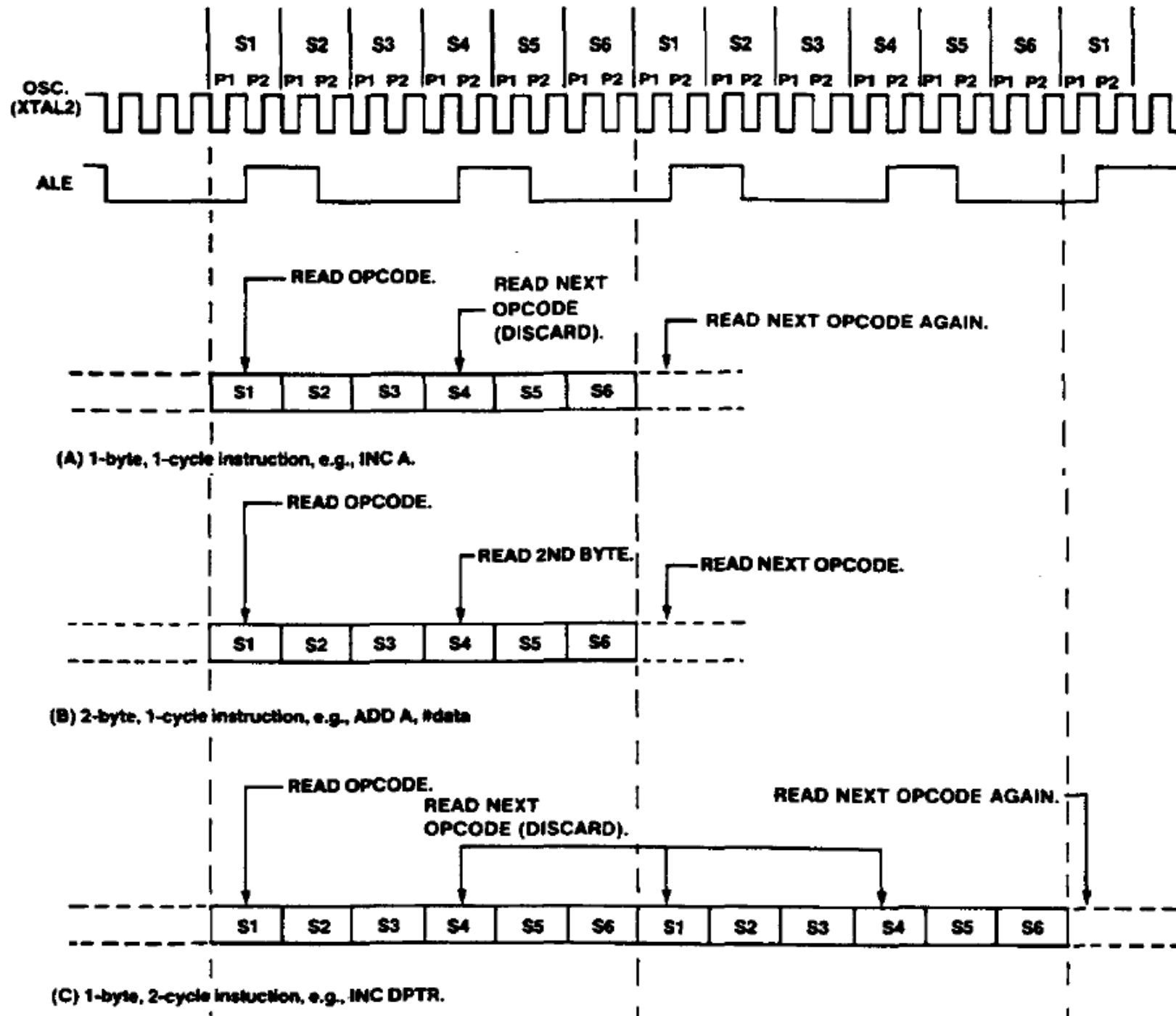
# The 8051 Oscillators





# States

- ❑ A machine cycle consists of a sequence of 6 states numbered S1 through S6.
- ❑ Each state time lasts for two oscillator periods. Thus a machine cycle takes 12 oscillator periods (e.g., 1  $\mu$ s if the oscillator frequency is 12 MHz).
- ❑ Each state is divided into a Phase 1 half and a Phase 2 half.



# Example

- The following shows crystal frequency for three different 8051- based systems. Find the period of the machine cycle in each case.

