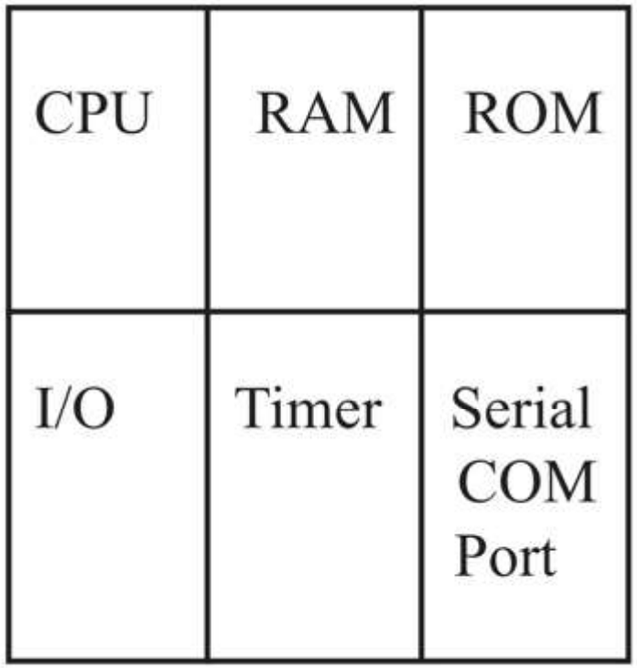
# MODULE:3 MICROCONTROLLER ARCHITECTURE: INTEL 8051

# REFERENCE


- Mohammad Ali Mazidi, Janice G. Mazidi, Rolin D. McKinlay, The 8051 Microcontroller and Embedded Systems, 2014, 2nd Edition, Pearson, India.

# MICROCONTROLLER




**Microcontroller has**

- **CPU (microprocessor)**
- **RAM**
- **ROM**
- **I/O ports**
- **Timer**
- **ADC and other peripherals**

**Microcontroller-It is a controlling device in which memory and I/O output component is present internally.**

Micro Controller contains a CPU, Memory, I/O all integrated into one chip.

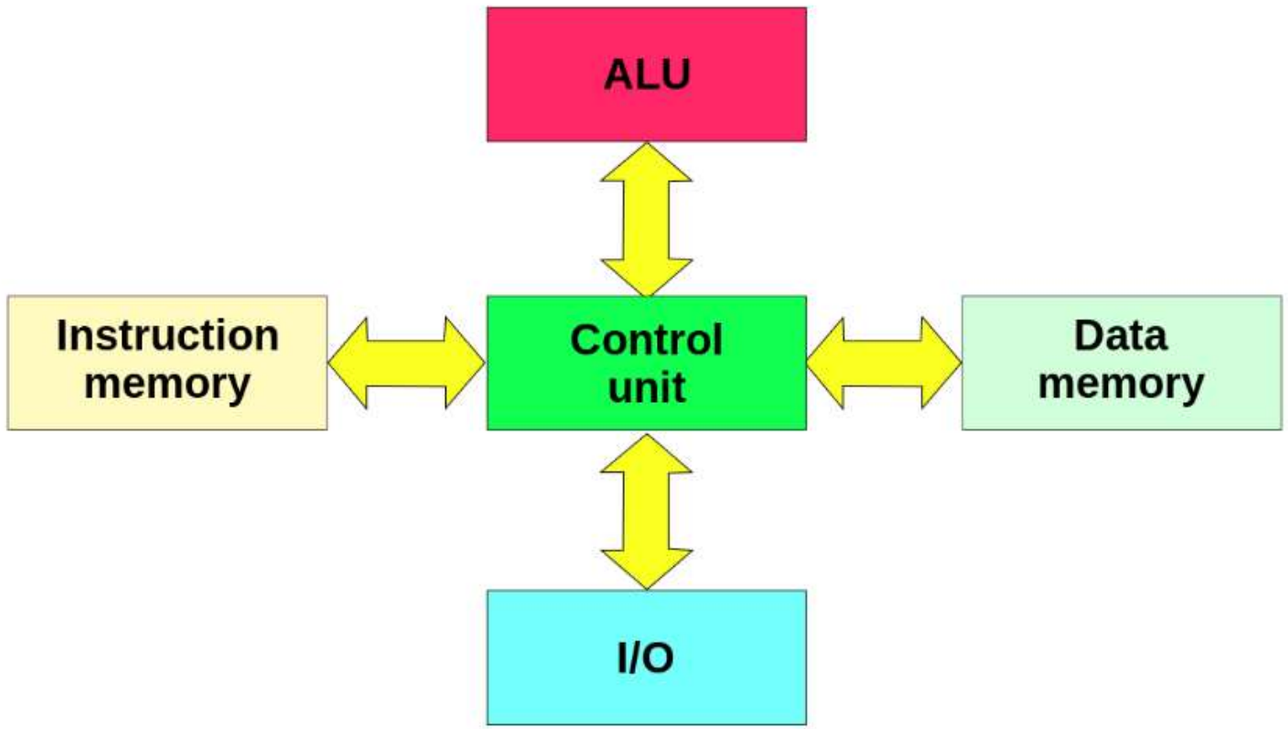
# Some Embedded Products Using Microcontrollers

- Appliances
- Intercom
- Telephones
- Security systems
- Garage door openers
- Answering machines
- Fax machines
- Home computers
- TVs
- Cable TV tuner
- VCR
- Camcorder
- Remote controls
- Video games
- Cellular phones
- Musical instruments
- Sewing machines
- Lighting control
- Paging
- Camera
- Pinball machines
- Toys
- Exercise equipment

# Harvard Architecture-Used in Microcontrollers

**Harvard Architecture**

The Harvard architecture stores machine instructions and data in separate memory units that are connected by different busses.



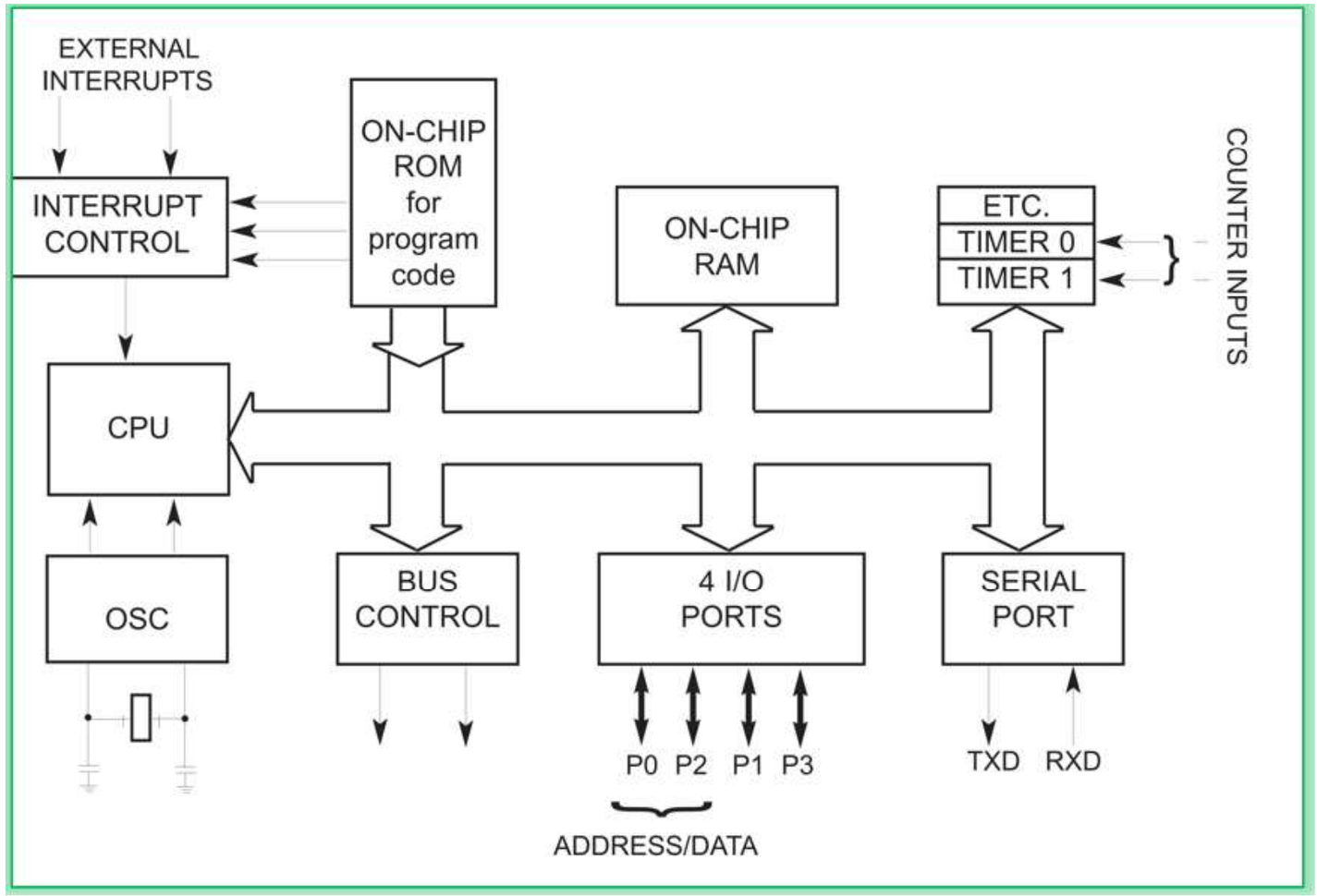
# 8051 Microcontroller Features

- **8051 is an 8-bit microcontroller**
- **4KB of ROM storage**
- **128 bytes of RAM storage**
- **Two 16-bit timers.**
- **It consists of four I/O ports ,each 8 bit wide**
- **An on-chip crystal oscillator is integrated in the microcontroller having crystal frequency of 12 MHz.**
- **Six interrupts sources**

| Feature | Quantity |
|---|---|
| ROM | 4K bytes |
| RAM | 128 bytes |
| Timer | 2 |
| I/O pins | 32 |
| Serial port | 1 |
| Interrupt sources | 6 |

*Note:* ROM amount indicates on-chip program space.

