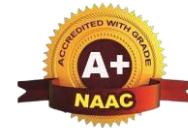


BEE403:MICROCONTROLLERS

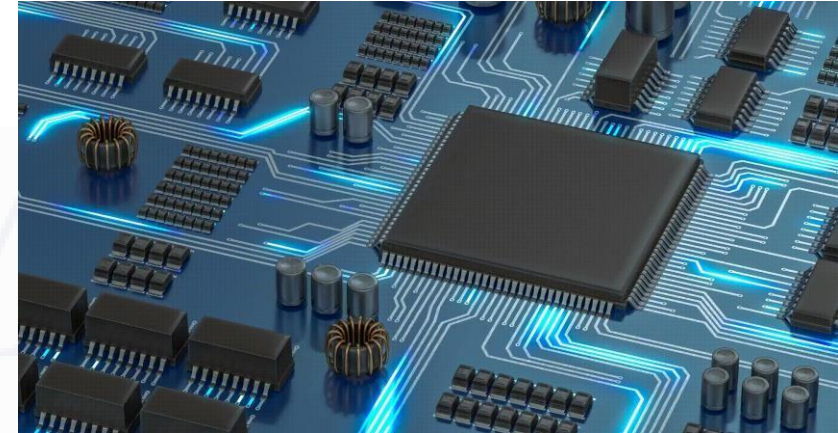
MODULE – 2:Assembly programming and Instructions of 8051



OUTLINE

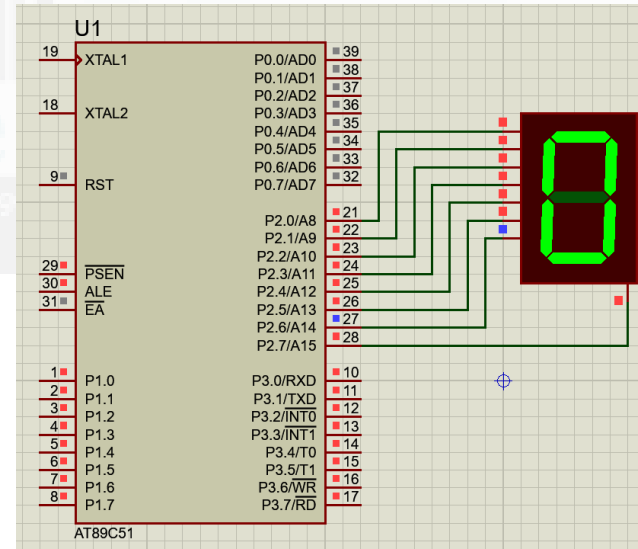


- Introduction to 8051 Assembly Programming
- Assembling and running an 8051 program
- Data types and Assembler directives
- Arithmetic instructions and programs
- logic instructions and programs
- Jump instructions, loop and Call instructions
- IO port programming, Programs
- Additional Programs on Jump, loop ,call and Port programming
- Additional Programs on Arithmetic Instructions





Introduction to 8051 Assembly Programming



What is a Programming Language?

Programming in the sense of Microcontrollers (or any computer) means writing a **sequence of instructions** that are executed by the processor in a particular order to **perform a predefined task**.

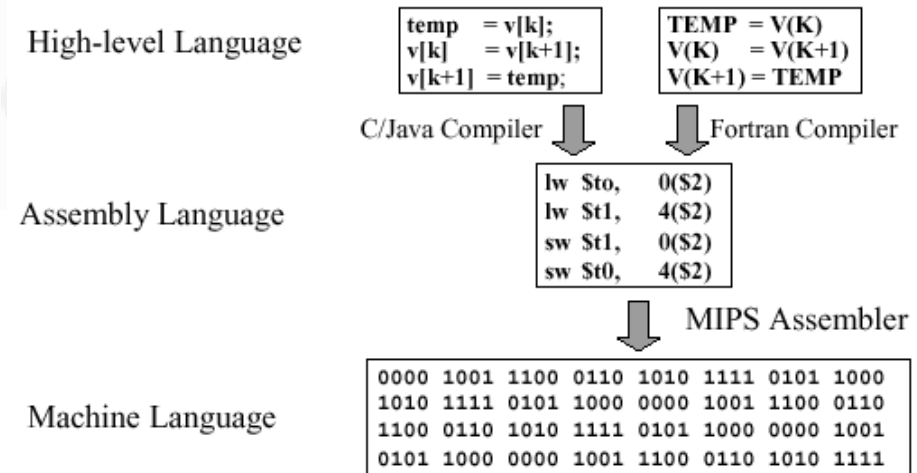
Programming also involves **debugging and troubleshooting** of instructions and instruction sequence to make sure that the desired task is performed.



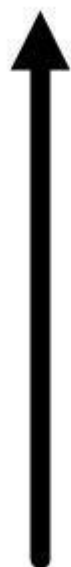
There are three types or levels of Programming Languages for 8051 Microcontroller.

• The three levels of Programming Languages are:

1. Machine Language
2. Assembly Language
3. High-level Language



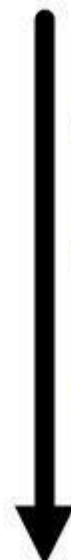
- Ease of Implementation
- Flexibility
- Portability



High-level
Languages

Assembly
Language

Machine
Language



- Speed of Execution
- Code Density
- Machine Specific

ELECTRONICS BUS

Machine language

- In Machine language or Machine Code, the instructions are written in binary bit patterns i.e. combination of binary digits 1 and 0, which are stored as HIGH and LOW Voltage Levels.
- This is the lowest level of programming languages and is the language that a Microcontroller or Microprocessor actually understands.

High Level language

- The name High-level language means that you need not worry about the **architecture** or other internal details of a microcontroller and they use words and statements that are easily understood by humans.
- Few examples of High-level Languages are **BASIC, C Pascal, C++ and Java.**
- A program called **Compiler** will convert the Programs written in **High-level languages to Machine Code.**

Assembly language

- Since Machine Language or Code involves all the instructions in 1's and 0's, it is very difficult for humans to program using it.
- Assembly Language is a **pseudo-English representation of the Machine Language**. The 8051 Microcontroller Assembly Language is a combination of English like words called **Mnemonics and Hexadecimal codes**.
- It is also a **low level language** and requires extensive understanding of the architecture of the Microcontroller.

Why Assembly language??

- Although High-level languages are easy to work with, the following reasons point out the advantage of Assembly Language
 1. The Programs written in **Assembly gets executed faster and they occupy less memory.**
 2. With the help of Assembly Language, you can directly exploit all the **features of a Microcontroller.**
 3. Using Assembly Language, you can have direct and accurate control of all the Microcontroller's resources like **I/O Ports, RAM, SFRs, etc.**
 4. Compared to High-level Languages, Assembly Language has less **rules and restrictions.**

Introduction to 8051 Programming in Assembly Language

- Even though there are many high-level languages that are currently in demand, assembly programming language is popularly used in many applications.
- It can be used for direct hardware manipulations.
- It is also used to write the **8051 programming code** efficiently with less number of clock cycles by consuming less memory compared to the other high-level languages.

Introduction to 8051 Programming in Assembly Language

```
ORG 0000H  
MOV P0, #00H  
DELAY  
MOV P0, #FFH  
DELAY
```

**Assembly
Code**

ASSEMBLER

```
0100001001001000  
0001000001010111  
1101000000101111  
1000000100100101  
0010001000000001
```

**Objective
Code**

Introduction to 8051 Programming in Assembly Language

- Assembly programming language is developed by various compilers and the “**keiluvision**” is best suitable for microcontroller programming development.
- An assembler converts the assembly language to binary language, and then stores it in the microcontroller memory to perform the specific task.

ASSEMBLING AND RUNNING AN 8051 PROGRAM

Steps to Create a Program

STEP 1: First we use an editor to type in a program similar to Program A widely used editor is the MS-DOS EDIT program (or Notepad in Windows), which comes with all Microsoft operating systems. Notice that the editor must be able to produce an ASCII file.

STEP 2: The “asm” extension for the source file is used by an assembler in the next step. The “asm” source file containing the program code created in step 1 is fed to an 8051 assembler machine code.

The assembler will produce an object file and a list file. The extension for the object file is “.obj” while the extension for the list file is “.lst”.

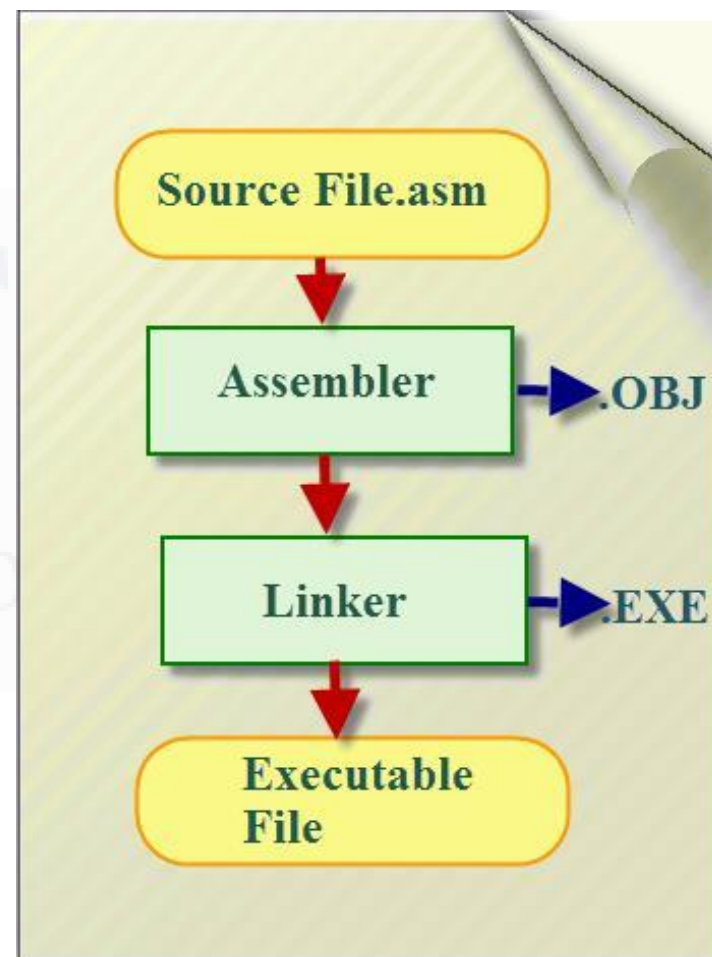
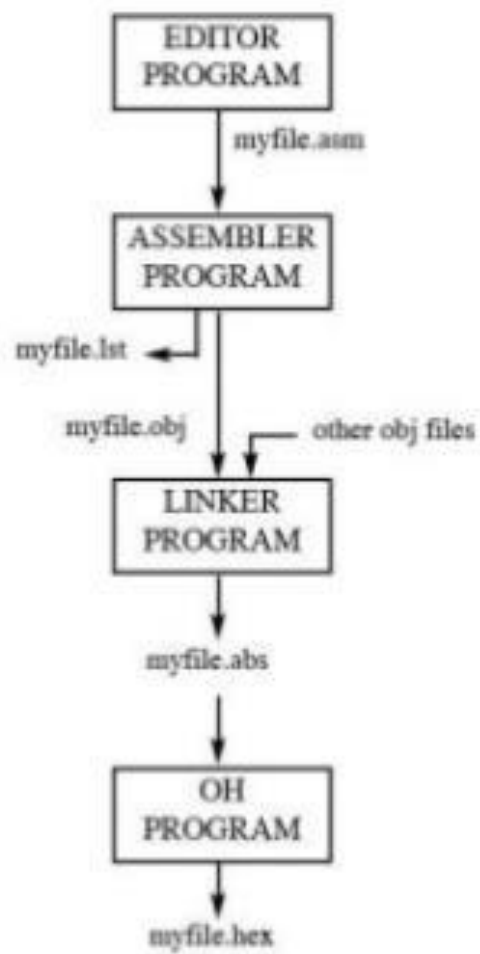
ASSEMBLING AND RUNNING AN 8051 PROGRAM

Steps to Create a Program

STEP 3: Assemblers require a third step called linking. The link program takes one or more object files and produces an absolute object file with the extension “abs”

This abs file is used by 8051 trainers that have a monitor program.