(a) 11.0592 MHz    (b) 16 MHz    (c) 20 MHz

## Solution:

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

Machine cycle is  $1/921.6 \text{ KHz} = 1.085\mu\text{s}$  (microsecond)

(b) Oscillator period =  $1/16 \text{ MHz} = 0.0625 \mu\text{s}$

Machine cycle (**MC**) =  $0.0625 \mu\text{s} \times 12 = 0.75 \mu\text{s}$

(c)  $20 \text{ MHz}/12 = 1.66 \text{ MHz}$

MC =  $1/1.66 \text{ MHz} = 0.60 \mu\text{s}$

# Example

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

```
 MOV A,#55H
AGAIN: MOV P1,A
 ACALL DELAY
 CPL A
 SJMP AGAIN
```

|        | <i>Time delay</i> | <i>Machine Cycle</i> |
|--------|-------------------|----------------------|
| DELAY: | MOV R3,#200       | ;1                   |
| HERE:  | DJNZ R3,HERE      | ;2                   |
|        | RET               | ;2                   |

**Solution:** The time delay is

$$[(200 \times 2) + 1 + 2] \times 1.085 \mu\text{s} = 437.252 \mu\text{s}.$$

- Note different systems may have different machine cycles per instruction; above is true for the original 8051

# Example

Find the time delay for the following subroutine, assuming a crystal frequency of 11.0592 MHz.

```
 ;Machine Cycle
DELAY: MOV R3,#250 ;1
HERE: NOP ;1
 NOP ;1
 NOP ;1
 NOP ;1
 DJNZ R3,HERE ;2
 RET ;2
```

## Solution:

the HERE loop and the two instructions outside the loop

$$\begin{aligned} & \{ [250 (1+1+1+1+2)] + 3 \} \times 1.085 \mu\text{s} \\ & = (1500+3) \times 1.085 \mu\text{s} = 1630.755 \mu\text{s}. \end{aligned}$$

# Nested Loop Example

```
 MOV A, #55H ; 1 x 1 MC
 MOV R3, #10 ; 1 x 1 MC
NEXT: MOV R2, #70 ; 10 x 1 MC
AGAIN: CPL A ; (10 x 70) x 1 MC
 DJNZ R2, AGAIN ; (10 x 70) x 2 MC
 DJNZ R3, NEXT ; 10 x 2 MC
```

$$1 \times (1 + 1) + 10 \times (1 + 2 + 70 \times (1 + 2)) = 2132 \text{ MC}$$

$$2132 \times 1.085 = 2313 \mu\text{s}$$

# Delay calculation for other 8051 versions

- ❑ Two factors for time delay
  - The crystal frequency
  - The 8051 design
- ❑ Advances in both IC technology and CPU design
- ❑ **The number of clock periods per machine cycle** varies among different versions of the 8051 ICs.

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

# More

- ❑ Considering the original 8051, the fastest instructions require one machine cycle (12 oscillator cycles), many others require two machine cycles (24 oscillator cycles), and the two very slow math operations require four machine cycles (48 oscillator cycles).
- ❑ Many 8051 derivative chips change instruction timing. Many optimized versions of the 8051 execute instructions in 4 oscillator cycles instead of 12; such a chip would be effectively 3 times faster than the standard 8051.
- ❑ How can one keep track of time in a time-critical application if we have no reference to time in the outside world?
- ❑ Luckily, the 8051 includes **timers** which allow us to time events with high precision



## Comparison of 8051 and DS89C4x0 machine cycles

| <b>Instruction</b> | <b>8051</b> | <b>DS89C4x0</b> |
|--------------------|-------------|-----------------|
| MOV R3,#value      | 1           | 2               |
| DEC Rx             | 1           | 1               |
| DJNZ               | 2           | 4               |
| LJMP               | 2           | 3               |
| SJMP               | 2           | 3               |
| NOP                | 1           | 1               |
| MUL AB             | 4           | 9               |

[http://www.keil.com/dd/docs/datashts/dallas/ds89c420\\_ug.pdf](http://www.keil.com/dd/docs/datashts/dallas/ds89c420_ug.pdf)

# Example

- Find the time delay for the following subroutine if it run on a DS89C420 chip, assuming a crystal frequency of 11.0592 MHz.