# 8051 block diagram –Architecture

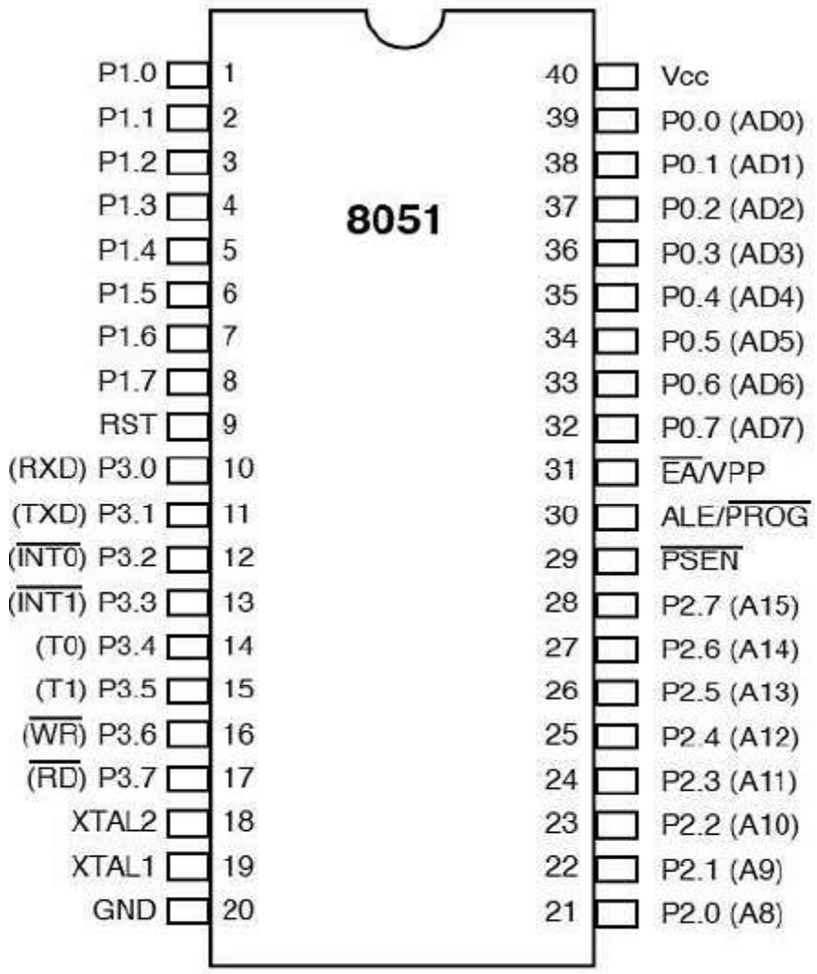


# Comparison of 8051 family members

| Feature | 8051 | 8052 | 8031 |
|---|---|---|---|
| ROM (on-chip program space in bytes) | 4K | 8K | 0K |
| RAM (bytes) | 128 | 256 | 128 |
| Timers | 2 | 3 | 2 |
| I/O pins | 32 | 32 | 32 |
| Serial port | 1 | 1 | 1 |
| Interrupt sources | 6 | 8 | 6 |

# PINDIAGRAM of 8051

40 PIN IC




Pins 1 to 8 – These pins are known as Port 1. This port doesn't serve any other functions. It is internally pulled up, bi-directional I/O port.

Pin 9 – It is a RESET pin, which is used to reset the microcontroller to its initial values.

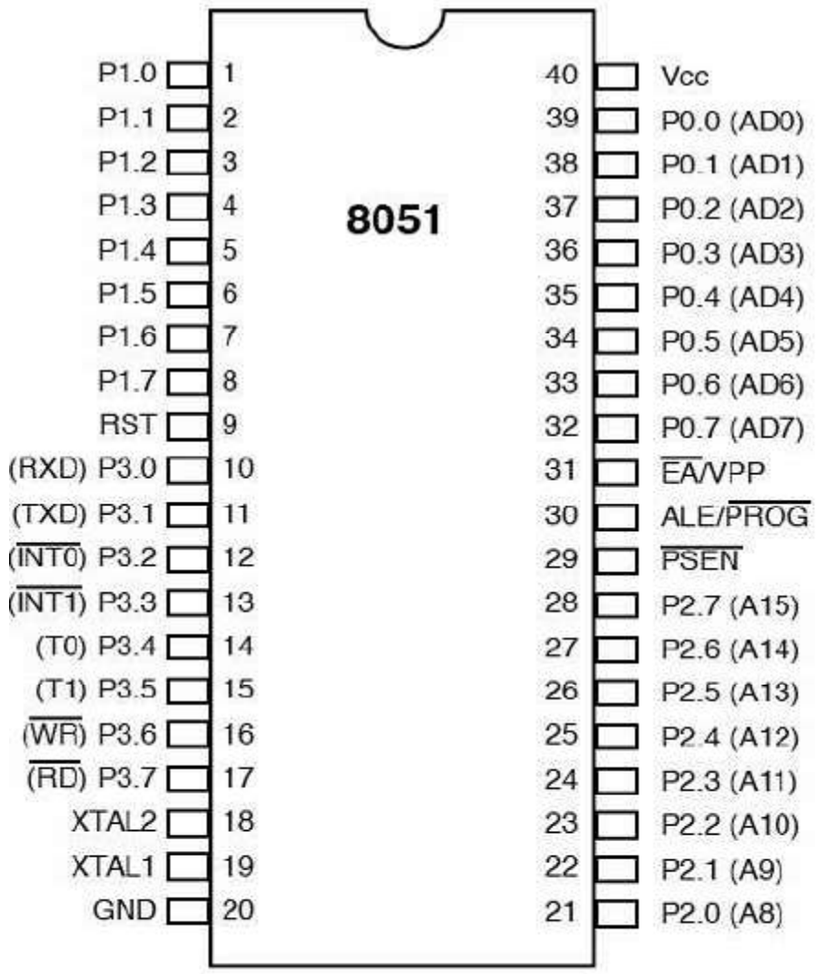
Pins 10 to 17 – These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc.

Pins 18 & 19 – These pins are used for interfacing an external crystal to get the system clock.

Pin 20 – This pin provides the power supply to the circuit.

Pins 21 to 28 – These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.

# PINDIAGRAM of 8051




Pin 29 – This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.

Pin 30 – This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.

Pin 31 – This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.

Pins 32 to 39 – These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.

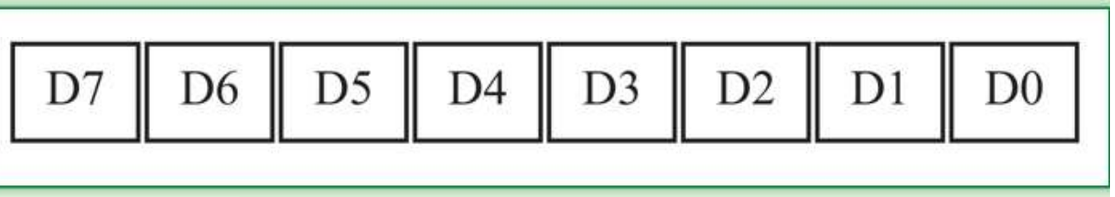
Pin 40 – This pin is used to provide power supply to the circuit.

# Registers in 8051

Register are used to store information temporarily, while the information could be

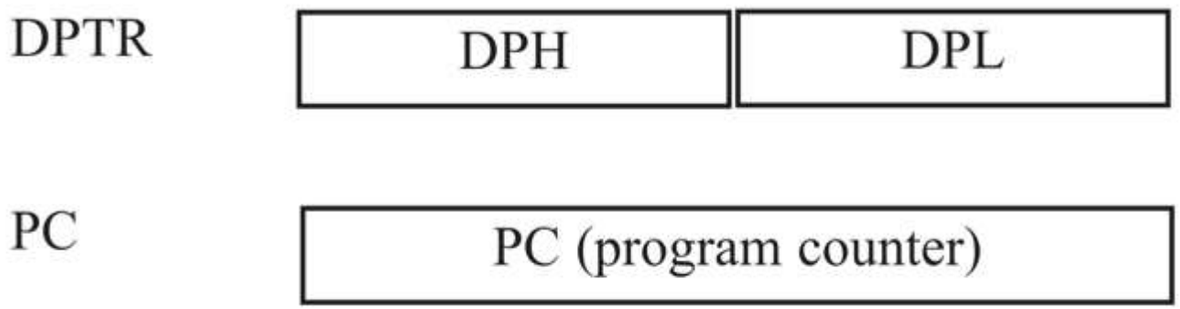
- a byte of data to be processed, or
- an address pointing to the data to be fetched

**The vast majority of 8051 registers are 8-bit registers.**

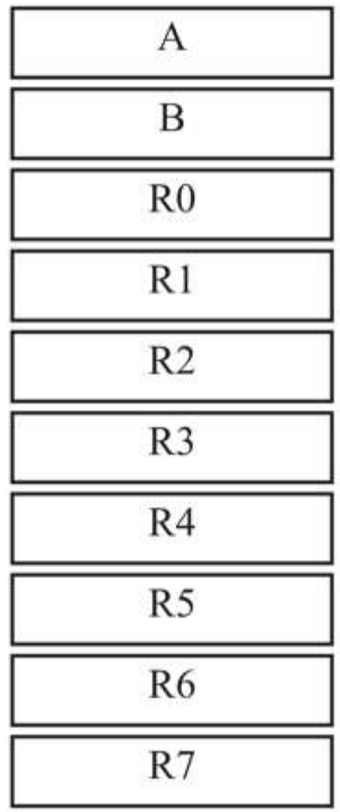



.

**Some 16 bit registers**




Some 8 bit registers



# Registers in 8051

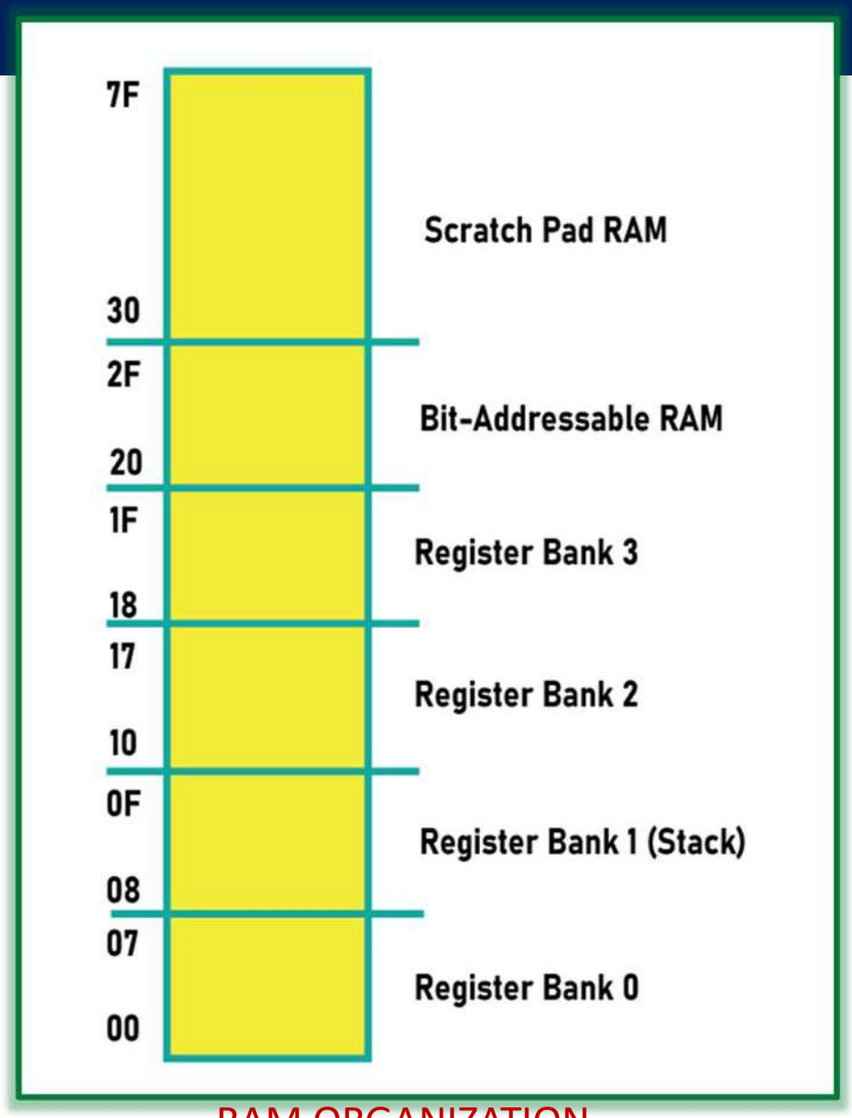
- A-The accumulator, register A, is used for all arithmetic and logic operations.
- B-The "B" register is very similar to the Accumulator in the sense that it may hold an 8-bit (1-byte) value.
  - The "B" register is used only by two 8051 instructions: MUL AB and DIV AB.
  - To quickly and easily multiply or divide A by another number, you may store the other number in "B" and make use of these two instructions.
  - The "B" register is often used as yet another temporary storage register.
- R0-R7- These registers function as auxiliary or temporary storage registers in many operations.