STEP 4: Next, the “abs” file is fed into a program called “OH” (object to hex converter), which creates a file with extension “hex” that is ready to burn into ROM. This program comes with all 8051 assemblers.



Assembly Programme Example

File is saved with extension .A51

Title Section

label

directive

```
C:\Niall\systems architecture\code\EX1.A51
;*****
;Niall O'Keeffe
;8/9/05
;Introduction to assembly language programming
;*****
MAIN:  ORG OH
        MOV A, #10
        MOV R0, #5
        ADD A, R0
        JMP MAIN      ;jump to start of program
        END
```

Code
comment

8051 DATA TYPES AND DIRECTIVES



8051 data type and directives

1. The 8051 microcontroller has only **one data type**.
2. It is 8 bits, and the size of each register is also **8 bits**.
3. It is the job of the programmer to break down data larger than 8 bits (00 to FFH, or 0 to 255 in decimal) to be processed by the CPU.
4. The data types used by the 8051 can be **positive or negative**



1. DB (define byte)

1. The DB directive is the most widely used **data directive in the assembler.**
2. It is used to define the 8-bit data
3. When DB is used to define data, the numbers can be in **decimal, binary, hex, or ASCII formats**
4. For decimal, the “D” after the decimal number is optional, but using “B” (binary) and “H” (hexadecimal) for the others is required.
5. Regardless of which is used, the assembler will convert the numbers **into hex.**

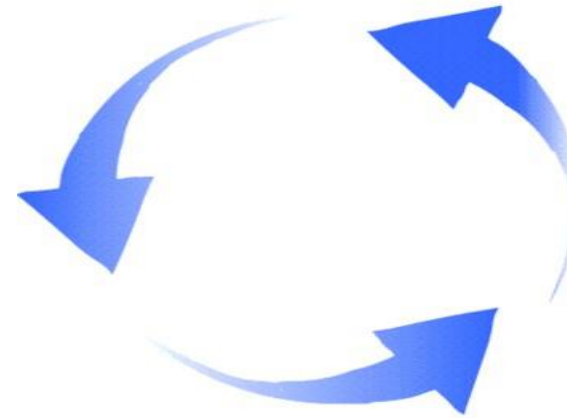
DB Example



DB (define byte)

```
                ORG 500H
DATA1:          DB 28                ;DECIMAL(1C in hex)
DATA2:          DB 00110101B         ;BINARY (35 in hex)
DATA3:          DB 39H               ;HEX
                ORG 510H
DATA4:          DB "2591"            ;ASCII NUMBERS
                ORG 518H
DATA6:          DB "My name is Joe"  ;ASCII CHARACTERS
```

ORG (origin)

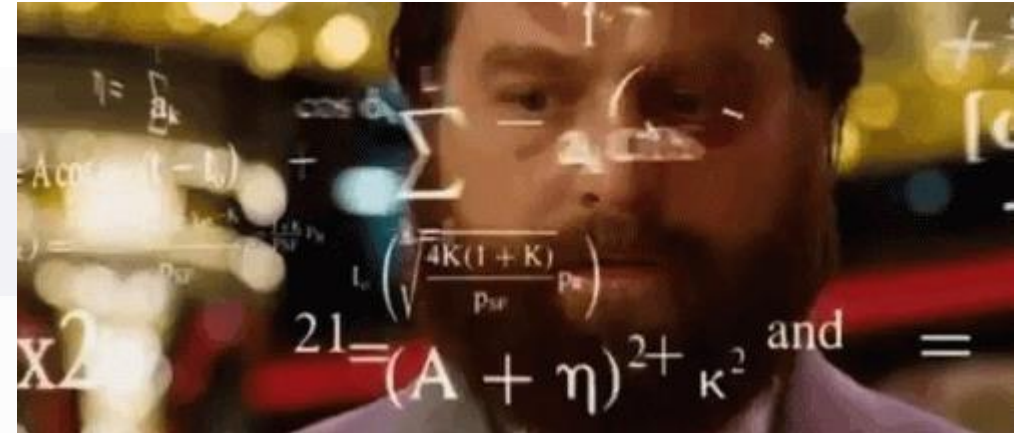


1. The ORG directive is used to indicate the **beginning of the address.**
2. The number that comes after ORG can be either in **hex or in decimal.**
If the number is not followed by H, it is decimal and the **assembler will convert it to hex.**
3. Some assemblers use “. ORG” (notice the dot) instead of “ORG” for the origin directive.

NOTE: Check your assembler.

Example

```
ORG 0000H
```



EQU (equate)

1. This is used to define a constant without **occupying a memory location**.
2. The EQU directive does not set aside storage for a data item but associates **a constant value** with a data label so that when the label appears in the program, its constant value will be substituted for the label.
3. The following uses EQU for the counter constant and then the constant is used to load the R3 register.

EQU Examp COUNT EQU 25
 . . .

 MOV R3 , #COUNT

When executing the instruction
“MOV R3, #COUNT”, the register R3 will be loaded with the value 25 (notice the # sign).

What is the advantage of using EQU?

By the use of EQU, the programmer can change it once and the assembler will change* all of its occurrences, rather than search the **entire program trying to find every occurrence.**



A T M E
College of Engineering



END directive

1. Another important pseudocode is the END directive. This indicates to the assembler the end of the source (asm) file.
2. The END directive is the last line of an 8051 program, meaning that in the source code anything after the END directive is ignored by the assembler.
3. Some assemblers use “. END” (notice the dot) instead of “END”.

QUIZ

1. Which is not an assembler directive?

- END
- ORG
- EQU
- INT

2. ORG directive is used to indicate?

- End of the address
- Beginning of the Address
- Address Loop
- Constant

3. DB is used to define data, the numbers can be in ?

- DECIMAL
- BINARY
- HEX
- ALL THE ABOVE

8051 Instructions The instructions of 8051 can be broadly classified under the following headings.

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Bit manipulation instructions

1. Data transfer instructions

In this group, the instructions perform data transfer operations of the following types.

a. Move the contents of a register Rn to A

i. MOV A,R2

ii. MOV A,R7

b. Move the contents of a register A to Rn

i. MOV R4,A

ii. MOV R1,A

1. Data transfer instructions

c. Move an immediate 8 bit data to register A or to Rn or to a memory location (direct or indirect)

- i. MOV A, #45H
- ii. MOV R6, #51H
- iii. MOV 30H, #44H
- iv. MOV @R0, #0E8H
- v. MOV DPTR, #F5A2H
- vi. MOV DPTR, #5467H

1. Data transfer instructions

d. Move the contents of a memory location to A or A to a memory location using direct and indirect addressing

- i. MOV A, 65H
- ii. MOV A, @R0
- iii. MOV 45H, A
- iv. MOV @R1, A

e. Move the contents of a memory location to Rn or Rn to a memory location using direct addressing

- i. MOV R3, 65H
- ii. MOV 45H, R2

1. Data transfer instructions

f. Move the contents of memory location to another memory location using direct and indirect addressing

MOV R0, 45H

MOV 54H, @R0

@45H	54H
F6H	F6H

g. Move the contents of an external memory to A or A to an external memory

i. MOVX A, @R1

ii. MOVX @R0,A

iii. MOVX A,@DPTR

iv. MOVX@DPTR,A



A T M E

College of Engineering

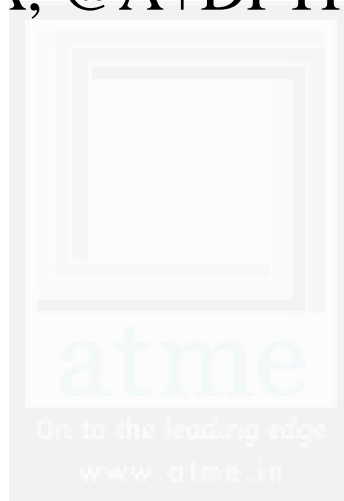


1. Data transfer instructions

h. Move the contents of program memory to A

i. `MOVCA, @A+PC`

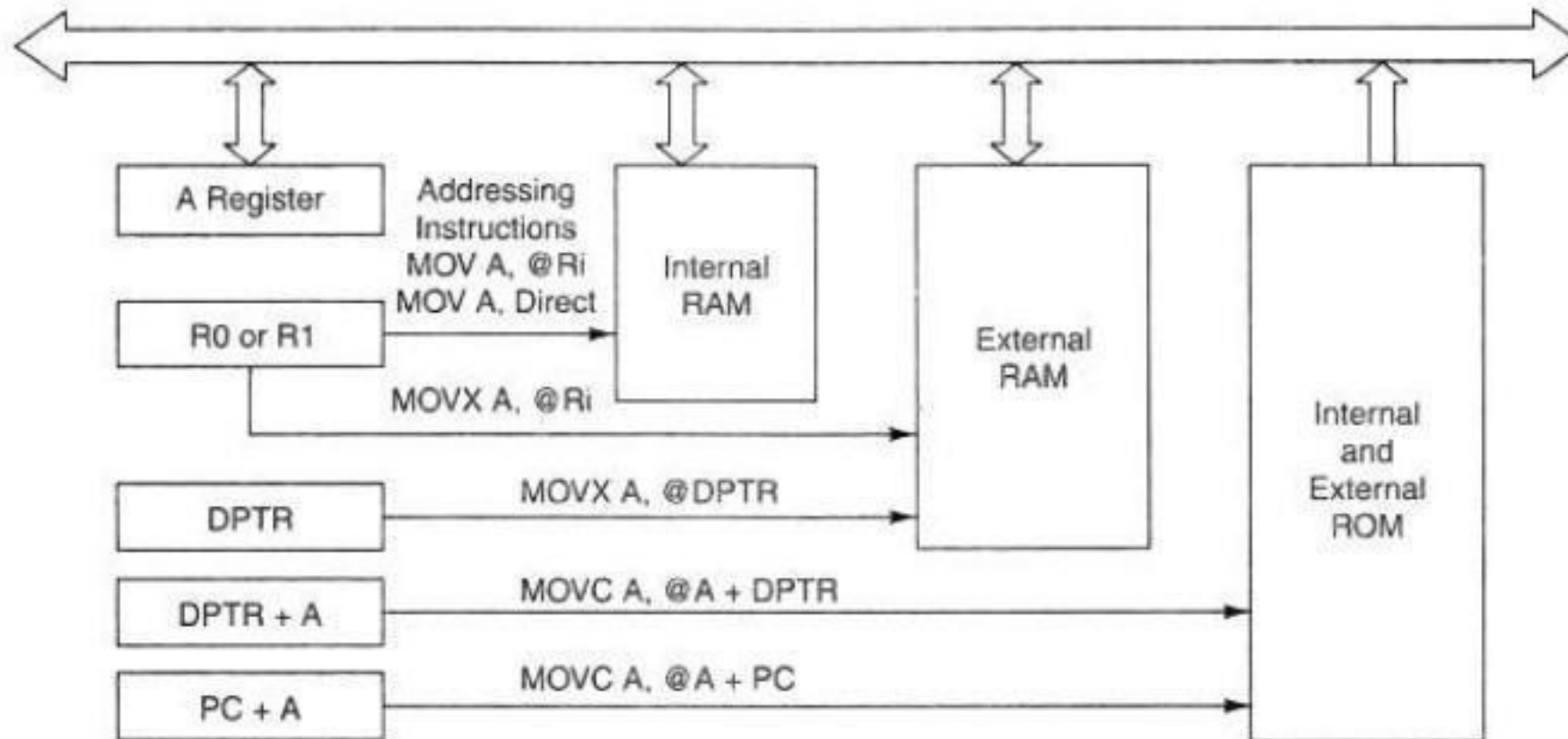
ii. `MOVCA, @A+DPTR`



A T M E

College of Engineering

1. Data transfer instructions



The 8051 can perform addition, subtraction. Multiplication and division operations on 8 bit numbers:

a) Addition:

In this group, we have instructions to

i. Add the contents of A with immediate data with or without carry.

i. ADD A, #45H ; A= 0AH + 45H → A=

ii. ADDC A, #B4H ; CY=01 + A= +B4H=

ii. Add the contents of A with register Rn with or without carry.