## *DS89C420 Machine Cycle*

```
DELAY: MOV R3,#250

HERE: NOP ;1
 NOP ;1
 NOP ;1
 NOP ;1
 DJNZ R3,HERE ;4

 RET
```

### **Solution:**

The time delay inside the HERE loop is

$$[250 (1+1+1+1+4)] \times 90 \text{ ns} = 2000 \times 90 \text{ ns} = 180 \mu\text{s}$$

Compare AT89C51 with DS89C420:  $1627 \mu\text{s} / 180 \mu\text{s} = 9$

# Example

Write a program to toggle all the bits of P1 every 200ms for DS89C4x20. Assume that the crystal frequency is 11.0592 MHz.

**Solution:**  $200\text{ms}/90\text{ns} = 2222 \times 10^3 \text{ MCs}$

$$2222 \times 10^3 / 4 \sim 555390 = 9 \times 242 \times 255$$

```
 MOV A, #55H
AGAIN: MOV P1, A
 ACALL DELAY
 CPL A
 SJMP AGAIN
DELAY: MOV R5, #9 ;2
HERE1: MOV R4, #242 ;2
HERE2: MOV R3, #255 ;2
HERE3: DJNZ R3, HERE3 ;4
 DJNZ R4, HERE2 ;4
 DJNZ R5, HERE1 ;4
 RET
```

$$9 \times 242 \times 255 \times 4\text{MC} \times 90 \text{ ns} \sim 200\text{ms}$$

# Why will we have to use timers?

- ❑ The use of the instructions in generating time delay is **not** the most **reliable** method.
- ❑ Software based delay subroutines:
  - Lack of code portability
  - Inefficiency (wait until we return from the subroutine, lack of responsiveness)
- ❑ To tackle these two concerns, we use timers and interrupts.

# Addressing Modes

- ❑ The CPU can access data in various ways.
- ❑ The data could be in a register, or in memory; RAM or ROM, or be provided as an immediate value.
- ❑ These various ways of accessing data are called *addressing modes*.
- ❑ Five addressing modes in the 8051
  1. immediate
  2. register
  3. direct
  4. register indirect
  5. indexed
- ❑ We use MOV as an example. One can use any instruction as long as that instruction supports the addressing mode.