# Registers in 8051

- PC- Program Counter 16 bit register which tells where the next instruction to execute can be found in the memory. (So the address range is from 0000-FFFF)
- PC starts at 0000h when the 8051 initializes or powered up
- DPTR-Data pointer is a 16 bit register meant for pointing to data.
- It is used by the 8051 to access external memory using the address indicated by DPTR. DPTR holds 2 bytes.

# Registers in 8051

- Stack pointer
- The register used to acces stack is Stack pointer.
- Stack is portion of RAM used to store information-either data or address.
- SP is 8 bit wide, which can take values from 00- FF H
- When 8051 is powered up SP contains value 07, which means RAM location 08 is the first location used for stack in 8051.



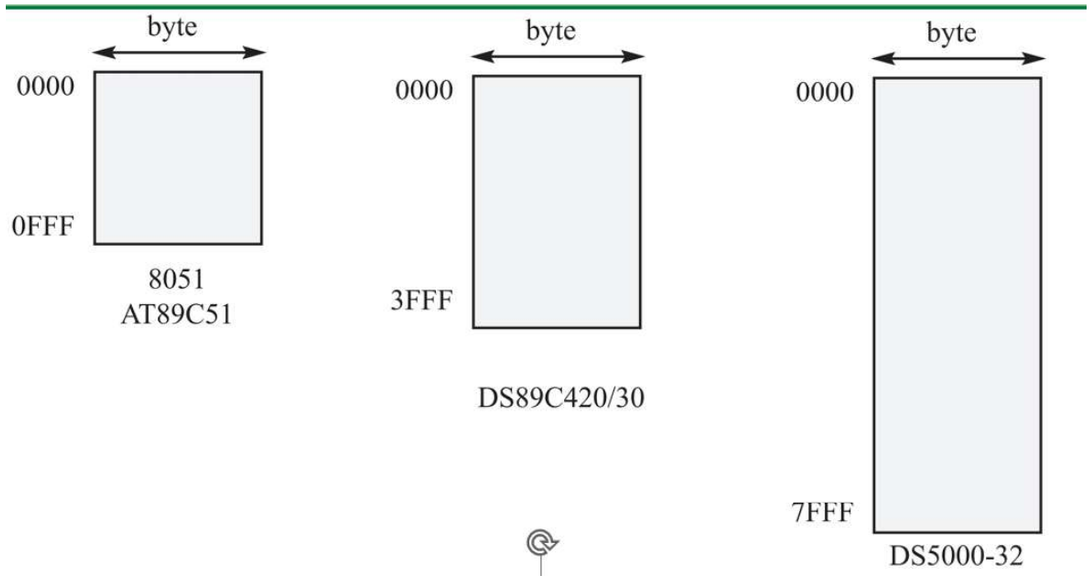

RAM ORGANIZATION

# Memory organization of 8051- On chip ROM space

- Some family members of 8051 have only 4K bytes of on-chip ROM (e.g. 8751, AT8951);
- some have 8K ROM like AT89C52,
- there are some family members with 32K bytes and 64K bytes of on-chip ROM such as Dallas Semiconductor.
- No member of the 8051 family can access more than 64K bytes of opcode since the program counter in 8051 is a 16-bit register (0000 to FFFF address).
- The first location of the program ROM inside the 8051 has the address of 0000H, whereas the last location can be different depending on the size of the ROM on the chip.
- Among the 8051 family members, AT8951 has 4k bytes of on-chip ROM having a memory address of 0000 (first location) to 0FFFH (last location).

# Memory organization of 8051- On chip ROM space

- Among the 8051 family members, AT8951 has 4k bytes of on-chip ROM having a memory address of 0000 (first location) to 0FFFH (last location).




4KB

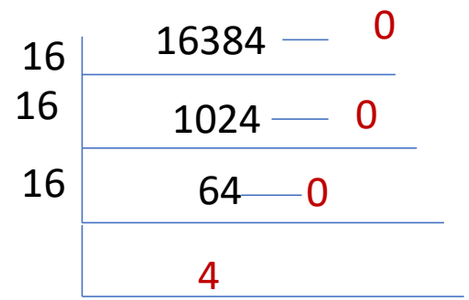
4KB=4096 bytes(4*1024 bytes) -12 address lines.

1111-F ,1111-F,1111-F So memory address range si 0000- 0FFF

**Calculate the last address location if ROM space is 64KB? - FFFF**

# Home Work

**Find the ROM memory address of each of the following 8051 chips.**
**(a)AT89C51 with 4KB (b) DS89C420 with 16KB (c) DS5000-32 with 32KB**




(b)With 16K bytes of on-chip ROM memory space, we have 16,384 bytes (16 x 1024 = 16,384), which gives 0000 - 3FFFH. (4000h-1)
(c) With 32K bytes we have 32,768 bytes (32 x 1024 = 32,768). Converting 32,768 to hex, we get 8000H; therefore, the memory space is 0000 to 7FFFH.

# Placing code in Program ROM -Example

| ROM Address | Machine Language | Assembly Language |
|---|---|---|
| 0000 | 7D25 | MOV R5,#25H |
| 0002 | 7F34 | MOV R7,#34H |
| 0004 | 7400 | MOV A,#0 |
| 0006 | 2D | ADD A,R5 |
| 0007 | 2F | ADD A,R7 |
| 0008 | 2412 | ADD A,#12H |
| 000A | 80FE | HERE: SJMP HERE |

| Address | Code | |
|---|---|---|
| 0000 | 7D | OPCODE |
| 0001 | 25 | OPERAND |
| 0002 | 7F | |
| 0003 | 34 | |
| 0004 | 74 | |
| 0005 | 00 | |
| 0006 | 2D | |
| 0007 | 2F | |
| 0008 | 24 | |
| 0009 | 12 | |
| 000A | 80 | |
| 000B | FE | |

# Memory organization of 8051- On chip RAM organization

8051 has 128 Bytes of RAM

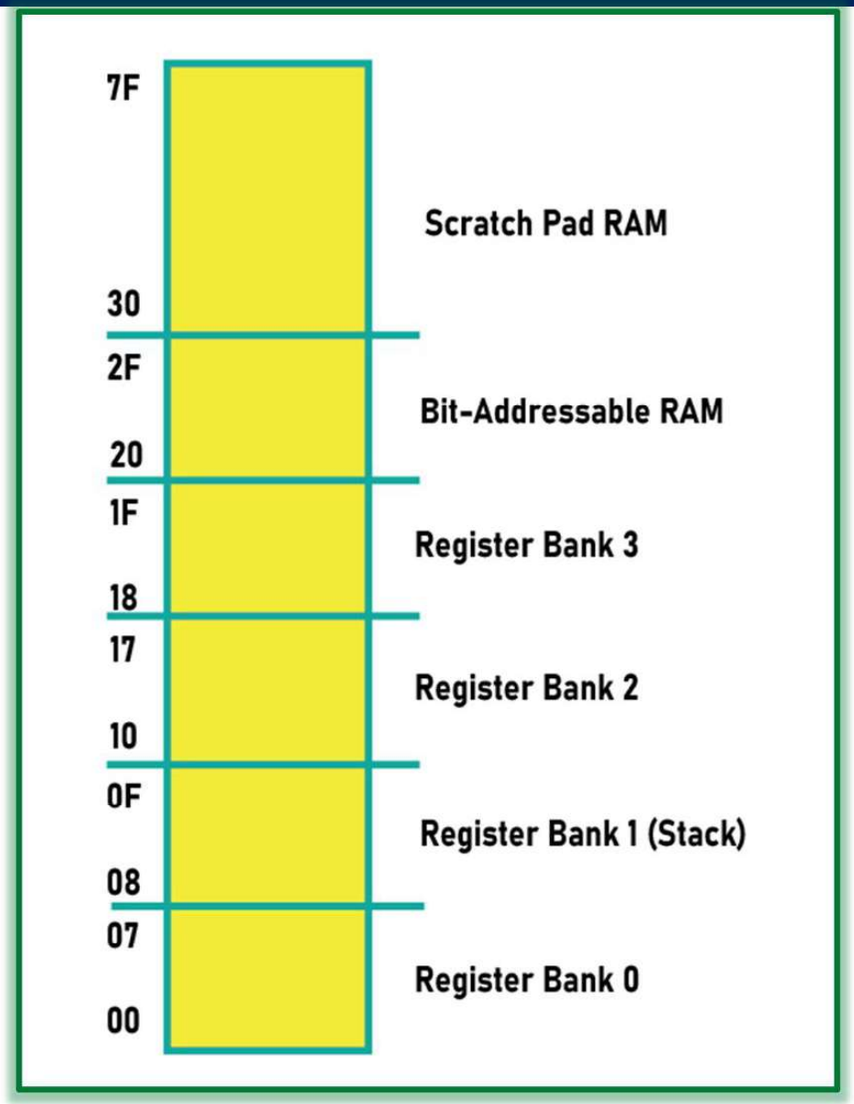
RAM allocation in 8051

The 128 bytes are divided into three different groups as follows:

1) A total of 32 bytes from locations 00 to 1F hex are set aside for register banks and the stack

2) A total of 16 bytes from locations 20H to 2FH are set aside for bit-addressable read/write memory

3) A total of 80 bytes from locations 30H to 7FH are used for read and write storage, called scratch pad (can be used to store variable data)



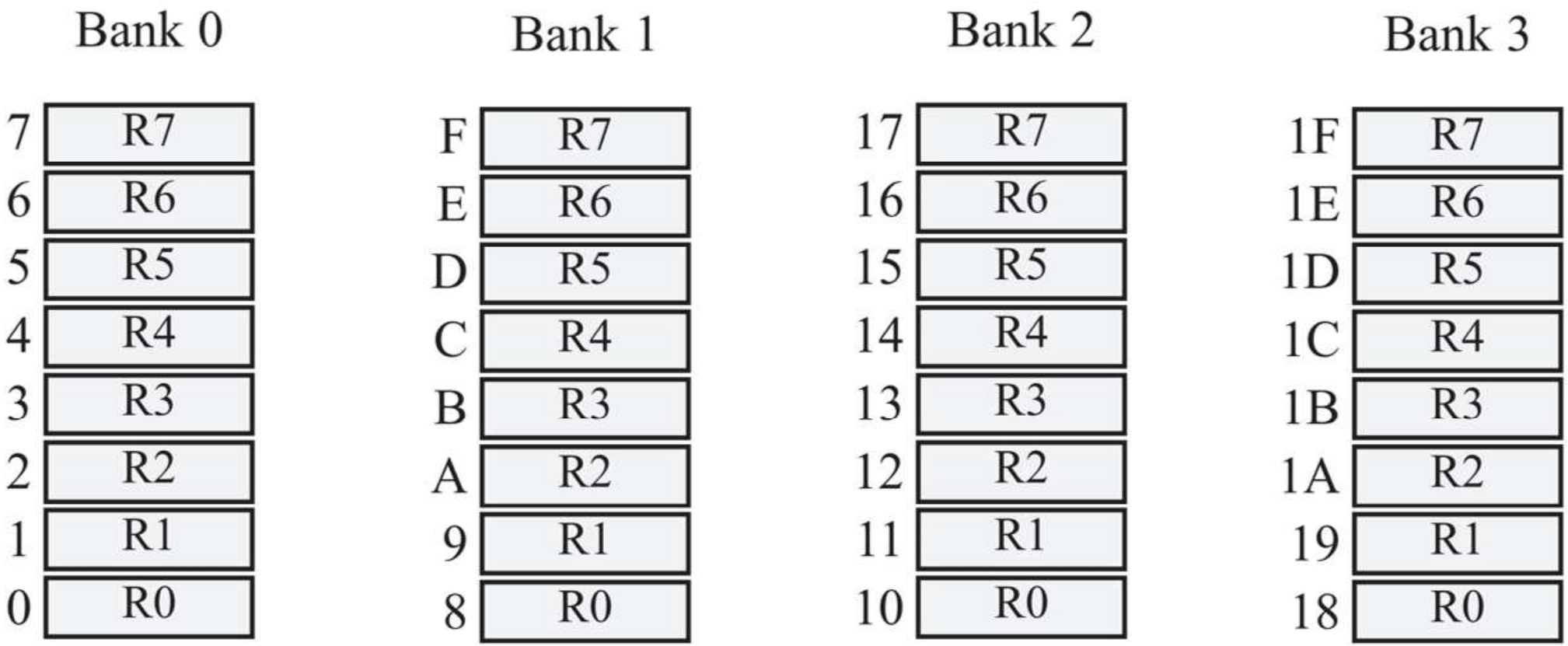
# Memory organization of 8051- On chip RAM organization