Example:

MOV R5, #45H

MOV A, #05H

ADD A, R5

iii. Add the contents of A with contents of memory with or without carry using direct and indirect addressing

- i. **ADD A, 51H**
- ii. **ADDC A, 75H**
- iii. **ADD A, @R1**
- iv. **ADDC A, @R0**

CYAC and OV flags will be affected by this operation

MOV R1,45H

ADD A=05H +@ R1=0AH =



Signed Addition

-23H

2's Complement

00100011

11011100

+ 1

11011101

UnSigned Addition

+

+

b) Subtraction:

SUBB A, #45 ; Function:
Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator.

In this group, we have instructions to

i. Subtract the contents of A with immediate data with or without carry.

i. SUBB A, #45H

ii. SUBB A, #B4H

ii. Subtract the contents of A with register Rn with or without carry.

i. SUBB A, R5

ii. SUBB A, R2

MOV R5, #05H

MOV A, #0AH

SUBB A, R5 ; A-R5 = 0A-05

b) Subtraction:

SUBB A, <src-byte>

Function: Subtract with borrow

iii. Subtract the contents of A with contents of memory with or without carry using **direct and indirect addressing**

i. SUBB A, 51H

ii. SUBB A, 75H

iii. SUBB A, @R1

iv. SUBB A, @R0

CYAC and OV flags will be affected by this operation.

c) Multiplication

MUL AB: This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register.

After multiplication the lower byte of the result will be stored in accumulator and higher byte of result will be stored in B register.

Eg. MOV A,#45H ; [A]=45H

MOV B, A

MOV A, #0AH

MULAB ; AXB = **02B2H**

d) Division:

DIV AB. This instruction divides the 8 bit unsigned number which is stored in A by the 8 bit unsigned number which is stored in B register.

After division the result will be stored in accumulator and remainder will be stored in B register.

Eg. MOV A,#02H ; [A]=02H

MOV B, A

MOV A, #0AH

DIV AB; A/B 0A/02= 05-→Q=A 00-→R=B

Write a program to add two 16 bit numbers and store result in R1 AND R2

FC45H and **02ECH**

ORG 0000H

CLR C ; CY=0

MOV A,#45H

ADD A,#ECH

MOV R1,A

MOV A,#02H

ADDC A,#FCH ; CY=01 +02+FC=FFH

MOV R2, A

01	ADD
FC	45
02	EC
ADDC	31

O/P

R1=31H R2=FFH

Write a program to add two 16 bit numbers
8100 and 8101 H **FC45H** and 8200H and 8201H **02ECH** and
store the result in 8300 8301 8302

1	
FC 8100	45 8101
02 8200	EC 8201
FF	31

BCD Addition

- **BCD** : Binary Coded Decimal
- **Binary Representation**: 0 to 9
- The 8051 performs addition in pure binary –this may lead to errors when performing BCD addition

1. Unpacked BCD	07	06	01	03	08	09
Packed BCD	77	96	84	45		

BCD Addition

Example

49 BCD	01001001 BCD
<u>38 BCD</u>	<u>00111000 BCD</u>
<u>87 BCD</u>	<u>10000001 (81BCD)</u>

Decimal Adjust

The result must be adjusted to yield the correct BCD result

1. DAA (decimal adjust instruction)
2. The carry flag is set if the adjusted number exceeds 99 BCD

MOV A, #25H

MOV B, A

MOV A, #47H

ADD A, B ;

DAA; UP COUNTER /DOWN

72H----→47+25=72

DA instruction works only on A

MOV A, #09H

ADD A, #11H; A = 1AH (expecting 20H if these are BCD numbers)

DA A;

DAA works as follows:

- If lower nibble is greater than 9 or auxiliary carry is 1, 6 is added to lower nibble
- If upper nibble is greater than 9 or carry is 1, 6 is added to upper nibble.

Increment: Increments the operand by one

INC increments the value of source by 1.

If the initial value of register is FFh, incrementing the value will cause it to reset to 0.

In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented.

If the initial value of DPTR is FFFFh, incrementing the value will cause it to reset to 0.

Increment: Increments the operand by one

Eg: INC A ; A=07

INC R4 ; R4=09

INC 54H ; 55H

INC @Ri @R3=05

INC DPTR ; 8101

Decrement: decrements the operand by one

- DEC decrements the value of source by 1.
- If the initial value of is 0, decrementing the value will cause it to **reset to FFh**.
- The Carry Flag is not set when the value "rolls over" from **00 to FFh**

Eg: DEC DPL

DEC DPH

DEC DPTR

a. -7

1. Write the number in binary form
2. Complement each bit
3. Add 1

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	1	1	1
1	1	1	1	1	0	0	0
							1

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	1	1	0	0	0
1	1	0	0	0	1	1	1
							1
1	1	0	0	1	0	0	0

**Show how the following numbers are represented
in the 8051
C. - 128**

Solution:





A T M E
College of Engineering



Logical instructions

www.atme.in

Logical Instructions

a) Logical AND

ANL destination, source: ANL does a bitwise "AND" operation between source and destination, leaving the resulting value in destination. The value in source is not affected.

"AND" instruction logically AND the bits of source and destination

ANL A, #FF

EXAMPLE

A = 05 , 0 0 0 0 0 1 0 1
 1 1 1 1 1 1 1 1
 0 0 0 0 0 1 0 1

b) Logical OR

ORL destination, source: ORL does a bitwise "OR" operation between source and destination, leaving the resulting value in destination. The value in source is not affected. " OR " instruction logically OR the bits of source and destination.

EXAMPLE

MOV A,#FAH

ORL A,#FFH

11111010

11111111

11111111 **FFH**

Logical Instructions

c) Logical Ex-OR

XRL destination, source: XRL does a bitwise "EX-OR" operation between source and destination, leaving the resulting value in destination. The value in source is not affected. " XRL " instruction logically EX-OR the bits of source and destination.

```
MOV A,#FAH
```

```
XRL A,#FFH
```

```
11111010
```

```
11111111
```

```
00000101  05H
```

d) Logical NOT

CPL complements operand, leaving the result in operand. If operand is a single bit then the state of the bit will be reversed. If operand is the Accumulator then all the bits in the Accumulator will be reversed.; A=08

CPLA = 00001000 \rightarrow 11110111

CPL C ;CY=0 \rightarrow CY=1

CPL bit address CPL P1.0 ;P1.0=0 P1.0=1

Logical Instructions

SWAPA ;A=08 80 F0 0F FC CF A1 1A

Swap the upper nibble and lower nibble of A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3 through 0 and bits 7 through 4). The operation can also be thought of as a 4- bit rotate instruction. **No flags are affected.**

Example: The Accumulator holds the value C5H (11000101B). The instruction, SWAP A leaves the Accumulator holding the value 5CH (01011100B)

Logical Instructions

XCH A,<byte>

Function: Exchange Accumulator with byte variable
Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: XCH A,@R0 // A=08 , @R0=AA
A=AA @R0=08

Logical Instructions

CPLA

Function: Complement Accumulator

Description: CPLA logically complements each bit of the Accumulator (one's complement). Bits which previously contained a 1 are changed to a 0 and vice-versa. **No flags are affected.**

Example: The Accumulator contains 5CH (01011100B). The following instruction, CPL A leaves the Accumulator set to A3H (10100011B).

Logical Instructions

Clear

- CLR A

- Clears each bit of the A register

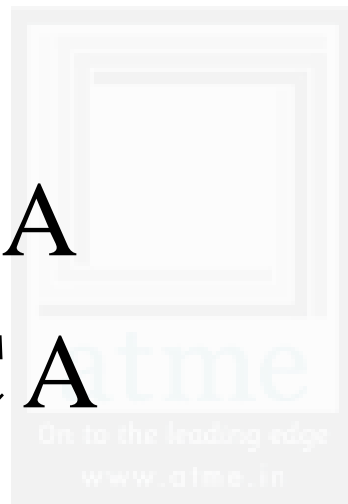
Rotate Instructions

1. RR A

2. RL A

3. RRC A

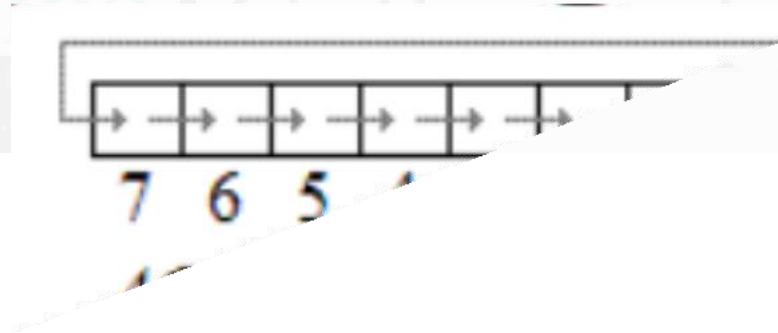
4. RLC A



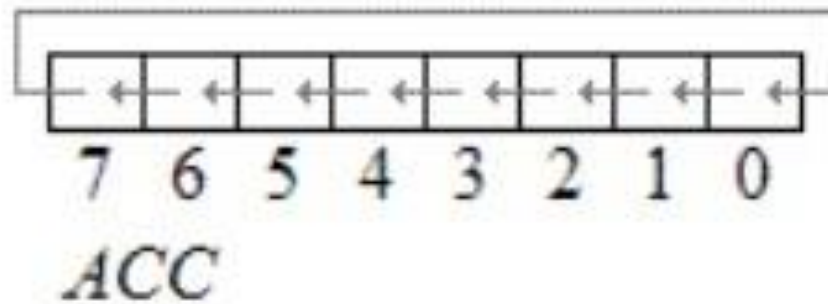
A T M E
College of Engineering

Rotate Instructions

- **RR A**
- This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right, with bit 0 going to bit 7.

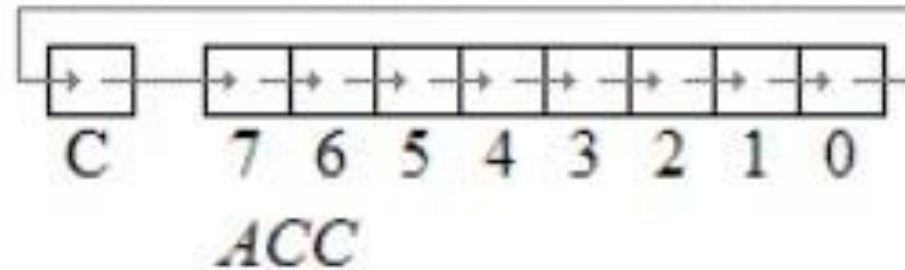


- **RL A**
- Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



Example: The Accumulator holds the value 0C5H (11000101B). The following instruction, RL A leaves the Accumulator holding the value **8BH (10001011B)** with the carry unaffected.

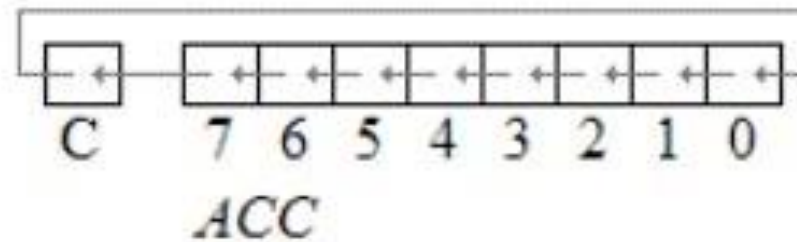
- **RRC A**
- Rotate right through the carry. Each bit is shifted one location to the right, with bit 0 going into the carry bit in the PSW, while the carry was at goes into bit 7



Example: The Accumulator holds the value 0C5H (11000101B), the carry is zero. The following instruction, RRC A leaves the Accumulator holding the value **62 (01100010B)** with the carry set.

- **RLC A**

- Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0.