# Addressing modes

1. immediate: the operand is a constant

MOV A,#1FH

2. register: — the operand is in a register

MOV A,R0

3. direct: access the data in the RAM with address

MOV A,1FH

4. register indirect: — the register holds the RAM address of the data

MOV A,@R0

5. indexed: — for on-chip ROM access

MOVC A,@A+DPTR

and external ROM/RAM access

MOVX A,@DPTR

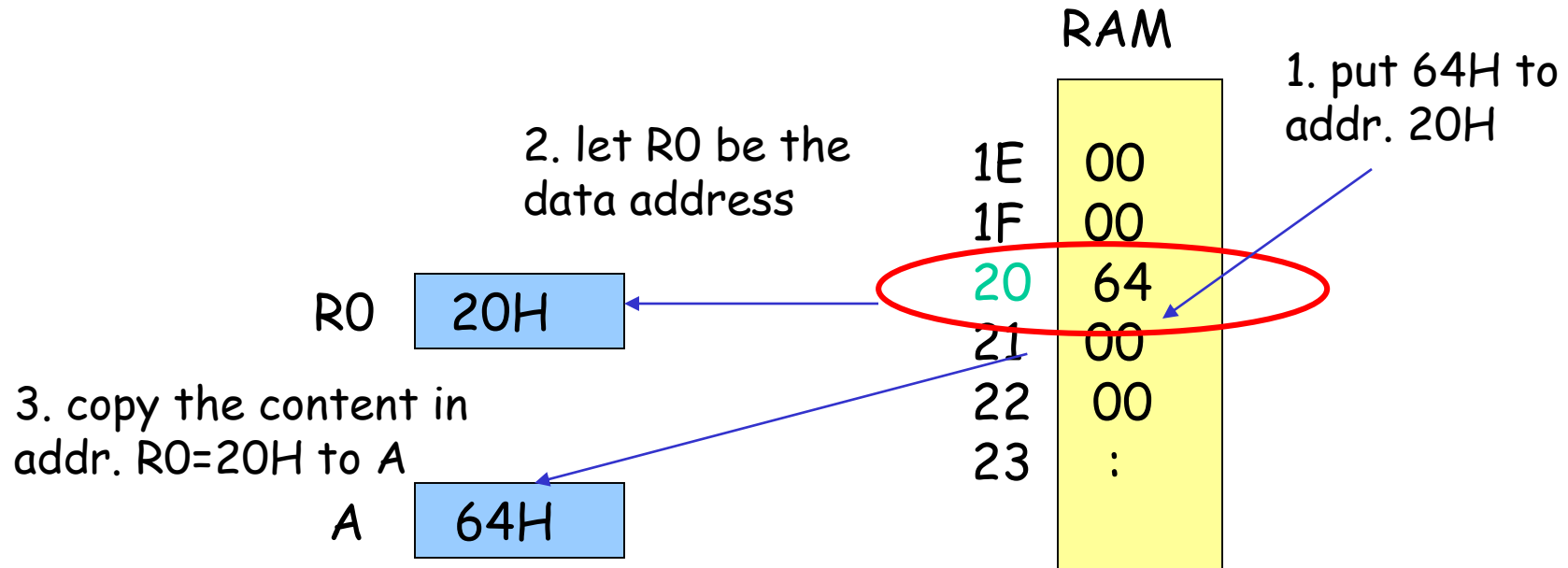
# Register indirect addressing mode

- ❑ In the register indirect addressing mode, a register is used as a **pointer** to the data.
  - That is, this register **holds the RAM address of the data**.
- ❑ Only registers **R0 and R1** can be used to hold the address of an operand located in RAM.
  - Usually, R0 and R1 are denoted by **Ri**.
- ❑ When R0 and R1 hold the addresses of RAM locations, they must be **preceded by the “@” sign**.

# Example

## □ Register Indirect Mode

```
1 0000 75 20 64 MOV 20H,#100
2 0003 78 20 MOV R0,#20H
3 0005 E6 MOV A,@R0
```





# Example

Write a program to copy the address itself into the RAM locations 40H to 4FH using

- (a) without a loop,
- (b) with a loop.

# Exercise

- Write a subroutine to return the minimum (or maximum) of 16 unsigned (or signed) 8 bit numbers located beginning at internal RAM location 50h.

# 8052 – Extra 128 Byte RAM

- ❑ 8052 has 256-bytes of RAM.
  - Lower 128 bytes: addresses 00-7FH
  - Upper 128 bytes: addresses 80-FFH
- ❑ However, 80-FFH has been used by SFRs.
  - SFRs are accessed by direct addressing mode
  - To send 55H to P2: `MOV 90H, #55H`
- ❑ This parallel address space in the 8052 forces us to use two different addressing modes to access them.

# How to distinguish them

□ We use different addressing modes to access them:

1. To access the **SFRs**, we use direct addressing mode.

- Ex: `MOV 90H, #55H`

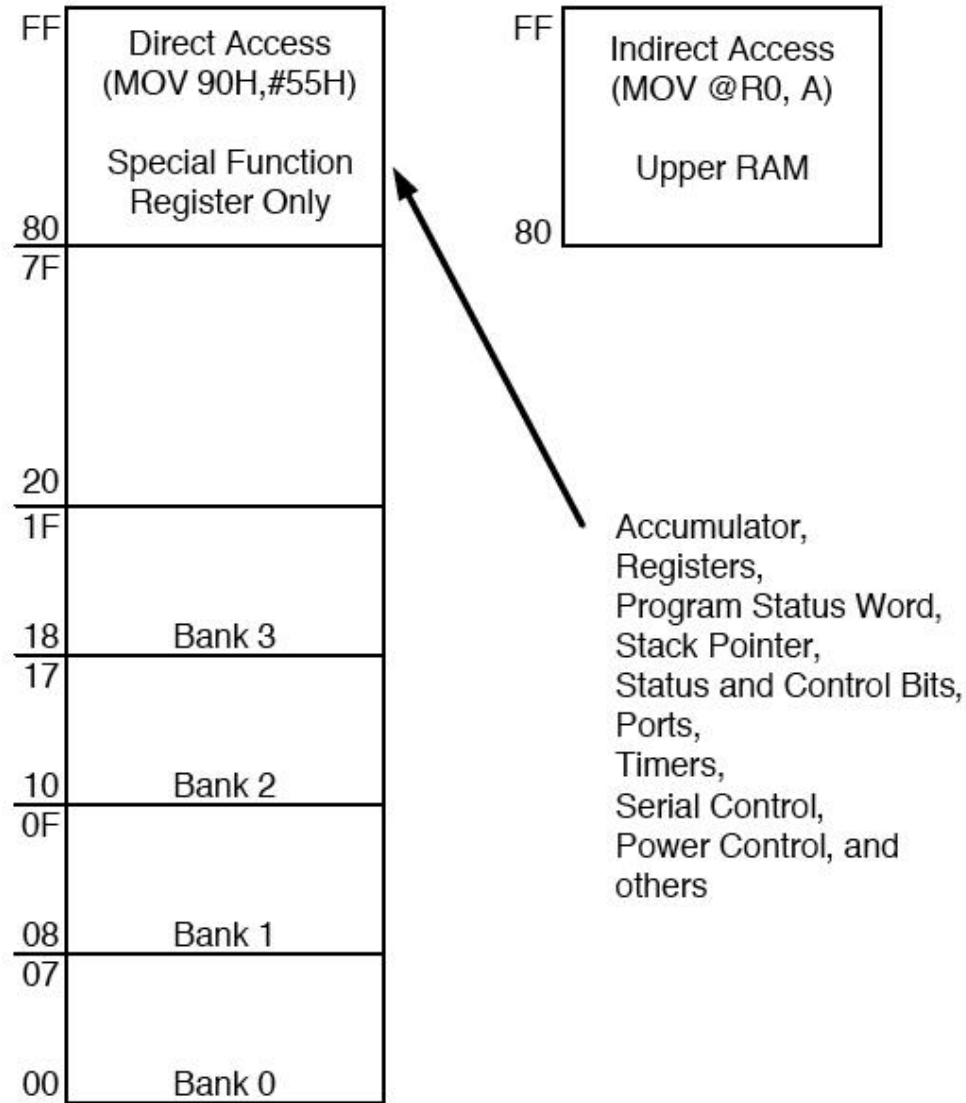
2. To access the **upper 128 bytes**, we use the **indirect addressing mode**, which uses R0 and R1 as pointers.

- Ex: `MOV R0, #90H`

- `MOV @R0, #55H`

- R0 and R1 have address values of 80H or higher.

# 8052 On-chip RAM address space



Assume that the on-chip ROM has a message. Write a program to copy it from code space into the upper memory space starting at address 80H excluding the end of string (EoS) character, 0.

Also, as you place a byte in upper RAM, give a copy to P0.

```

 ORG 0000
 MOV DPTR, #MYDATA
 MOV R1, #80H ;upper 128-byte RAM
B1: CLR A
 MOVC A, @A+DPTR ;read from code
 JZ EXIT
 MOV @R1, A ;copy to upper RAM
 MOV P0, A ;give a copy to P0
 INC DPTR
 INC R1
 SJMP B1
EXIT: SJMP $
 ORG 300H
MYDATA: DB "The promise of World Peace", 0
 END
```

# Features Specific to Atmel AT 89S52

Table 5-1. AT89S52 SFR Map and Reset Values

|      |                   |                   |                    |                    |                  |                  |                    |      |
|------|-------------------|-------------------|--------------------|--------------------|------------------|------------------|--------------------|------|
| 0F8H |                   |                   |                    |                    |                  |                  |                    | 0FFH |
| 0F0H | B<br>00000000     |                   |                    |                    |                  |                  |                    | 0F7H |
| 0E8H |                   |                   |                    |                    |                  |                  |                    | 0EFH |
| 0E0H | ACC<br>00000000   |                   |                    |                    |                  |                  |                    | 0E7H |
| 0D8H |                   |                   |                    |                    |                  |                  |                    | 0DFH |
| 0D0H | PSW<br>00000000   |                   |                    |                    |                  |                  |                    | 0D7H |
| 0C8H | T2CON<br>00000000 | T2MOD<br>XXXXXX00 | RCAP2L<br>00000000 | RCAP2H<br>00000000 | TL2<br>00000000  | TH2<br>00000000  |                    | 0CFH |
| 0C0H |                   |                   |                    |                    |                  |                  |                    | 0C7H |
| 0B8H | IP<br>XX000000    |                   |                    |                    |                  |                  |                    | 0BFH |
| 0B0H | P3<br>11111111    |                   |                    |                    |                  |                  |                    | 0B7H |
| 0A8H | IE<br>0X000000    |                   |                    |                    |                  |                  |                    | 0AFH |
| 0A0H | P2<br>11111111    |                   | AUXR1<br>XXXXXXX0  |                    |                  |                  | WDTRST<br>XXXXXXXX | 0A7H |
| 98H  | SCON<br>00000000  | SBUF<br>XXXXXXXX  |                    |                    |                  |                  |                    | 9FH  |
| 90H  | P1<br>11111111    |                   |                    |                    |                  |                  |                    | 97H  |
| 88H  | TCON<br>00000000  | TMOD<br>00000000  | TL0<br>00000000    | TL1<br>00000000    | TH0<br>00000000  | TH1<br>00000000  | AUXR<br>XXXXXX00   | 8FH  |
| 80H  | P0<br>11111111    | SP<br>00000111    | DP0L<br>00000000   | DP0H<br>00000000   | DP1L<br>00000000 | DP1H<br>00000000 |                    | 87H  |
|      |                   |                   |                    |                    |                  |                  | PCON<br>0XXX0000   |      |