## 8051 Register Banks and their RAM Addresses

| Bank 0 | | Bank 1 | | Bank 2 | | Bank 3 | |
|---|---|---|---|---|---|---|---|
| 7 | R7 | F | R7 | 17 | R7 | 1F | R7 |
| 6 | R6 | E | R6 | 16 | R6 | 1E | R6 |
| 5 | R5 | D | R5 | 15 | R5 | 1D | R5 |
| 4 | R4 | C | R4 | 14 | R4 | 1C | R4 |
| 3 | R3 | B | R3 | 13 | R3 | 1B | R3 |
| 2 | R2 | A | R2 | 12 | R2 | 1A | R2 |
| 1 | R1 | 9 | R1 | 11 | R1 | 19 | R1 |
| 0 | R0 | 8 | R0 | 10 | R0 | 18 | R0 |

# Memory organization of 8051- On chip RAM organization

## 8051 Register Banks and their RAM Addresses



# Memory organization of 8051- On chip RAM organization

## Example- Predicting RAM content based on simple MOV instruction- MOV DESTIANTION, SOURCE

**State the contents of RAM locations after the following program:**

```
MOV R0,#99H    ;load R0 with value 99H
MOV R1,#85H    ;load R1 with value 85H
MOV R2,#3FH    ;load R2 with value 3FH
MOV R7,#63H    ;load R7 with value 63H
MOV R5,#12H    ;load R5 with value 12H
```

**After the execution of the above program we have the following:**

**RAM location 0 has value 99H**

**RAM location 1 has value 85H**

**RAM location 2 has value 3FH**

**RAM location 7 has value 63H**

**RAM location 5 has value 12H**

# Memory organization of 8051- On chip RAM organization

**Repeat Example using RAM addresses instead of register names.**

Default register bank-Bank 0

```
MOV 00,#99H     ;load R0 with value 99H
MOV 01,#85H     ;load R1 with value 85H
MOV 02,#3FH     ;load R2 with value 3FH
MOV 07,#63H     ;load R7 with value 63H
MOV 05,#12H     ;load R5 with value 12H
```

Direct addressing

# Program Status word (PSW)

**The program status word (PSW) register, also referred to as the flag register, is an 8-bit register.**

| CY | AC | F0 | RS1 | RS0 | OV | -- | P |
|---|---|---|---|---|---|---|---|

| | | |
|---|---|---|
| CY | PSW.7 | Carry flag. |
| AC | PSW.6 | Auxiliary carry flag. |
| F0 | PSW.5 | Available to the user for general purpose. |
| RS1 | PSW.4 | Register Bank selector bit 1. |
| RS0 | PSW.3 | Register Bank selector bit 0. |
| OV | PSW.2 | Overflow flag. |
| -- | PSW.1 | User-definable bit. |
| P | PSW.0 | Parity flag. Set/cleared by hardware each instuction cycle to indicate an odd/even number of 1 bits in the accumulator. |

| RS1 | RS0 | Register Bank | Address |
|---|---|---|---|
| 0 | 0 | 0 | 00H - 07H |
| 0 | 1 | 1 | 08H - 0FH |
| 1 | 0 | 2 | 10H - 17H |
| 1 | 1 | 3 | 18H - 1FH |

- The PSW Register contains that status bits that reflect the current status of the CPU
- Only 6 bits are used by 8051
- Two unsed bits are user defined

Bits of Program Status word
4-Conditional flags
2-bits for Register bank select
2bits-user defined flags

# Special Function Register-PSW (PROGRAM STATUS WORD)

- **CY, the carry flag – This carry flag is set (1) whenever there is a carry out from the D7 bit. It is affected after an 8-bit addition or subtraction operation. It can also be reset to 1 or 0 directly by an instruction such as "SETB C" and "CLR C" where "SETB" stands for set bit carry and "CLR" stands for clear carry.**

- **AC, auxiliary carry flag – If there is a carry from D3 and D4 during an ADD or SUB operation, the AC bit is set; otherwise, it is cleared. It is used for the instruction to perform binary coded decimal arithmetic.**

- **P, the parity flag – The parity flag represents the number of 1's in the accumulator register only. If the A register contains odd number of 1's, then P = 1; and for even number of 1's, P = 0.**

- **OV, the overflow flag – This flag is set whenever the result of a signed number operation is too large causing the high-order bit to overflow into the sign bit. It is used only to detect errors in signed arithmetic operations.**
- **The OV flag is set if there is a carry from D6-D7 or from D7 to D8 but not both.**

  **Hence the conditions are either**
  - **carry from D6-D7 or from D7 (CY=0) Or**
  - **carry from D7 to D8 (CY= 1).**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CY | AC | F0 | RS1 | RS0 | OV | -- | P |

# Instructions that affect PSW

| Instruction | CY | OV | AC |
|---|---|---|---|
| ADD | X | X | X |
| ADDC | X | X | X |
| SUBB | X | X | X |
| MUL | 0 | X | |
| DIV | 0 | X | |
| DA | X | | |
| RRC | X | | |
| RLC | X | | |
| SETB C | 1 | | |

| Instruction | CY | OV | AC |
|---|---|---|---|
| CLR C | 0 | | |
| CPL C | X | | |
| ANL C, bit | X | | |
| ANL C, /bit | X | | |
| ORL C, bit | X | | |
| ORL C, /bit | X | | |
| MOV C, bit | X | | |
| CJNE | X | | |

X can be 0 or 1

# How to switch register banks

| | RS1 (PSW.4) | RS0 (PSW.3) |
|---|---|---|
| Bank 0 | 0 | 0 |
| Bank 1 | 0 | 1 |
| Bank 2 | 1 | 0 |
| Bank 3 | 1 | 1 |

# Example- status of PSW- Flag bits after addition

**Show the status of the CY, AC, and P flags after the addition of 38H and 2FH in the following instructions.**

```
MOV A,#38H
ADD A,#2FH   ;after the addition A=67H, CY=0
```

| | |
|---|---|
| 38 | 00111000 |
| + 2F | 00101111 |
| 67 | 01100111 |

**CY = 0 since there is no carry beyond the D7 bit.**
**AC = 1 since there is a carry from the D3 to the D4 bit.**
**P = 1 since the accumulator has an odd number of 1s (it has five 1s).**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CY | AC | F0 | RS1 | RS0 | OV | -- | P |

# Example -status of PSW flags

**Show the status of the CY, AC, and P flags after the addition of 9CH and 64H in the following instructions.**

**MOV A,#88H** 1 0001000

**ADD A,#93H** 1 0010011

1 00011011

**CY =1 Carry from D7 bit**

**AC=0 no auxiliary carry from D3to D4**

**P=0 Even number of 1's**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CY | AC | F0 | RS1 | RS0 | OV | -- | P |

# Determine the status of PSW register conditional flags

Signed arithmetic

**MOV A, #-128 ; A=1000 0000 (A=80H)**
**MO R4,#-2 ; R4=1111 1110 (R4=FEH)**
**ADD A,R4; A=0111 1110 (A= 7Eh =+126; invalid)**

**-128 1000 0000**
**-2 1111 1110**
**-130 0111 1110 OV=1 (Carry from D7)**

7Eh= 7x $16^1$ +Ex$16^0$ = 112+14 =+126

**CY =1 Carry from D7 bit**
**AC=0 no auxiliary carry from D3to D4**
**P=0 Even number of 1's**
**OV=1**

128 = 80h
1000 0000 – Take 1's complement
0111 1111
1 -1-s complement +1
1000 0000 - -2's complement

2= 10h
0000 0010 – Take 1's complement
1111 1101- 1's complement
1- 1's complement +1
1111 1110 – 2's complement

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CY | AC | F0 | RS1 | RS0 | OV | -- | P |

# Determine the status of PSW conditional flags

Signed arithmetic

**MOV A, #-2 ;**
**MO R1,#-5 ;**
**ADD A,R1;**
**-2 1111 1110**
**-5 1111 1011**
**-7 1111 1001 OV=0 (Carry from both D6 and D7)**

A=1111 1110 (A=FEH)
R1=1111 1011 (R1=FBH)
A= 1111 1001 (A= F9h =-7 correct, OV=0)

7 0000 0111
1111 1000
1
1111 1001

**CY =1 Carry from D7 bit**
**AC=1 no auxiliary carry from D3to D4**
**P=0 Even number of 1's**
**OV=0**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CY | AC | F0 | RS1 | RS0 | OV | -- | P |

# Home work – Status of conditional flags of PSW

**MOV A, #+7;**
**MOV R1,#+18;**
**ADD A,R1;**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| CY | AC | F0 | RS1 | RS0 | OV | -- | P |