Example: The Accumulator holds the value 0C5H(11000101B), and the carry is zero. The following instruction, RLC A leaves the Accumulator holding the value **8BH (10001010B)** with the carry set.

Show how 8051 does the following calculations

- A) ADD +37 and -115 = -78

Solution:

+37

0010 0101

-115

10001101

-78

10110010

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	1	0	0	1	0	1

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	1	1	0	0	1	1
1	0	0	0	1	1	0	0
							1
1	0	0	0	1	1	0	1

Show how 8051 does the following calculations

- MOV A,#-43
- MOV R2,#-78
- ADD A,R2
- **Solution:**

• 11010101

• 10110010

1 10000111

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	0	1	0	1	0	1	1
1	1	0	1	0	1	0	0
							1
1	1	0	1	0	1	0	1

Observe and notice the role of overflow flag

MOV A,#-128

MOV R4,#-2

ADD A,R4

Solution:

-128 10000000

-2 11111110

-130 01111110 **OV=1**

Observe and notice the role of overflow flag

MOV A,#-2

MOV R4,#-5

ADD A,R4

Solution:

-02 11111110

-05 11111011

-07 11111001 OV=0

Show the results of the following

- MOV A,#54H
- XRL A,#78H

• **Solution**

• **54 X-OR 78**

01010100

01111000

00101100

I/P	I/P	O/P
0	0	0
0	1	1
1	0	1
1	1	0

Show the results of the following

MOV A,#25H

MOV B,#65H

MUL AB

- **Solution**
- **25H*65H=0E99H**
- **A=? 99**
- **B=? 0E**

Show the results of the following

MOV A,#78H

SWAP A

Solution:

A=87H

Which instructions are illegal

1. MOV R3,#50
2. MOV R1,#500
3. MOV R7,#00
4. MOV R2,R3

Solution:

On to the leading edge
www.atme.in

A T M E
College of Engineering

Which are the two 16 bit registers

Solution:

DPTR

PC



R0=25H, A=35H What is O/P of the following code?

ADD A,R0; 25+35= 5A

MOV R0,A; A=5A--→R0=5A

ADD A,R0; A=5A +5A =B4H

Solution:

A= B4H

What is the status of CY,AC and P Flags

MOV A,#9CH

ADD A,#64H

Solution:

9C 1001**1100**

64 01100100

CY=1 AC=1 P=1

What type of Addressing mode is this instruction

1. MOV A,#45H
2. **MOVC A,@A+DPTR**
3. MOV A,@R1

Solution:

JUMP, LOOP AND CALL INSTRUCTIONS

Branch (JUMP) Instructions

Jump and Call Program Range There are 3 types of jump instructions.

They are:-

1. Relative Jump
2. Short Absolute Jump
3. Long Absolute Jump

1. Relative Jump

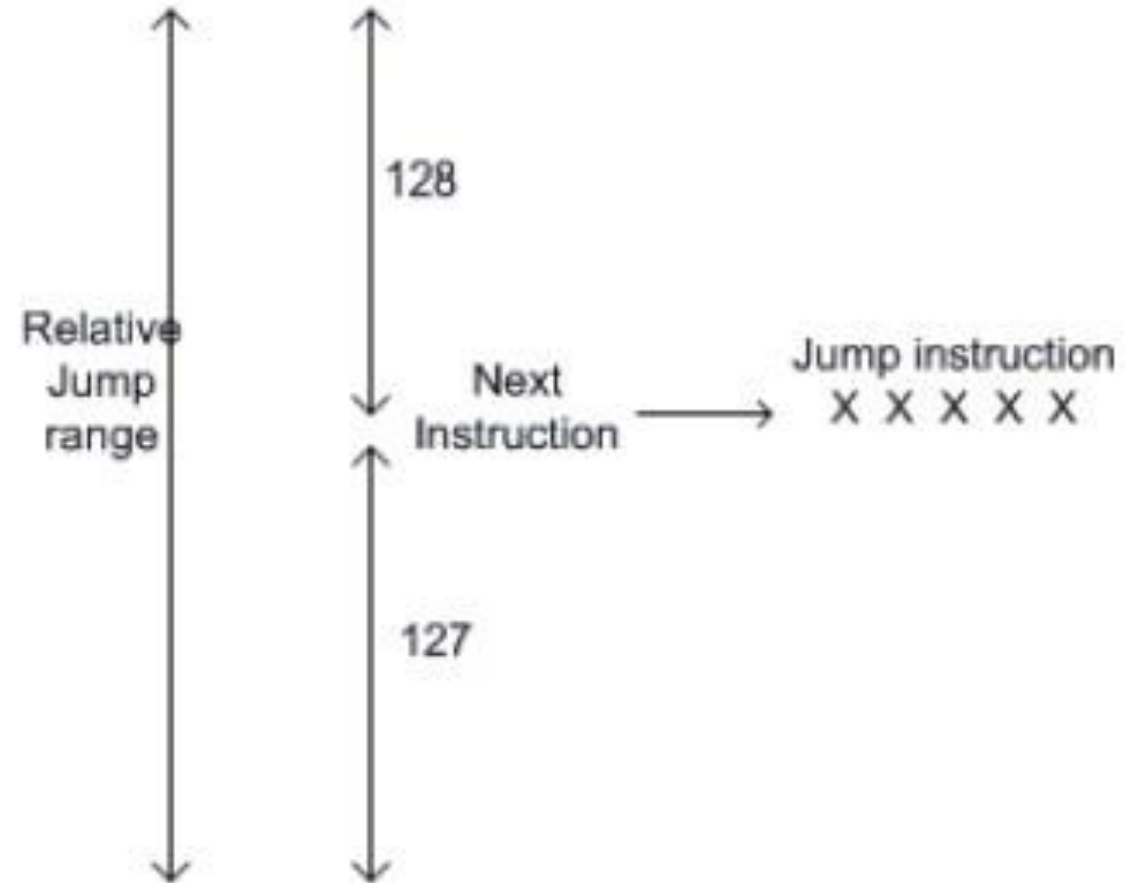
- Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by 128 or less) is called a **relative jump**. Schematically, the relative jump can be shown as follows: -

Advantage

Only **1 byte** of jump address needs to be specified in the 2's complement form, ie. For jumping ahead, the range is 0 to 127 and for jumping back, the range is -1 to -128.

Disadvantages of the absolute jump: -

1. Short jump range (-128 to 127 from the instruction following the jump instruction)



1. Relative Jump

ORG 0000H

MOV A,#FFH

ADD A,# FFH ; 01 FEH

JC HERE ; JUMP IF CARRY IS GENERATED

NOP

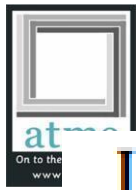
HERE: DB CARRY IS GENERATED

 **END**



SJMP

- **Operation:** SJMP Function
- Short Jump Syntax: SJMP 0500
- **Description:** SJMP jumps unconditionally to the address specified reladdr.
- Reladdr must be within -128 or +127 bytes of the instruction that follows the SJMP instruction
- MOV A,#45H ;2



A T M E



JC <relative address>

JNC <relative address>

JB bit, <relative address>

JNB bit, <relative address>

JBC bit, <relative address>

CJNE <destination byte>, <source byte>, <relative address>

DJNZ <byte>, <relative address>

JZ <relative address>

JNZ <relative address>

MOV R0,#00

BACK: NOP

DJNZ R0,BACK

Absolute Jump

In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.

In 8051, 64 kbyte of program memory space is divided into **32 pages of 2 kbyte each**.

The hexadecimal addresses of the pages are given as follows:-

ADDRESS DB BLUE

ACALL <address 11>

AJMP <address 11>

<i>Page (Hex)</i>	<i>Address (Hex)</i>
00	0000 - 07FF
01	0800 - 0FFF
02	1000 - 17FF
03	1800 - 1FFF
.	.
.	.
1E	F000 - F7FF
1F	F800 - FFFF

LJMP

- **Long Absolute Jump/Call**
- Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump.
- Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is not re-locatable.

Operation: LJMP

Function: Long Jump

Syntax: LJMP code address.

Description: LJMP jumps unconditionally to the specified code address.

LCALL <address 16>

LJMP <address 16>

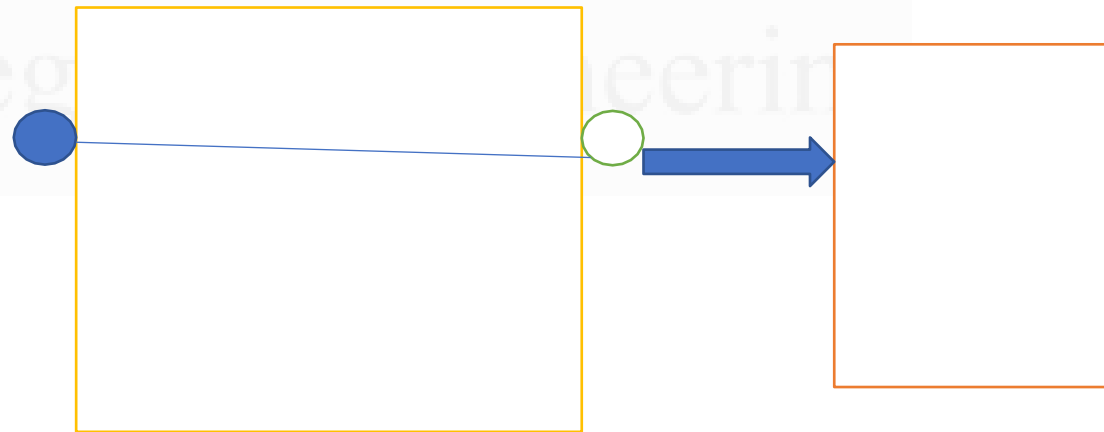
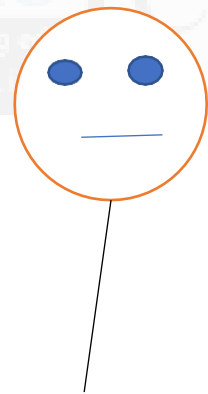
JMP @A+DPTR

Conditional Jump instructions.

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

All conditional jumps are short jumps.

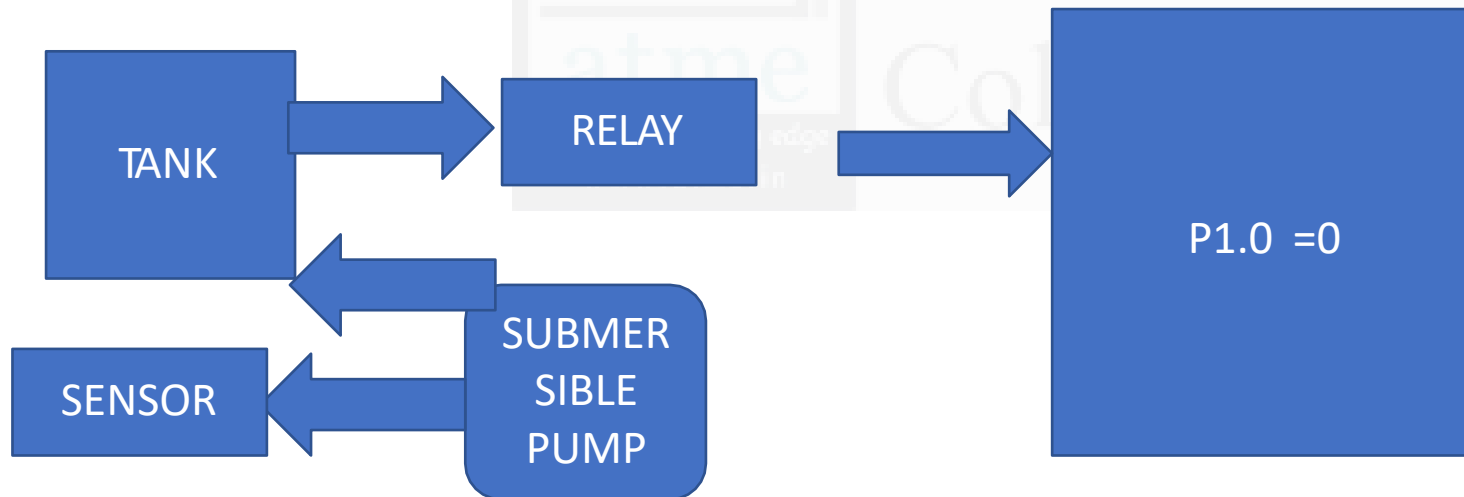
- **Operation: JNC**
- Function: Jump if Carry Not Set
- Syntax: JNC reladdr
- Description: JNC branches to the address indicated by reladdr if the carry bit is not set. If the carry bit is set program execution continues with the instruction following the JNC instruction



- Operation: JC
- Function: Jump if Carry Set
- Syntax: JC reladdr
- Description: JC will branch to the address indicated by reladdr if the Carry Bit is set. If the Carry Bit is not set program execution continues with the instruction following the JC instruction.
- MOV A,#FFH
- ADD A,#FFH; 01 FEH
- JC HERE
- HERE: DB INDIA

Operation: JNB

- Function: Jump if Bit Not Set
- Syntax: JNB bit addr, reladdr
- Description: JNB will branch to the address indicated by reladdress if the indicated bit is not set. If the bit is set program execution continues with the instruction following the JNB instruction.