# Dual Data Pointers

**Dual Data Pointer Registers:** To facilitate accessing both internal and external data memory, two banks of 16-bit Data Pointer Registers are provided: DP0 at SFR address locations 82H-83H and DP1 at 84H-85H. Bit DPS = 0 in SFR AUXR1 selects DP0 and DPS = 1 selects DP1. The user should **ALWAYS** initialize the DPS bit to the appropriate value before accessing the respective Data Pointer Register.

**Table 5-4.      AUXR1: Auxiliary Register 1**

AUXR1

Address = A2H

Reset Value = XXXXXX0B

Not Bit Addressable

|     |   |   |   |   |   |   |     |
|-----|---|---|---|---|---|---|-----|
|     | – | – | – | – | – | – | DPS |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 0   |

–

Reserved for future expansion

DPS

Data Pointer Register Select

DPS

0

Selects DPTR Registers DP0L, DP0H

1

Selects DPTR Registers DP1L, DP1H



# Exercise

- Write a subroutine to read a 4K block from the internal ROM located at 1000h and write it into a block beginning at 3800h in the external RAM.

# Exam Question

**Q1) [25 points]** Write a subroutine for the Atmel AT89S52 with the name SIN that will read an integer degree  $\theta$  from P0:P1 ( $1 \leq \theta \leq 89$  given) in ASCII and return the five fractional digits of  $\sin(\theta)$  in the five 8052 upper memory locations 80h-84h. For example, let P0 read 34h and P1 read 32h, then  $\theta = 42$ ,  $\sin(42) = 0.66913$  and the memory locations 80h-84h should hold '6', '6', '9', '1' and '3', respectively, when the subroutine SIN is to be returned. The accumulator needs to be zero when using `MOVC A,@A+DPTR`. Use ONLY the cells below and one instruction per cell, accompanied by your comments when needed. Do not need to check whether the P0:P1 reading is a legitimate degree reading in the given interval; assume so.

|           |              |                                 |
|-----------|--------------|---------------------------------|
| SINTABLE: | DB 0,0,0,0,0 | ;5 extra bytes to simplify code |
|           | DB '01745'   | ; sin(1)=0.01745                |
|           | DB '03489'   | ; sin(2)=0.03489                |
|           | ...          | ; ...                           |
|           | DB '99985'   | ;sin(89)=0.99985                |

# Solution

|        |                    |                           |
|--------|--------------------|---------------------------|
| SIN:   | MOV DPTR,#SINTABLE | ;use comments             |
|        | MOV A,P0           | ;A=34h                    |
|        | ANL A,#0Fh         | ;A=04h                    |
|        | MOV B,#10          |                           |
|        | MUL AB             | ;A=40                     |
|        | MOV 30h,P1         |                           |
|        | ANL 30h,#0Fh       |                           |
|        | ADD A,30h          | ;A=42                     |
| BACKA: | MOV R5,#5          |                           |
| BACK5: | INC DPTR           |                           |
|        | DJNZ R5,BACK5      |                           |
|        | DJNZ ACC,BACKA     | ;Dptr is incemented by 50 |
|        | MOV R0,#80h        |                           |
|        | MOV R5,#5          |                           |
| BACK:  | CLR A              |                           |
|        | MOVC A,@A+DPTR     |                           |
|        | MOV @R0,A          |                           |
|        | INC DPTR           |                           |
|        | INC R0             |                           |
|        | DJNZ R5,BACK       |                           |
|        |                    |                           |
| EXIT:  | RET                |                           |

# MCU 8051 IDE