# Status of PSW- Home work

**Show the status of the CY, AC, and P flags after the addition of 9CH and 64H in the following instructions.**

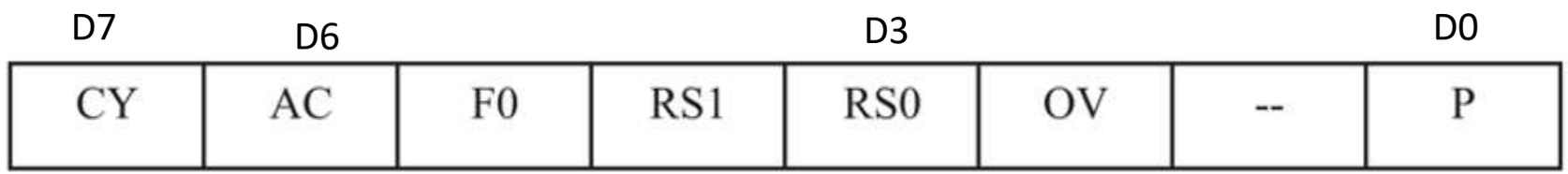
**MOV A,#9CH**
**ADD A,#64H**

# Status of PSW- Home work

**Show the content of PSW register after the execution of following instructions.**

**MOV A,#OBFH**
**ADD A,#1BH**

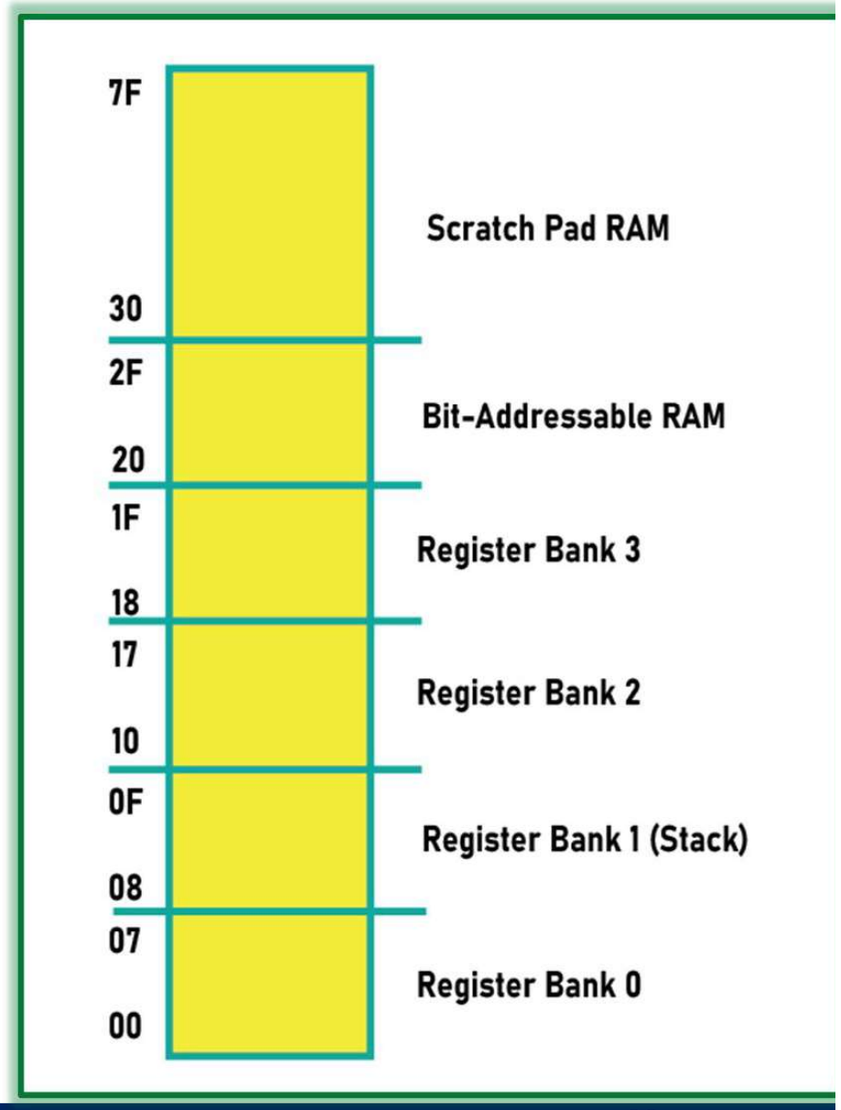
PSW.0:
PSW.1:
PSW.2:
PSW.3:
PSW.4:
PSW.5:
PSW.6:
PSW.7:



# Example –state the content of RAM location

State the contents of the RAM locations after the following program:

```
SETB PSW.4      ;select bank 2
MOV R0,#99H     ;load R0 with value 99H
MOV R1,#85H     ;load R1 with value 85H
MOV R2,#3FH     ;load R2 with value 3FH
MOV R7,#63H     ;load R7 with value 63H
MOV R5,#12H     ;load R5 with value 12H
```




| CY | AC | F0 | RS1 | RS0 | OV | -- | P |
|---|---|---|---|---|---|---|---|

# Example -state content of RAM location

**State the contents of the RAM locations after the following program:**

```
SETB PSW.4    ;select bank 2
MOV R0,#99H   ;load R0 with value 99H
MOV R1,#85H   ;load R1 with value 85H
MOV R2,#3FH   ;load R2 with value 3FH
MOV R7,#63H   ;load R7 with value 63H
MOV R5,#12H   ;load R5 with value 12H
```

# Registers in 8051

These are 32 general purpose registers address from 00h to 1Fh.

- The address range of Register Bank 0 ( 00 h to 07 h)
- The address range of Register Bank 1 ( 08 h to F h)
- The address range of Register Bank 2 ( 10 h to 17 h)
- The address range of Register Bank 3 ( 18 h to 1F h)

Special function registers-A Special Function Register (SFR) is a register that controls or monitors the various functions of controller (80h-FFh address range)

- Math or CPU Registers: A and B Register
- Status Register: PSW (Program Status Word) Register
- Pointer Registers: DPTR (Data Pointer – DPL, DPH) and SP (Stack Pointer) Registers
- I/O Port Registers: P0 (Port 0), P1 (Port 1), P2 (Port 2) and P3 (Port 3) Registers
- Peripheral Control Registers: PCON, SCON, TCON, TMOD, IE and IP Registers
- Peripheral Data Registers: TL0, TH0, TL1, TH1 and SBUF Registers

# Registers in 8051 General purpose and special function registers

These are 32 general purpose registers address from 00h to 1Fh.

- The address range of Register Bank 0 ( 00 h to 07 h)
- The address range of Register Bank 1 ( 08 h to F h)
- The address range of Register Bank 2 ( 10 h to 17 h)
- The address range of Register Bank 3 ( 18 h to 1F h)

Special function registers

- Math or CPU Registers: A and B Register
- Status Register: PSW (Program Status Word) Register
- Pointer Registers: DPTR (Data Pointer – DPL, DPH) and SP (Stack Pointer) Registers
- I/O Port Registers: P0 (Port 0), P1 (Port 1), P2 (Port 2) and P3 (Port 3) Registers
- Peripheral Control Registers: PCON, SCON, TCON, TMOD, IE and IP Registers
- Peripheral Data Registers: TL0, TH0, TL1, TH1 and SBUF Registers

# Special function registers

- PERIPHERAL CONTROL REGISTERS
- SCON (Serial Control)  (8 bit)
- The Serial Control or SCON SFR is used to control the 8051 Microcontroller's Serial Port.
- PCON (Power Control) (8 bit)
- The PCON or Power Control register, as the name suggests is used to control the 8051 Microcontroller's Power Modes
- IE (Interrupt Enable) (8 bit)
- The IE or Interrupt Enable Register is used to enable or disable individual interrupts.
- If a bit is SET, the corresponding interrupt is enabled and if the bit is cleared, the interrupt is disabled. The Bit7 of the IE register i.e. EA bit is used to enable or disable all the interrupts
- IP (Interrupt Priority) (8 bit)
- The IP or Interrupt Priority Register is used to set the priority of the interrupt as High or Low

# Special function registers

- The Counters and Timers in 8051 microcontrollers contain two special function registers:
- TCON (Timer Control) (8 bit)
- Timer Control or TCON Register is used to start or stop the Timers of 8051
- TMOD (Timer Mode) (8 bit)
- The TMOD or Timer Mode register or SFR is used to set the Operating Modes of the Timers T0 and T1. The lower four bits are used to configure Timer0 and the higher four bits are used to configure Timer1.

# Special function registers

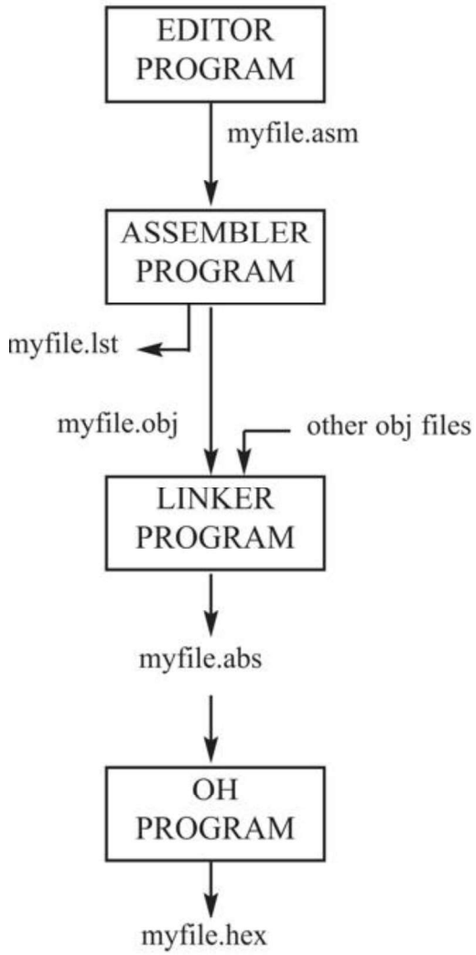
- **Peripheral Data Registers**
- **SBUF (Serial Data Buffer)** (8 bit)
- The Serial Buffer or SBUF register is used to hold the serial data while transmission or reception.
- TL0/TH0 (Timer 0 Low/High) (8 bit)
- The Timer 0 consists of two SFRs: TL0 and TH0. The TL0 is the lower byte and the TH0 is the higher byte and together they form a 16-bit Timer0 Register.
- TL1/TH1 (Timer 1 Low/High) (8 bit)
- The TL1 and TH1 are the lower and higher bytes of the Timer 1.

# Special function registers

- I/O Port Registers(P0, P1, P2 and P3) (8 bit)
- The 8051 Microcontroller four Ports which can be used as Input and/or Output.
- These four ports are P0, P1, P2 and P3. Each Port has a corresponding register with same names (the Port Registers are also P0, P1, P2 and P3)

# 8051 assembly language programming and instruction set

Steps to create a program



**An Assembly language instruction consists of four fields:**

**[label:] Mnemonic [operands] [;comment]**

# 8051 assembly language programming- Directives

ORG (origin)

- The ORG directive is used to indicate the beginning of the address from which the 8051 Microcontroller will start executing the code.
- The number that comes after ORG can be either in hex and decimal
- If the number is not followed by H, it is decimal and the assembler will convert it to hex

END

- This indicates to the assembler the end of the source (asm) file
- The END directive is the last line of an 8051 program
- Mean that in the code anything after the END directive is ignored by the assembler

EQU (equate)

- This is used to define a constant without occupying a memory location
- The EQU directive does not set aside storage for a data item but associates a
- constant value with a data label
- When the label appears in the program, its constant value will be substituted for the label

```
COUNT EQU 25
... ....
MOV R3, #COUNT
```

# 8051 assembly language programming- Directives

**DB Define byte**

- The DB directive is the most widely used data directive in the assembler
- It is used to define the 8-bit data
- When DB is used to define data, the numbers can be in decimal, binary, hex, ASCII formats

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

# 8051 Instruction set

| DATA TRANSFER | ARITHMETIC | LOGICAL | BOOLEAN | PROGRAM BRANCHING |
|---|---|---|---|---|
| MOV | ADD | ANL | CLR | LJMP |
| MOVC | ADDC | ORL | SETB | AJMP |
| MOVX | SUBB | XRL | MOV | SJMP |
| PUSH | INC | CLR | JC | JZ |
| POP | DEC | CPL | JNC | JNZ |
| XCH | MUL | RL | JB | CJNE |
| XCHD | DIV | RLC | JNB | DJNZ |
| | DA A | RR | JBC | NOP |
| | | RRC | ANL | LCALL |
| | | SWAP | ORL | ACALL |
| | | | CPL | RET |
| | | | | RETI |
| | | | | JMP |

# 8051 Instruction set- Data transfer instructions

| Mnemonic | Description |
|---|---|
| MOV | Move Data |
| MOVC | Move Code |
| MOCX | Move External Data |
| PUSH | Move Data to Stack |
| POP | Copy Data from Stack |
| XCH | Exchange Data between two Registers |
| XCHD | Exchange Lower Order Data between two Registers |

# 8051 Instruction set- Data transfer instructions

| Mnemonic | Instruction | Description |
|---|---|---|
| MOV | A, #Data | A ← Data |
| | A, Rn | A ← Rn |
| | A, Direct | A ← (Direct) |
| | A, @Ri | A ← @Ri |
| | Rn, #Data | Rn ← data |
| | Rn, A | Rn ← A |
| | Rn, Direct | Rn ← (Direct) |
| | Direct, A | (Direct) ← A |
| | Direct, Rn | (Direct) ← Rn |
| | Direct1, Direct2 | (Direct1) ← (Direct2) |
| | Direct, @Ri | (Direct) ← @Ri |
| | Direct, #Data | (Direct) ← #Data |
| | @Ri, A | @Ri ← A |
| | @Ri, Direct | @Ri ← Direct |
| | @Ri, #Data | @Ri ← #Data |
| | DPTR, #Data16 | DPTR ← #Data16 |
| | | |
| MOVC | A, @A+DPTR | A ← Code Pointed by A+DPTR |
| | A, @A+PC | A ← Code Pointed by A+PC |
| | A, @Ri | A ← Code Pointed by Ri (8-bit Address) |
| | | |
| MOVX | A, @DPTR | A ← External Data Pointed by DPTR |
| | @Ri, A | @Ri ← A (External Data 8-bit Addr) |
| | @DPTR, A | @DPTR ← A (External Data 16-bit Addr) |
| | | |
| PUSH | Direct | Stack Pointer SP ← (Direct) |
| | | |
| POP | Direct | (Direct) ← Stack Pointer SP |

| | | |
|---|---|---|
| XCH | Rn | Exchange ACC with Rn |
| | Direct | Exchange ACC with Direct Byte |
| | @Ri | Exchange ACC with Indirect RAM |
| | | |
| XCHD | A, @Ri | Exchange ACC with Lower Order Indirect RAM |

# Instruction set-Data transfer instruction
# MOV

MOV destination, source ;copy source to destination.