- **Operation: JB**
- **Function: Jump if Bit Set**
- **Syntax: JB bit addr, reladdr**
- **Description:** JB branches to the address indicated by reladdr if the bit indicated by bit addr is set. If the bit is not set program execution continues with the instruction following the JB instruction.

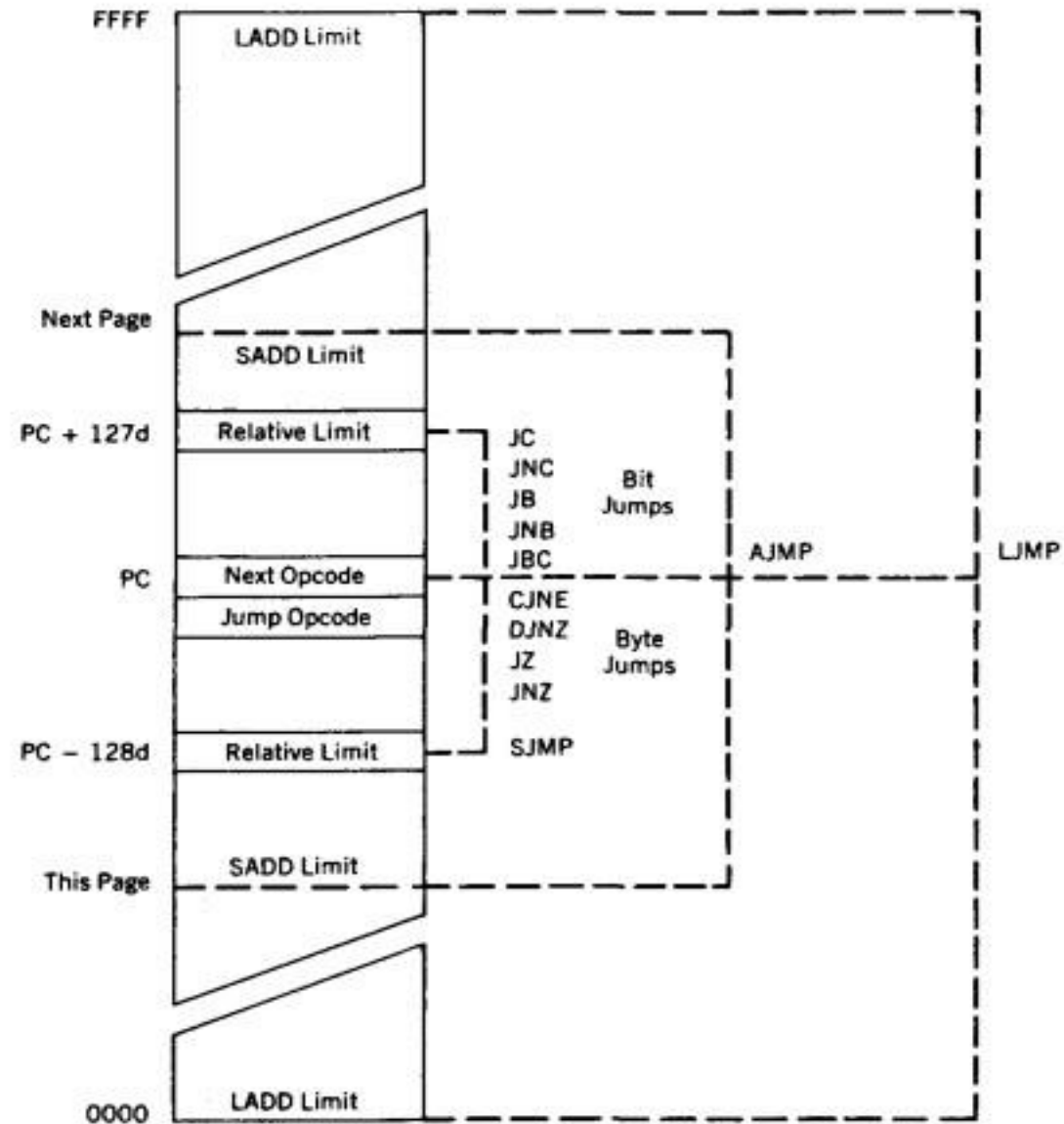
- **Operation: JNZ**
- **Function: Jump if Accumulator Not Zero**
- **Syntax: JNZ reladdr**
- **Description: JNZ will branch to the address indicated by reladdr if the Accumulator contains any value except 0. If the value of the Accumulator is zero program execution continues with the instruction following the JNZ instruction.**

- Operation: JZ
- Function: Jump if Accumulator Zero
- Syntax: JNZ reladdr
- Description: JZ branches to the address indicated by reladdr if the Accumulator contains the value 0. If the value of the Accumulator is non-zero program execution continues with the instruction following the JNZ instruction.

- **Operation: DJNZ**
- Function: Decrement and Jump if Not Zero
- Syntax: DJNZ register, reladdr
- Description: DJNZ decrements the value of register by 1. If the initial value of register is 0, decrementing the value will cause it to reset to 255 (0xFF Hex). If the new value of register is not 0 the program will branch to the address indicated by relative addr. If the new value of register is 0 program flow continues with the instruction following the DJNZ instruction

- **Operation: CJNE**
- **Function: Compare and Jump If Not Equal**
- **Syntax: CJNE operand1,operand2,reladdr**
- **Description:** CJNE compares the value of operand1 and operand2 and branches to the indicated relative address if operand1 and operand2 are not equal. If the two operands are equal program flow continues with the instruction following the CJNE instruction. The Carry bit (C) is set if operand1 is less than operand2, otherwise it is cleared.

Memory Address (HEX)



2.5. 1 Bit level jump instructions

- Bit level JUMP instructions will check the conditions of the bit and if condition is true, it jumps to the address specified in the instruction.
- All the bit jumps are **relative jumps**.

Bit Jumps

Bit jumps all operate according to the status of the carry flag in the PSW or the status of any bit-addressable location. All bit jumps are relative to the program counter.

Jump instructions that test for bit conditions are shown in the following table:

Mnemonic	Operation
JC radd	Jump relative if the carry flag is set to 1
JNC radd	Jump relative if the carry flag is reset to 0
JB b,radd	Jump relative if addressable bit is set to 1
JNB b,radd	Jump relative if addressable bit is reset to 0
JBC b,radd	Jump relative if addressable bit is set, and clear the addressable bit to 0

Note that no flags are affected unless the bit in JBC is a flag bit in the PSW. When the bit used in a JBC instruction is a port bit, the SFR latch for that port is read, tested, and altered.

Byte Jumps

Byte jumps—jump instructions that test bytes of data—behave as bit jumps. If the condition that is tested is *true*, the jump is taken; if the condition is *false*, the instruction after the jump is executed. All byte jumps are relative to the program counter.

The following table lists examples of byte jumps:

Mnemonic	Operation
CJNE A,add,radd	Compare the contents of the A register with the contents of the direct address; if they are <i>not</i> equal, then jump to the relative address; set the carry flag to 1 if A is less than the contents of the direct address; otherwise, set the carry flag to 0
CJNE A,#n,radd	Compare the contents of the A register with the immediate number n; if they are <i>not</i> equal, then jump to the relative address; set the carry flag to 1 if A is less than the number; otherwise, set the carry flag to 0
CJNE Rn,#n,radd	Compare the contents of register Rn with the immediate number n; if they are <i>not</i> equal, then jump to the relative address; set the carry flag to 1 if Rn is less than the number; otherwise, set the carry flag to 0
CJNE @Rp,#n,radd	Compare the contents of the address contained in register Rp to the number n; if they are <i>not</i> equal, then jump to the relative address; set the carry flag to 1 if the contents of the address in Rp are less than the number; otherwise, set the carry flag to 0

Subroutine CALL and RETURN Instructions

- Subroutines are handled by CALL and RET instructions There are two types of CALL instructions
- Call instructions may be included explicitly in the program as mnemonics or implicitly included using hardware interrupts.
- Subroutine: Subroutine is a standalone program or small program in a main program
- “**A Subroutine** is a program that may be used many times in the execution of a larger program. The subroutine could be written into the body of the main program everywhere it is needed resulting in the fastest possible code execution.”

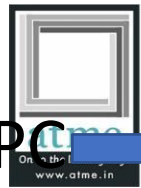
LCALL address (16 bit)

- This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address.
- This is a **3 byte instruction.**

- a. During execution of LCALL, $[PC] = [PC] + 3$; (if address where LCALL resides is say, $0x3254h$, then during execution of this instruction $[PC] = 3254h + 3h = 3257h$)
- b. $[SP] = [SP] + 1$; (if SP contains default value 07, then SP increments to 08)
- c. $[[SP]] = [PC_{7-0}]$; (lower byte of PC content ie., 57 will be stored at $SP = 08$)
- d. $[SP] = [SP] + 1$; (SP increments again and $[SP] = 09$)
- e. $[[SP]] = [PC_{15-8}]$; (higher byte of PC content ie., 32 will be stored at $SP = 09$)

With these the

f. $[PC] =$



STACK

SP

05

09

SP

14

08

SP

07



0510: MOV A, #45H

0511: ADD A, #FFH

0512: **ACALL**

0514: DB INDIA IS A SECULAR COUNTRY

ACALL address (11 bit)

- a. During execution of SCALL, $[PC] = [PC] + 2$; (if address where LCALL resides is say, 0x8549; during execution of this instruction $[PC] = 8549h + 2h = 854Bh$)
- b. $[SP] = [SP] + 1$; (if SP contains default value 07, then SP increments and $[SP] = 08$)
- c. $[[SP]] = [PC_{7-0}]$; (lower byte of PC content ie., 4B will be stored in memory location 08.)
- d. $[SP] = [SP] + 1$; (SP increments again and $[SP] = 09$)
- e. $[[SP]] = [PC_{15-8}]$; (higher byte of PC content ie., 85 will be stored in memory location 09.)

With these the address (0x854B) which was in PC is stored in stack.

- f. $[PC_{10-0}] = \text{address (11 bit)}$; the new address of subroutine is loaded to PC. No flags are affected.

RET instruction

- RET instruction pops top two contents from the stack and load it to PC.