The instruction tells the CPU to move (in reality, COPY) the source operand to the destination operand

```
MOV A,#55H ;load value 55H into reg. A
MOV R0,A ;copy contents of A into R0 ;(now A=R0=55H)
MOV R1,A ;copy contents of A into R1 ;(now A=R0=R1=55H)
MOV R2,A ;copy contents of A into R2 ;(now A=R0=R1=R2=55H)
MOV R3,#95H ;load value 95H into R3 ;(now R3=95H)
MOV A,R3 ;copy contents of R3 into A ;now A=R3=95H
```

# Instruction set

MOV C

The MOVC instruction moves a byte from the code or program memory to the accumulator

**MOVC A, @A+DPTR**

| C | AC | F0 | RS1 | RS0 | OV | | P |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

**Operation**

```
MOVC
A = (A + DPTR)
```

**Example**

```
MOVC A, @A+DPTR
```

# Instruction set

**MOV X**

The MOVX instruction transfers data between the accumulator and external data memory. External memory may be addressed via 16-bits in the DPTR register or via 8-bits in the R0 or R1 registers.

MOVX A, @DPTR

| C | AC | F0 | RS1 | RS0 | OV | | P |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

**Bytes** 1

**Operation**

```
MOVX
A = (DPTR)
```

**Example**

```
MOVX A, @DPTR
```

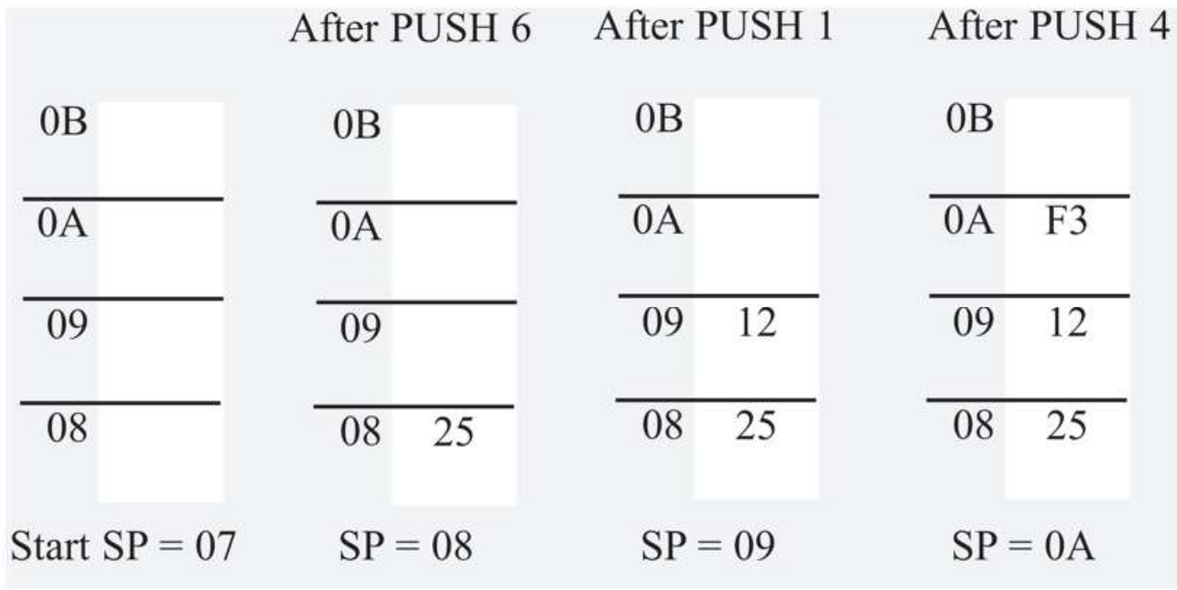
# Instruction set- Stack operations

- Pushing into the Stack
- In the 8051, the stack pointer (SP) points to the last used location of the stack.
- When data is pushed onto the stack, the stack pointer (SP) is incremented by 1.
- When PUSH is executed, the contents of the register are saved on the stack and SP is incremented by 1.
- To push the registers onto the stack, we must use their RAM addresses. For example, the instruction "PUSH 1" pushes register R1 onto the stack.

# Instruction set- Stack operations –PUSH

- Show the stack and stack pointer for the following. Assume the default stack area and register 0 is selected.
- MOV R6,#25H
- MOV R1,#12H
- MOV R4,#0F3H
- PUSH 6
- PUSH 1
- PUSH 4

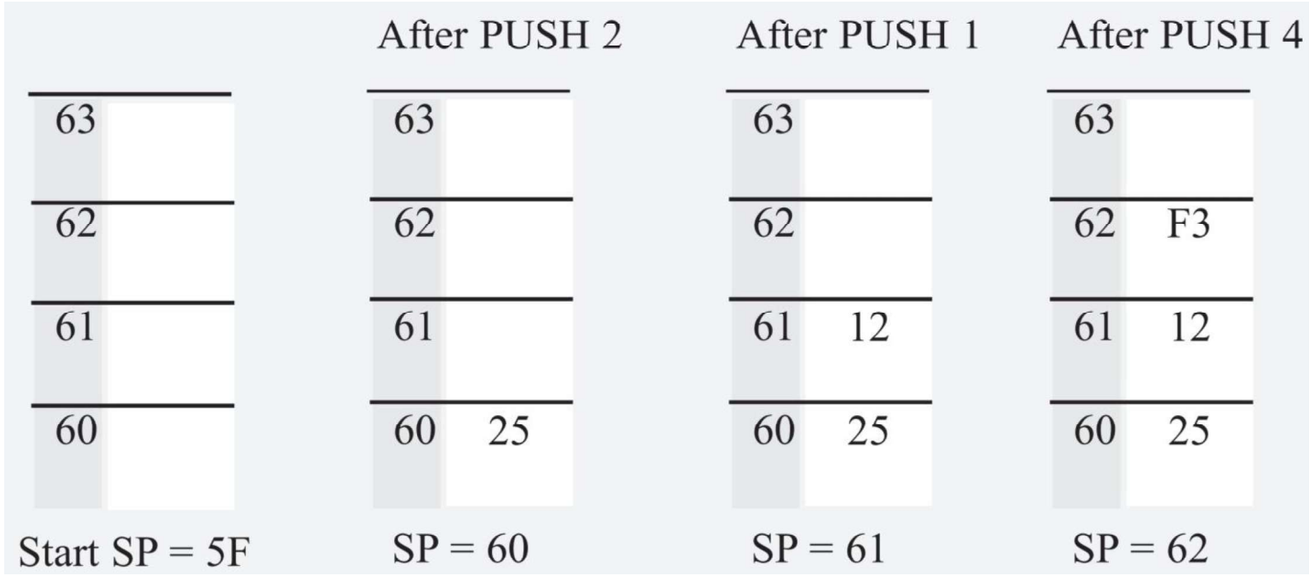
**Solution:**



# Instruction set- Stack operations –Stack and bank 1 conflict

- Show the stack and stack pointer for the following instructions.

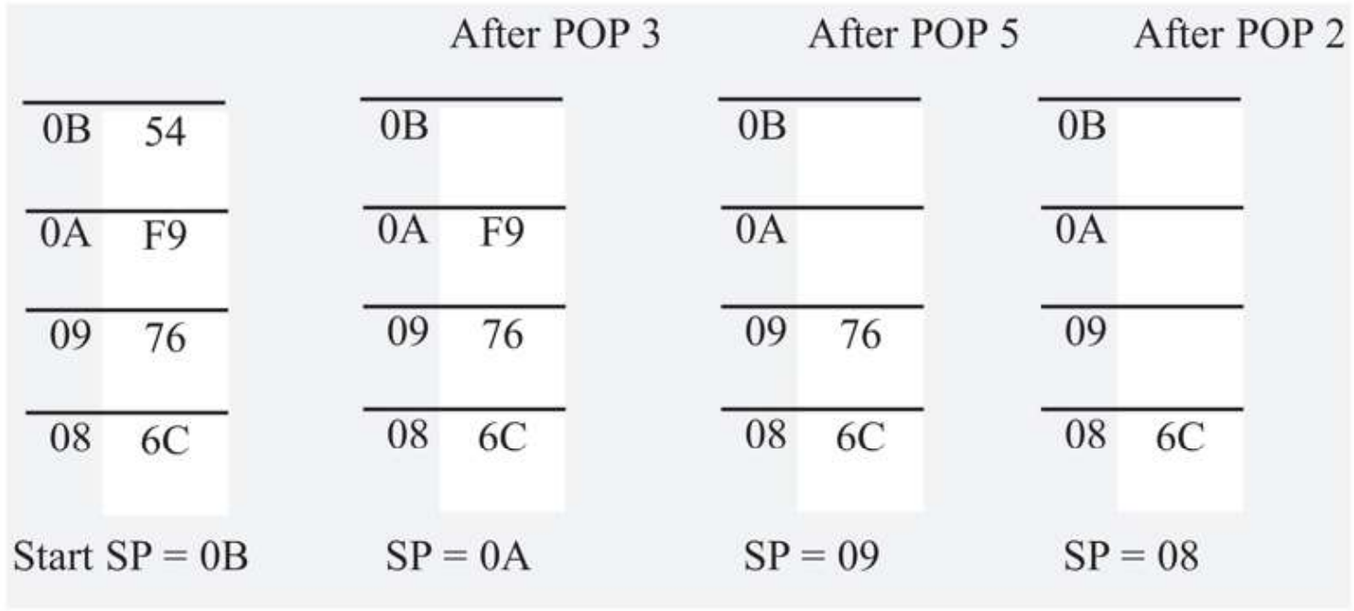
- MOV SP,#5FH ;make RAM location 60H
- ;first stack location
- MOV R2,#25H
- MOV R1,#12H
- MOV R4,#0F3H
- PUSH 2
- PUSH 1
- PUSH 4



# Instruction set- Stack operations - POP

- Examining the stack, show the contents of the registers and SP after execution of the following instructions. All values are in hex.
- POP 3 ;POP stack into R3
- POP 5 ;POP stack into R5
- POP 2 ;POP stack into R2

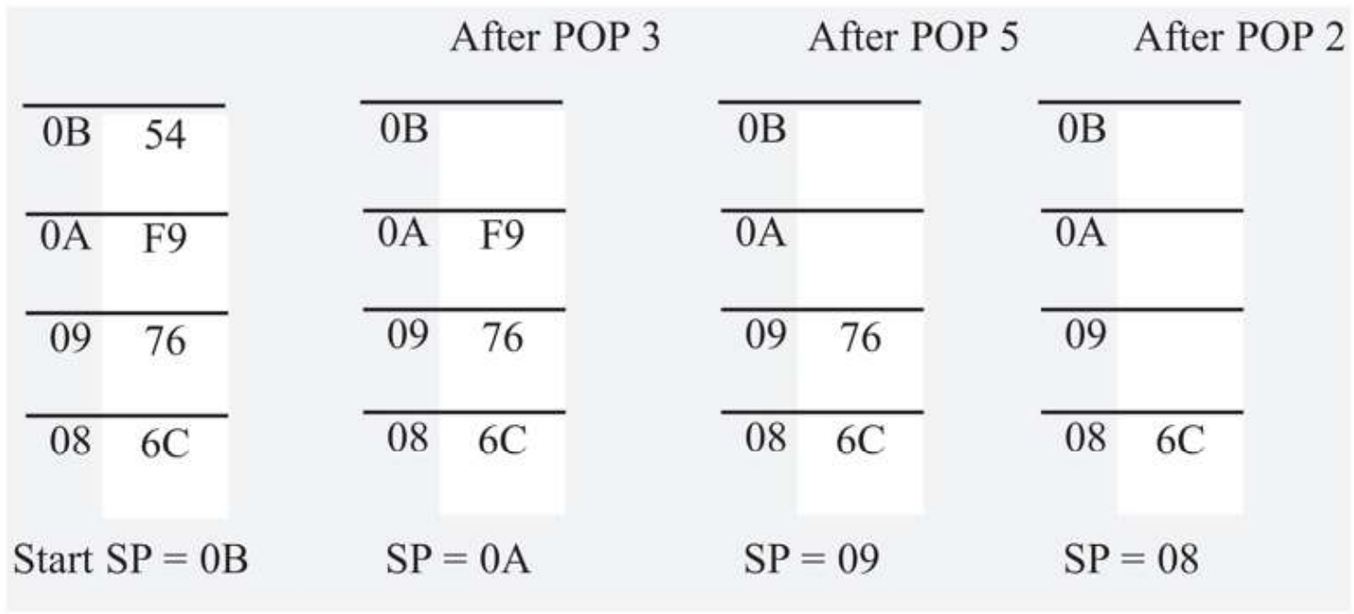
**Solution:**



# Instruction set- Stack operations - POP

- Examining the stack, show the contents of the registers and SP after execution of the following instructions. All values are in hex.
- POP 3 ;POP stack into R3
- POP 5 ;POP stack into R5
- POP 2 ;POP stack into R2

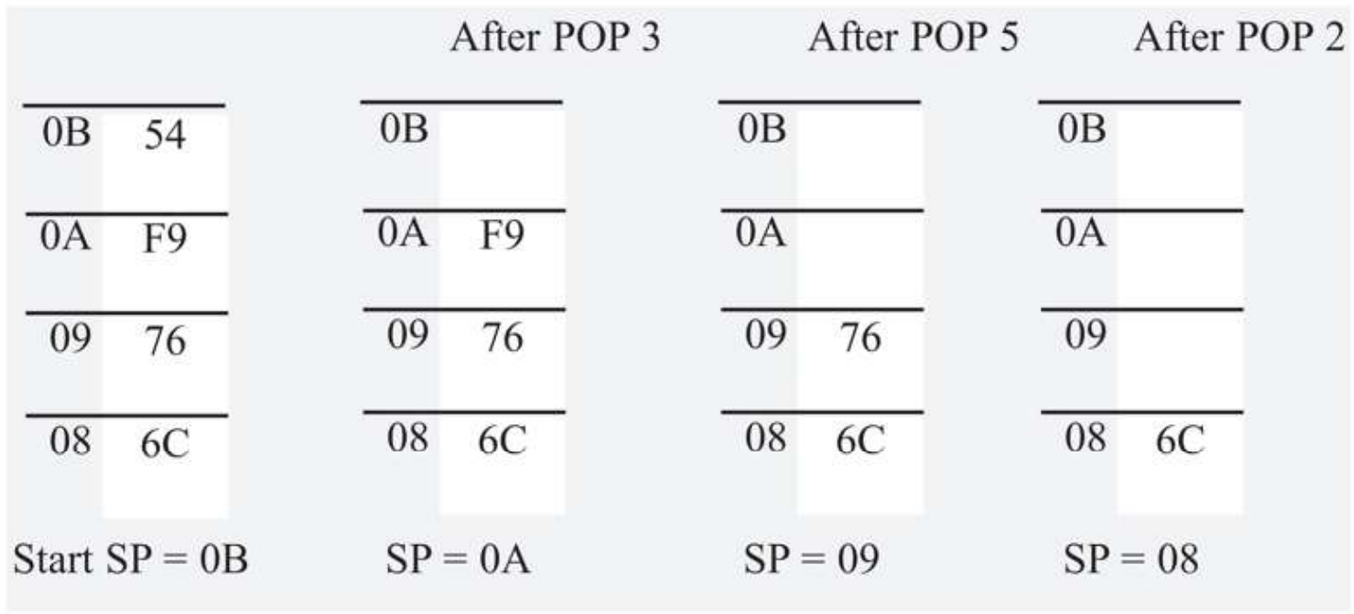
**Solution:**



# Instruction set- Stack operations - POP

- Examining the stack, show the contents of the registers and SP after execution of the following instructions. All values are in hex.
- POP 3 ;POP stack into R3
- POP 5 ;POP stack into R5
- POP 2 ;POP stack into R2

**Solution:**



# Addressing modes

- In 8051 There are six types of addressing modes.
- Immediate Addressing Mode
- Register Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Indexed Addressing Mode

Accessing memories

# Immediate addressing mode

- In this Immediate Addressing Mode, the data is provided in the instruction itself. The data is provided immediately after the opcode
- MOV A, #0AFH; when the data is starting with A to F, the data should be preceded by 0.
- MOV R3, #45H;
- MOV DPTR, #FE00H;  DPTR is loaded with external memory location

# Register addressing mode

- In the register addressing mode the source or destination data should be present in a register (R0 to R7).
- MOV A, R5;
- MOV R0, A;
- Source and destination  size should match
- MOV DPTR, A will give an error.Why?
- We can move beteeen  accumulator and Rn.No movement between Rn registers is allowed.
- MOV R4,R7 is invalid

# Direct Addressing mode

- The direct addressing mode most often used to access RAM locations 30 – 7FH
- MOV R0,40h,MOV content of RAM location 40h to R0
- MOV 56h, A; Mov content of A to RAM LOCATION 56h
- MOV R4,7Fh; Mov content of RAM location 7F h to R4.

- RAM locations 0 to 7 are allocated to bank 0 registers R0-R7
- MOV A,4 ; SAME AS
- MOV A,R4; COPY R4 TO A

# Direct Addressing mode

- MOV R2,#5; R2 with value 5
- MOV A,2;Copy R2 to A (A=R2=05)
- MOV B,2 ;Copy R2 to B
- MOV 7,2 ;Copy R2 to R7

# SFR Registers and their addresses

- SFR address range 80H-FFh since 00-7F are addresses of RAM memory inside 8051
- Not all address space of 80H –FFh are used by SFR .The unused locations  80h-FF are reserved and cannot be used  by programmer.

# SFR Registers and their addresses

- SFR address range 80H-FFh since 00-7F are addresses of RAM memory inside 8051
- Not all address space of 80H –FFh are used by SFR .The unused locations  80h-FF are reserved and cannot be used  by programmer.

# SFR Registers and their addresses

| Symbol | Name | Address |
|---|---|---|
| ACC* | Accumulator | 0E0H |
| B* | B register | 0F0H |
| PSW* | Program status word | 0D0H |
| SP | Stack pointer | 81H |
| DPTR | Data pointer 2 bytes | |
| DPL | Low byte | 82H |
| DPH | High byte | 83H |
| P0* | Port 0 | 80H |
| P1* | Port 1 | 90H |
| P2* | Port 2 | 0A0H |
| P3* | Port 3 | 0B0H |
| IP* | Interrupt priority control | 0B8H |
| IE* | Interrupt enable control | 0A8H |
| TMOD | Timer/counter mode control | 89H |
| TCON* | Timer/counter control | 88H |

# SFR Registers and their addresses

| Symbol | Name | Address |
|---|---|---|
| T2CON* | Timer/counter 2 control | 0C8H |
| T2MOD | Timer/counter mode control | 0C9H |
| TH0 | Timer/counter 0 high byte | 8CH |
| TL0 | Timer/counter 0 low byte | 8AH |
| TH1 | Timer/counter 1 high byte | 8DH |
| TL1 | Timer/counter 1 low byte | 8BH |
| TH2 | Timer/counter 2 high byte | 0CDH |
| TL2 | Timer/counter 2 low byte | 0CCH |
| RCAP2H | T/C 2 capture register high byte | 0CBH |
| RCAP2L | T/C 2 capture register low byte | 0CAH |
| SCON* | Serial control | 98H |
| SBUF | Serial data buffer | 99H |
| PCON | Power control | 87H |

* Bit-addressable

# Example

- Write code to send 55H to ports P1 and P2, using (a) their names, (b) their addresses.

(a)
```
MOV A,#55H      ;A=55H
MOV P1,A        ;P1=55H
MOV P2,A        ;P2=55H
```

(b) P1 address = 90H; P2 address = A0H
```
MOV A,#55H          ;A=55H
MOV 90H,A       ;P1=55H
MOV 0A0H,A          ;P2=55H
```

# Example

- Show the code to push R5, R6, and A onto the stack and then pop them back them into R2, R3, and B, where register B = register A, R2 = R6, and R3 = R5.

```
PUSH 05          ;push R5 onto stack
PUSH 06          ;push R6 onto stack
PUSH 0E0H        ;push register A onto stack
POP  0F0H        ;pop top of stack into register B
                 ;now register B = register A
POP  02          ;pop top of stack into R2
                 ;now R2 = R6
POP  03          ;pop top of stack into R3
                 ;now R3 = R5
```

# Example

- Show the code to push R5, R6, and A onto the stack and then pop them back them into R2, R3, and B, where register B = register A, R2 = R6, and R3 = R5.

```
PUSH 05          ;push R5 onto stack
PUSH 06          ;push R6 onto stack
PUSH 0E0H        ;push register A onto stack
POP  0F0H        ;pop top of stack into register B
                 ;now register B = register A
POP  02          ;pop top of stack into R2
                 ;now R2 = R6
POP  03          ;pop top of stack into R3
                 ;now R3 = R5
```

# Check if correct or not?

- PUSH A is invalid
- Pushing the accumulator onto the stack must be coded as PUSH 0E0H

# Register Indirect

- A register is used as a pointer to the data
- Only register R0 and R1 are used for this purpose
- R2 – R7 cannot be used to hold the address of an operand located in RAM
- When R0 and R1 hold the addresses of RAM locations, they must be preceded by the "@" sign
- MOV A,@R0 ;move contents of RAM ;whose address is held by R0 into A
- MOV @R1,B ;move contents of B into RAM ;whose address is held by R1
- Since R0 and R1 are 8 bits wide, their use is limited to access any information in the internal RAM
- For external access we need 16-bit pointer
- In such case, the DPTR register is used