MOV R0,#07 ; R0=08H

BACK:MOV R1,#45H

MOV A,#0AH

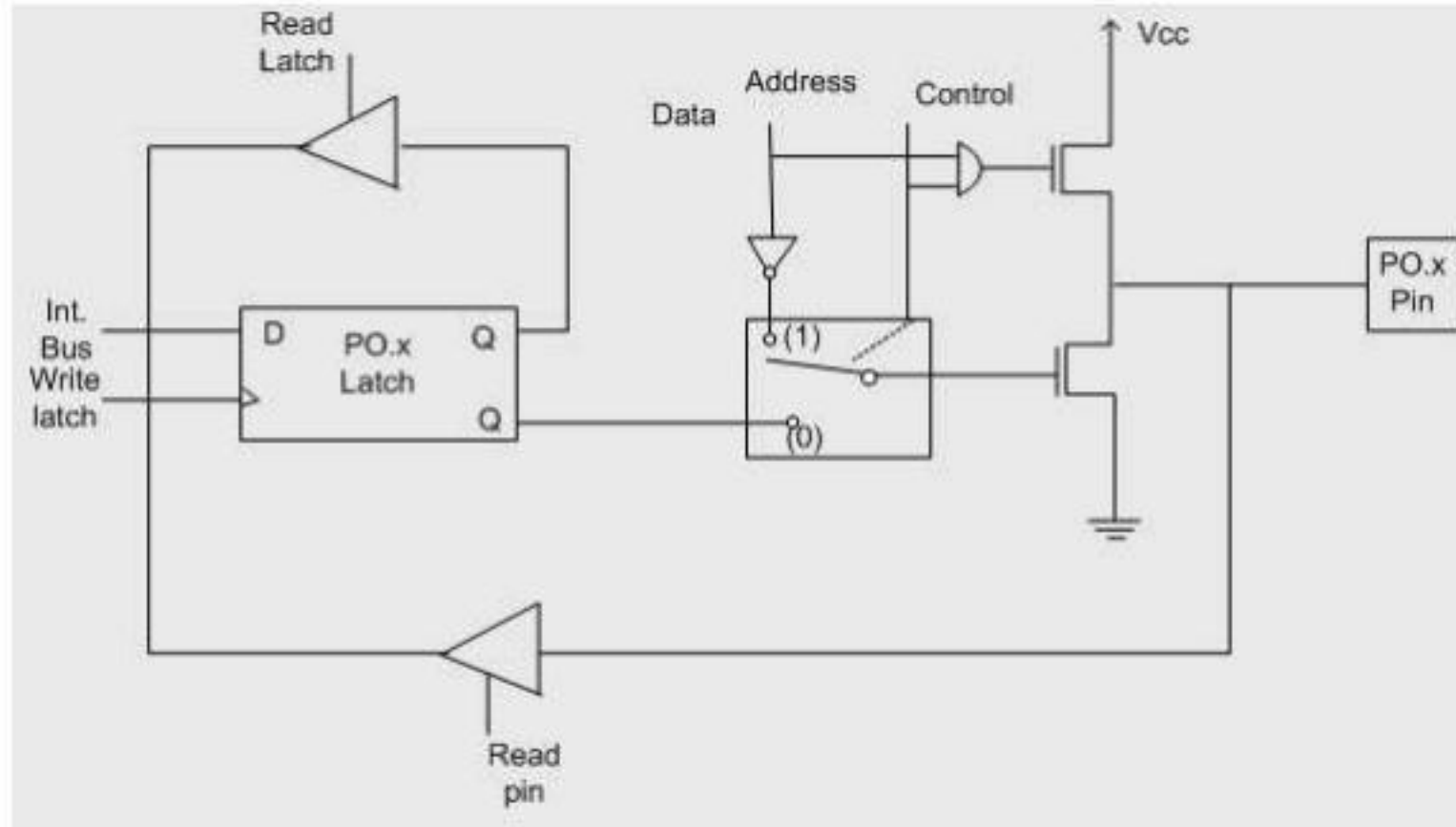
ADD A, R1 ; **45H+0AH**

RET

I O port programming

- 1. I/O Port pins, Ports and Circuits: One major feature of a microcontroller is versatility built into the I/O circuits that connect the 8051 to the outside world.
- 2. Out of 40 pins 24 pins may each be used for one of two entirely different functions yielding a total pin configuration of 64.
- 3. But the port pins have been multiplexed to perform different functions to make 8051 as 40 Pin IC
- The port pin circuitry

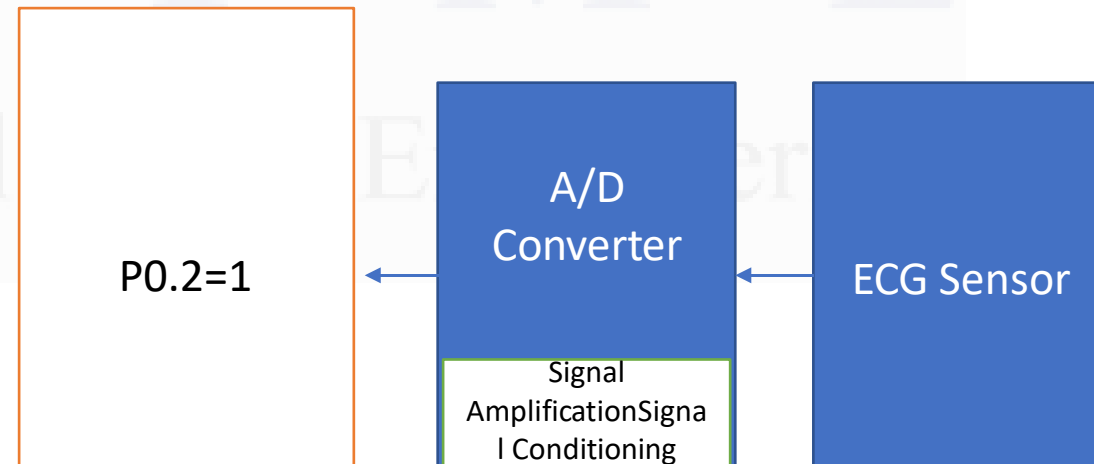
Port-0



PROGRAMS

MAKE P0.2 AS I/P PORT

SETB P0.2 ; P0.2=1 [I/P PORT]



Port 1 is configured as an input port.

Toggle the port 55H 01010101 = 10101010 AAH

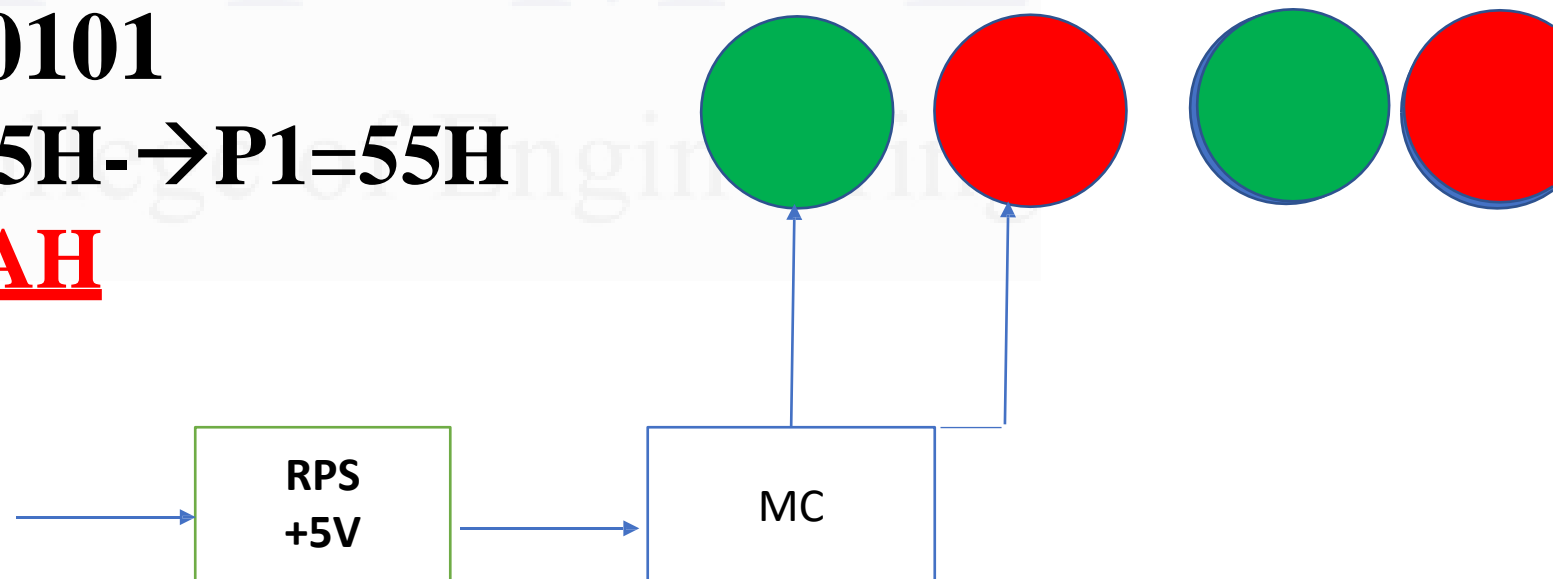
Toggle all bits of continuously

MOV A,#55 H;01010101

BACK: MOV P1,A ;55H→P1=55H

CPL A ;55H→AAH

SJMP BACK

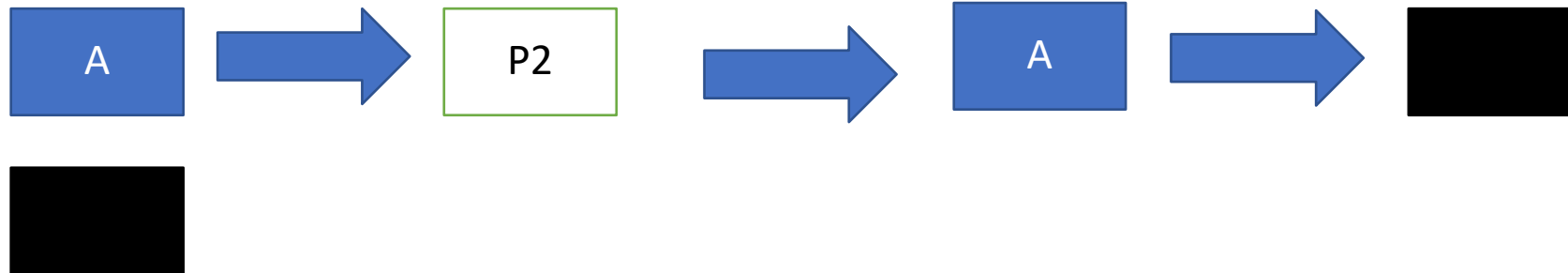


Get a byte and place it in P2 and P1

MOV A, # FFH ; FF=11111111 1 BYTE

MOV P2, A ; A=FFH---→P2=FFH

MOV P1, A; A=FFH----→P1=FFH



Write a program to configure port 1 as I/P port. Then data is received from the port and saved in R5,R6,R7

```
MOV    A,#0FFH    ;A=FF hex
MOV    P1,A        ;make P1 an input port
                     ;by writing all 1s to it

MOV    A,P1        ;get data from P1
MOV    R7,A        ;save it in reg R7
ACALL  DELAY       ;wait

MOV    A,P1        ;get another data from P1
MOV    R6,A        ;save it in reg R6
ACALL  DELAY       ;wait

MOV    A,P1        ;get another data from P1
MOV    R5,A        ;save it in reg R5
```

Toggle the bit of P1.2 continuously

Method 1:

```
BACK:    CPL    P1.2        ;complement P1.2 only
          ACALL  DELAY
          SJMP   BACK
```

Method 2:

```
;another variation of the above program follows
AGAIN:   SETB   P1.2        ;change only P1.2=high
          ACALL  DELAY
          CLR    P1.2        ;change only P1.2=low
          ACALL  DELAY
          SJMP   AGAIN
```

Example 4-6

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

- (a) If switch = 0, send letter 'N' to P2.
- (b) If switch = 1, send letter 'Y' to P2.

Use the carry flag to check the switch status. This is a repeat of the last example.

Solution:

```
                SETB P1.7                ;make P1.7 an input
AGAIN:          MOV  C,P1.2              ;read the SW status into CF
                JC    OVER               ;jump if SW = 1
                MOV  P2,#'N'             ;SW = 0, issue 'N' to P2
                SJMP  AGAIN               ;keep monitoring
OVER:           MOV  P2,#'Y'             ;SW = 1, issue 'Y' to P2
                SJMP  AGAIN               ;keep monitoring
```

PROGRAMS CONTINUED

Write a program to configure port 1 as I/P port. Then data is received from the port and saved in R5,R6,R7

Solution:

MOV A,#FFH

MOV P1,A

MOV A,P1

MOV R7,A

ACALL DELAY

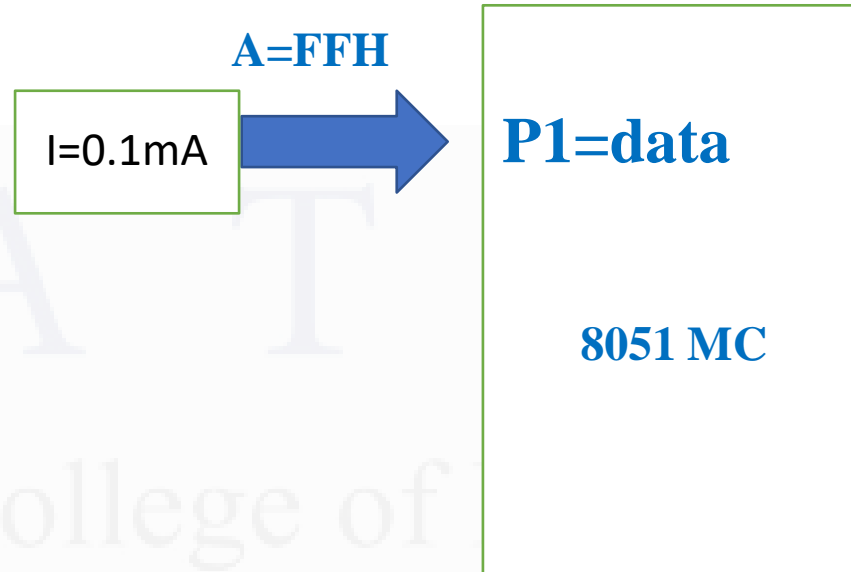
MOV A,P1

MOV R6,A

ACALL DELAY

MOV A,P1

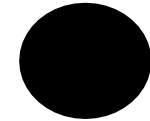
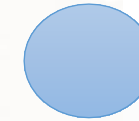
MOV R5,A



Toggle the bit of P1.2 continuously

Method 1:

```
BACK:    CPL    P1.2        ;complement P1.2 only  
         ACALL  DELAY  
         SJMP   BACK
```



Method 2:

```
;another variation of the above program follows  
AGAIN:   SETB   P1.2        ;change only P1.2=high  
         ACALL  DELAY  
         CLR    P1.2        ;change only P1.2=low  
         ACALL  DELAY  
         SJMP   AGAIN
```

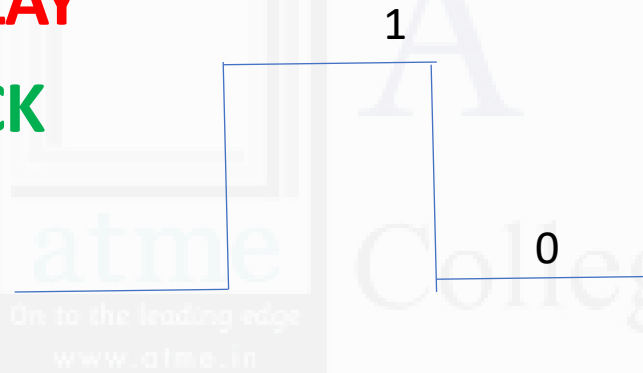
BACK: SETB P1.2 ; P1.2=1

ACALL DELAY

CLR P1.2 ; P1.2=0

ACALL DELAY

SJMP BACK

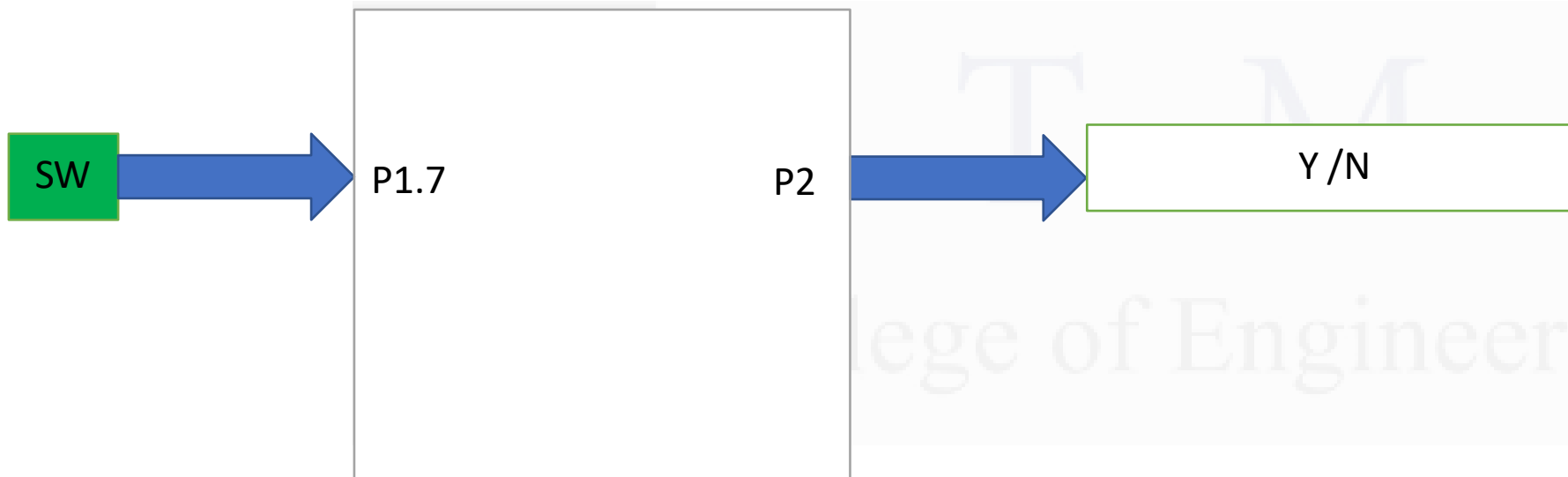


Example 4-6

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

- (a) If switch = 0, send letter 'N' to P2.
- (b) If switch = 1, send letter 'Y' to P2.

Use the carry flag to check the switch status. This is a repeat of the last example.



AGAIN: MOV C, P1.2 ; P1.7=0-→C=0

JC OVER ; CY=0

MOV P2,#'N'

SJMP **AGAIN**

OVER: MOV P2,# 'Y'

SJMP **AGAIN**

PROGRAMS ON ARITHMETIC INSTRUCTIONS

**Two numbers are stored in registers R0 and R1.
Verify if their sum is greater than FFH send Y to P1**

MOV A,R0 ; R0=05H

ADD A,R1 ;R1=07H ; 05H+07H= 0CH

JC MESSAGE

SJMP NEXT

MESSAGE: MOV A,# 'Y'

MOV P1, A

NEXT: NOP

END

Analyze the following program

CLR C

MOV A,#4CH

SUBB A,#6EH

JNC NEXT

CPLA

NEXT:MOV R1,A

Analyze the following program

Solution:

4C	0100 1100	01001100
- 6E	01101110	2's Complement 10010010
- 22		0 11011110

CY=1 Since the result is negative in 2's complement

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. Place the temperature value into the registers indicated by the following

If T=75

If T<75

If T>75

Solution:

MOV P1,#FFH ;P1=I/P

MOV A,P1 ; A=65

CJNE A,#75,OVER

MOV R0,A ;R0=75

SJMP EXIT

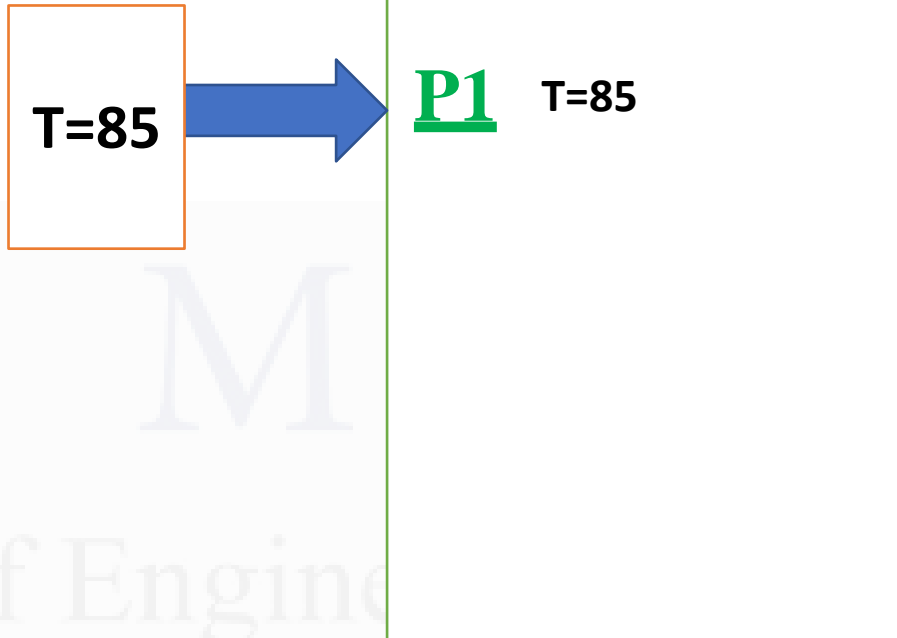
OVER: JNC NEXT

MOV R1,A ; R1=65

SJMP EXIT

NEXT: MOV R2,A ;R2=85

EXIT: NOP



P1.3 Toggle Continuously

```
BACK :SETB P1.3; P1.3=1
```

```
ACALL DELAY
```

```
CLR P1.3
```

```
ACALL DELAY
```

```
SJMP BACK
```

Programs Continued

Write a program to transfer value 41H serially(one bit at a time) via P2.1.Put **two highs** at the start and end of the data. Send the byte LSB first

Solution:

MOV A,#41H

SETB P2.1

SETB P2.1

MOV R5,#00

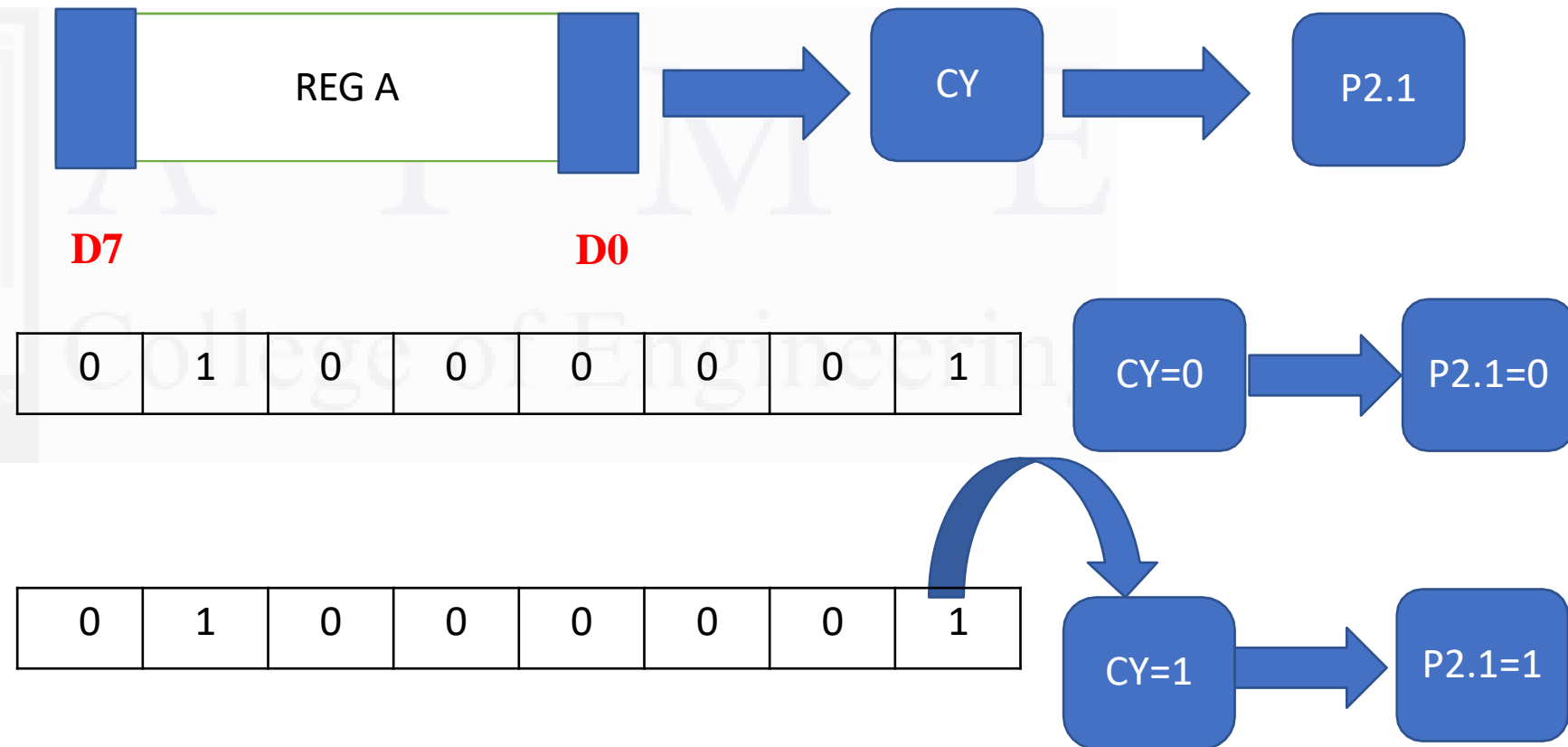
HERE: RRC A

MOV P2.1,C

DJNZ R5,**HERE**

SETB P2.1 ;High

SETB P2.1 ; High



0	1	0	0	0	0	0	1
---	---	---	---	---	---	---	---

CY=1

1	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

CY=1

0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

CY=0

A=A0H

A=A0H

Write a test program for the DS89C420/30 chip to toggle all the bits of P0, P1, and P2 every 1/4 of a second. Assume a crystal frequency of 11.0592 MHz.

Solution:

;Tested for the DS89C420/30 with XTAL = 11.0592 MHz.

```

                                ←
BACK:  ORG      0
                                ←
        MOV     A, #55H
                                ←
        MOV     P0, A
                                ←
        MOV     P1, A
                                ←
        MOV     P2, A
                                ←
        ACALL   QSDELAY
                                ← ;Quarter of a second delay
        MOV     A, #0AAH
                                ←
        MOV     P0, A
                                ←
        MOV     P1, A
                                ←
        MOV     P2, A
                                ←
        ACALL   QSDELAY
                                ←
        SJMP    BACK
                                ←

```

P1.0=1

P1.0=0

Write the following programs.

- (a) Create a square wave of 50% duty cycle on bit 0 of port 1.
- (b) Create a square wave of 66% duty cycle on bit 3 of port 1.

Solution:

- (a) The 50% duty cycle means that the “on” and “off” states (or the high and low portions of the pulse) have the same length. Therefore, we toggle P1.0 with a time delay in between each state.

Method 1:

```
HERE:    SETB    P1.0        ;set to high bit 0 of port 1
          LCALL   DELAY      ;call the delay subroutine
          CLR     P1.0       ;P1.0=0
          LCALL   DELAY
          SJMP    HERE       ;keep doing it
```

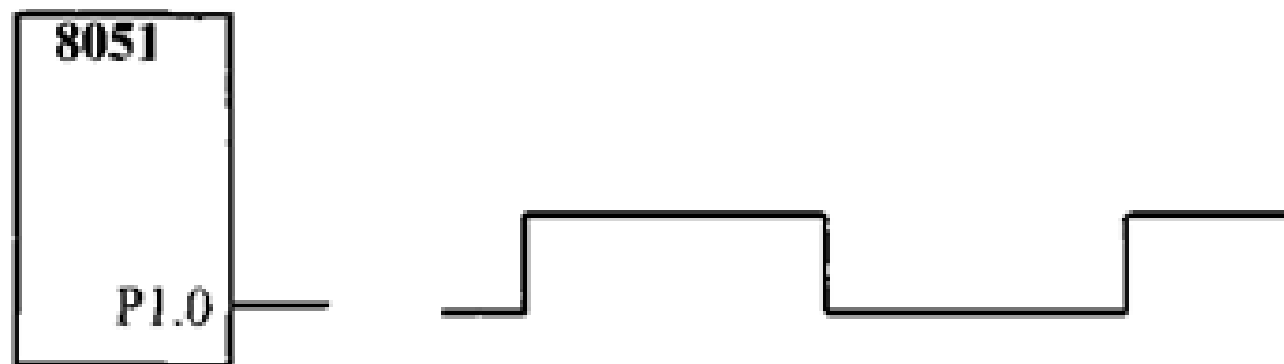
$$\frac{1}{2} = 50$$

$$\frac{2}{3} = 66.6$$

Method 2:

Another way to write the above program is:

```
HERE:      CPL      P1.0      ;complement bit 0 of port 1  
           LCALL    DELAY      ;call the delay subroutine  
           SJMP     HERE      ;keep doing it
```



(b) The 66% duty cycle means the “on” state is twice the “off” state.

```
BACK:    SETB    P1.3        ;set port 1 bit .3 high
          LCALL   DELAY       ;call the delay subroutine
          LCALL   DELAY       ;call the delay subroutine again
          CLR     P1.3        ;clear bit 2 of port 1(P1.3=low)
          LCALL   DELAY       ;call the delay subroutine
          SJMP    BACK        ;keep doing it
```



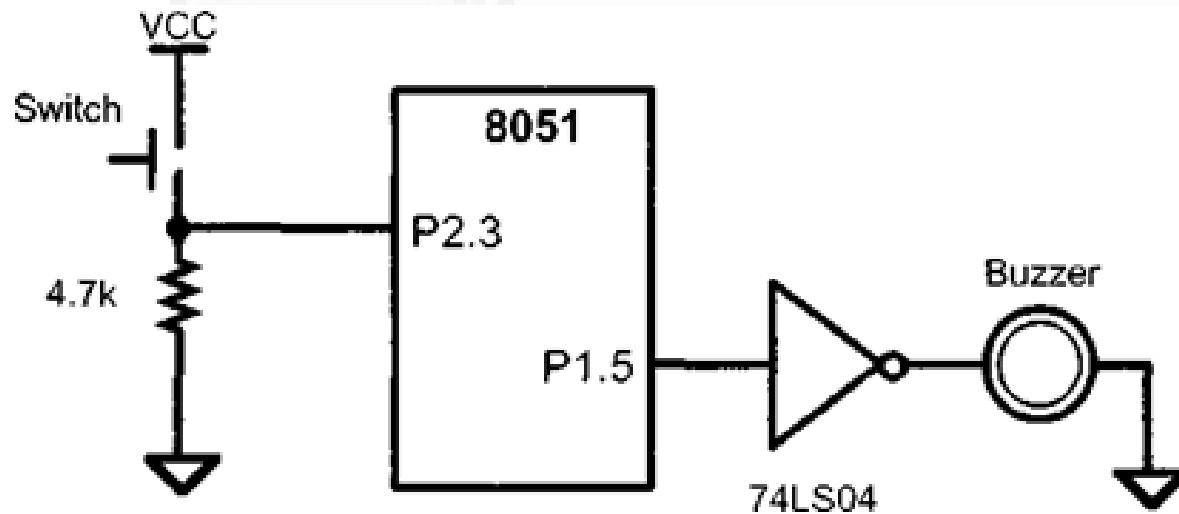
Write a program to perform the following:

- (a) keep monitoring the P1.2 bit until it becomes high
- (b) when P1.2 becomes high, write value 45H to port 0
- (c) send a high-to-low (H-to-L) pulse to P2.3

Solution:

```
                SETB P1.2                ;make P1.2 an input
                MOV  A,#45H              ;A=45H
AGAIN:          JNB  P1.2,AGAIN          ;get out when P1.2=1
                MOV  P0,A                ;issue A to P0
                SETB P2.3                ;make P2.3 high
                CLR  P2.3                ;make P2.3 low for H-to-L
```

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**





A T M E

College of Engineering



Solution:

```
HERE: JNB  P2.3, HERE
```

```
SETB  P1.5
```

```
CLR   P1.5
```

```
SJMP  HERE
```

```
;keep monitoring for high
```

```
;set bit P1.5=1
```

```
;make high-to-low
```

```
;keep repeating
```



College of Engineering

A switch is connected to pin P1 .0 and an LED to pin P2.7.
Write a program to get the status of the switch and send it to the LED.

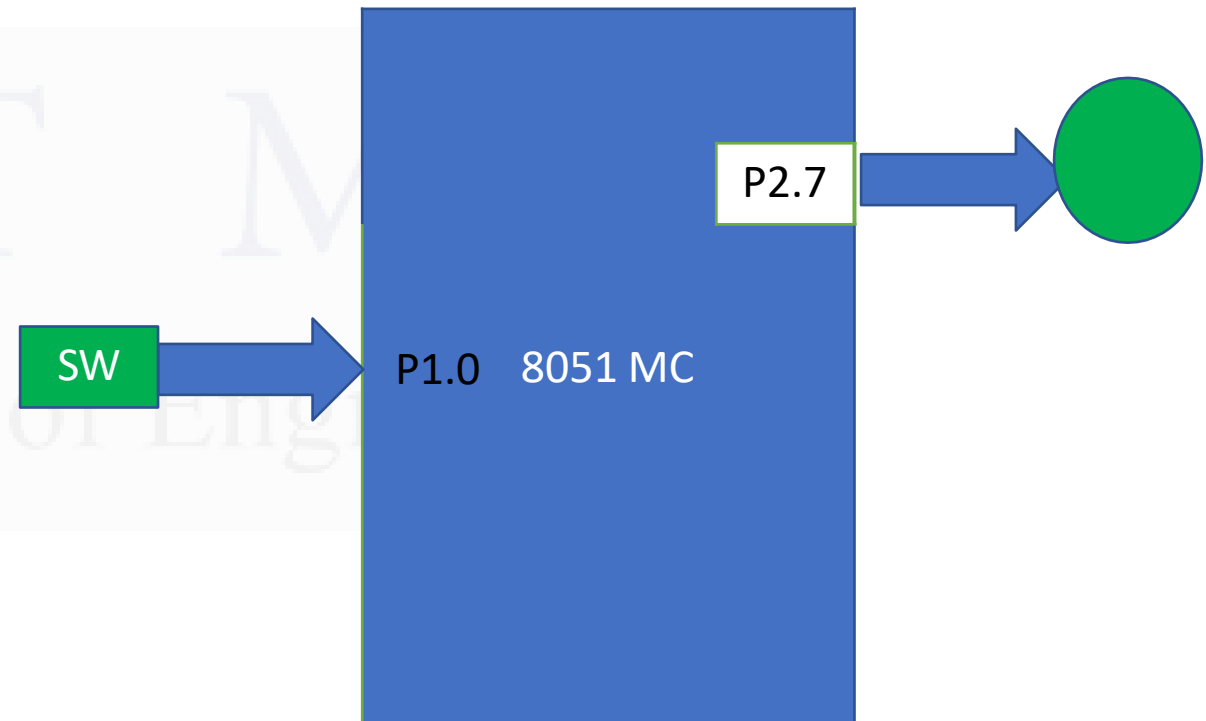
Solution:

SETB P1.0

AGAIN: MOV C,P1.0

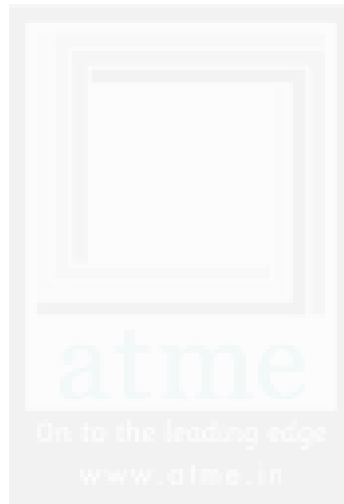
MOV P2.7,C

SJMP **AGAIN**





A T M E
College of Engineering



Thank You
A T M E
College of Engineering