# Indexed Addressing mode

- Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM
- The instruction used for this purpose is

  MOVC A,@A+DPTR
- Use instruction MOVC, “C” means -The contents of A are added to the 16-bit register DPTR to form the 16-bit address of the needed data

# Example

- Write a program to copy the value 55H into RAM memory locations 40H to 45H using
- (a) direct addressing mode, (b) register indirect addressing mode
- Direct addressing mode

```
MOV A,#55H   ;load A with value 55H
MOV 40H,A    ;copy A to RAM location 40H
MOV 41H,A    ;copy A to RAM location 41H
MOV 42H,A    ;copy A to RAM location 42H
MOV 43H,A    ;copy A to RAM location 43H
MOV 44H,A    ;copy A to RAM location 44H
```

# Example

- Write a program to copy the value 55H into RAM memory locations 40H to 45H using
- (b) register indirect addressing mode

```
MOV A,#55H      ;load A with value 55H
MOV R0,#40H      ;load the pointer. R0=40H
MOV @R0,A        ;copy A to RAM location R0 points to
INC R0           ;increment pointer. Now R0=41H
MOV @R0,A        ;copy A to RAM location    R0 points to
INC R0           ;increment pointer. Now R0=42H
MOV @R0,A        ;copy A to RAM location    R0 points to
INC R0           ;increment pointer. Now R0=43H
MOV @R0,A        ;copy A to RAM location R0 points to
INC R0           ;increment pointer. Now R0=44H
MOV @R0,A
```

# Home work

- Check the following instructions are correct or not.If they are incorrect give the reason
- PUSH B
- MOV A,@R3

- Write a program to copy the value 25H into RAM memory locations 50H to 55H using
- (a) direct addressing mode (b) register indirect addressing mode

# Addressing modes-Summary

- In 8051 There are five types of addressing modes.

| | Example |
|---|---|
| • Immediate Addressing Mode | MOV Rn, #value |
| • Register Addressing Mode | MOV A, Rn ; No direct transfer between registers |
| • Direct Addressing Mode | MOV A, 50H ; Use SFR address;eg PUSH 0EH |
| • Register Indirect Addressing Mode | MOV A,@R0 ;only R0 AND R1 CAN BE USED |
| • Indexed Addressing Mode | MOVC A,@A+DPTR |

# LOOPING INSIDE 8051

- Repeating a sequence of instructions a certain number of times is called a loop
- Loop action is performed by following exmaple
- DJNZ reg, Label
- The register is decremented
- If it is not zero, it jumps to the target address referred to by the label
- Prior to the start of loop, the register is loaded with the counter for the number of repetitions
- Counter can be R0 – R7 or RAM location

# LOOPING INSIDE 8051

- Repeating a sequence of instructions a certain number of times is called a loop
- Loop action is performed by following exmaple
- DJNZ reg, Label
- The register is decremented
- If it is not zero, it jumps to the target address referred to by the label
- Prior to the start of loop, the register is loaded with the counter for the number of repetitions
- Counter can be R0 – R7 or RAM location

# LOOPING INSIDE 8051

- Write a program to
- (a) clear ACC, then
- (b) add 3 to the accumulator ten times

```
        MOV   A,#0        ;A=0, clear ACC
        MOV   R2,#10      ;load counter R2=10
AGAIN:  ADD   A,#03       ;add 03 to ACC
        DJNZ  R2,AGAIN    ;repeat until R2=0(10 times)
        MOV   R5,A        ;save A in R5
```

# LOOPING

- What is the maximum number of times that the loop in Example can be repeated?

```
        MOV   A,#0        ;A=0, clear ACC
        MOV   R2,#10      ;load counter R2=10
AGAIN:  ADD   A,#03       ;add 03 to ACC
        DJNZ  R2,AGAIN    ;repeat until R2=0(10 times)
        MOV   R5,A        ;save A in R5
```

**Since R2 holds the count and R2 is an 8-bit register, it can hold a maximum of FFH (255 decimal); therefore, the loop can be repeated a maximum of 256 times.**

# LOOP INSIDE LOOP

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

Since 700 is larger than 255 (the maximum capacity of any register), we use two registers to hold the count. The following code shows how to use R2 and R3 for the count.

```
             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 it 70 times (inner loop)
             DJNZ  R3,NEXT
```

In this program, R2 is used to keep the inner loop count. In the instruction "DJNZ R2,AGAIN", whenever R2 becomes 0 it falls through and "DJNZ R3,NEXT" is executed. This instruction forces the CPU to load R2 with the count 70 and the inner loop starts again. This process will continue until R3 becomes zero and the outer loop is finished.

# LOOP

- Write a program to copy a block of 10 bytes of data from RAM locations starting at 35H to RAM locations starting at 60H.

```
        MOV  R0,#35H    ;source pointer
        MOV  R1,#60H    ;destination pointer
        MOV  R3,#10     ;counter
BACK:MOV A,@R0;get a byte from source
        MOV  @R1,A;copy it to destination
        INC  R0         ;increment source pointer
        INC  R1         ;increment destination pointer
        DJNZ R3,BACK    ;keep doing it for all ten bytes
```

# Homework

- Write a program to copy the value 55H into RAM memory locations 40H to 45H using a loop.
- Write a program to clear 16 RAM locations starting at RAM address 60H.

# Conditional Jump Instructions

| Instruction | Action |
|---|---|
| JZ | Jump if A = 0 |
| JNZ | Jump if A ≠ 0 |
| DJNZ | Decrement and jump if register ≠ 0 |
| CJNE A, data | Jump if A ≠ data |
| CJNE reg, #data | Jump if byte ≠ #data |
| JC | Jump if CY = 1 |
| JNC | Jump if CY = 0 |
| JB | Jump if bit = 1 |
| JNB | Jump if bit = 0 |
| JBC | Jump if bit = 1 and clear bit |

CJNE- Compare and Jump if not equal

All conditional jumps are short jumps

The address of the target must within -128 to +127 bytes of the contents of PC

# Example

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

```
        MOV   A,#0        ;clear A(A=0)
        MOV   R5,A        ;clear R5
        ADD   A,#79H      ;A=0+79H=79H
        JNC   N_1         ;if no carry, add next number
        INC   R5          ;if CY=1, increment R5
N_1:    ADD   A,#0F5H     ;A=79+F5=6E and CY=1
        JNC   N_2         ;jump if CY=0
        INC   R5          ;If CY=1 then increment R5(R5=1)
N_2:    ADD   A,#0E2H     ;A=6E+E2=50 and CY=1
        JNC   OVER        ;jump if CY=0
        INC   R5          ;if CY=1, increment 5
OVER: MOV   R0,A          ;Now R0=50H, and R5=02
```

# Calculating address of short jumps

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

# Example

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

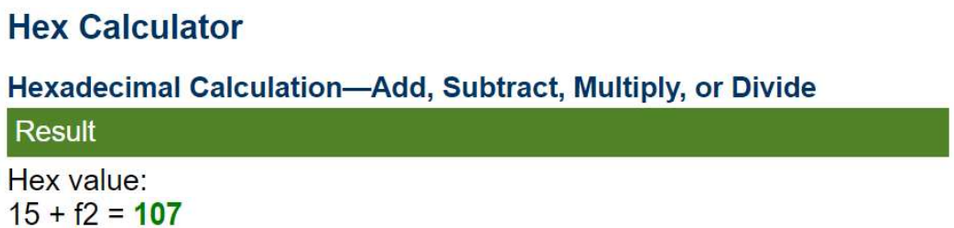
```
Line    PC              Opcode                  Mnemonic Operand
01      0000                                    ORG      0000
02      0000    7800                            MOV      R0,#0
03      0002    7455                    MOV     A,#55H
04      0004    6003                    JZ      NEXT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

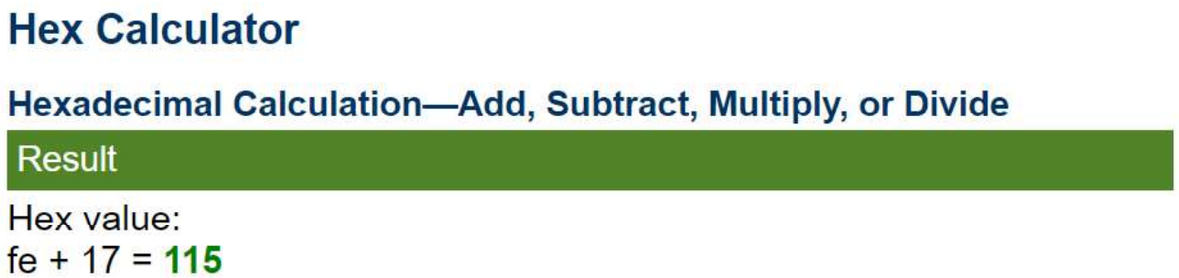
# Example

- First notice that the JZ and JNC instructions both jump forward.
  The target address for a forward jump is calculated by adding the PC of the following instruction to the second byte of the short jump instruction, which is called the relative address.
  In line 4 the instruction “JZ NEXT” has opcode of 60 and operand of 03 at the addresses of 0004 and 0005.
  The 03 is the relative address, relative to the address of the next instruction INC R0, which is 0006.
  By adding 0006 to 3, the target address of the label NEXT, which is 0009, is generated.
  In the same way for line 9, the “JNC OVER” instruction has opcode and operand of 50 and 05 where 50 is the opcode and 05 the relative address.
  Therefore, 05 is added to 000D, the address of instruction “CLR A”, giving 12H, the address of label OVER.

# Example

- Verify the calculation of backward jumps
  In that program list, “JNC AGAIN” has opcode 50 and relative address F2H. When the relative address of F2H is added to 15H, the address of the instruction below the jump, we have 15H + F2H = 07 (the carry is dropped). Notice that 07 is the address of label AGAIN.
  Look also at “SJMP HERE”, which has 80 and FE for the opcode and relative address, respectively. The PC of the following instruction, 0017H, is added to FEH, the relative address, to get 0015H, address of the HERE label (17H + FEH = 15H). Carry is dropped.



# Example-Backward jump address calculation

- 

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

# Example

- In this program, assume that the word “USA” is burned into ROM locations starting at 200H, and that the program is burned into ROM locations starting at 0. Analyze how the program works and state where “USA” is stored after this program is run.

```
ORG     0000H               ;burn into ROM starting at 0
        MOV     DPTR,#200H  ;DPTR=200H look-up table address
        CLR     A           ;clear A(A=0)
        MOVC    A,@A+DPTR   ;get the char from code space
        MOV     R0,A        ;save it in R0
        INC     DPTR        ;DPTR=201 pointing to next char
        CLR     A           ;clear A(A=0)
        MOVC    A,@A+DPTR   ;get the next char
        MOV     R1,A        ;save it in R1
        INC     DPTR        ;DPTR=202 pointing to next char
        CLR     A           ;clear A(A=0)
        MOVC    A,@A+DPTR   ;get the next char
        MOV     R2,A        ;save it in R2
HERE:SJMP       HERE        ;stay here

;Data is burned into code space starting at 200H
                ORG 200H
MYDATA: DB "USA"
                END         ;end of program
```

# Example

- In the above program ROM locations 200H - 202H have the following contents.
  200=('U')  201=('S')  202=('A")
  We start with DPTR = 200H, and A = 0.
  The instruction "MOVC A, @A+DPTR" moves the contents of ROM location 200H (200H + 0 = 200H) to register A.
  Register A contains 55H, the ASCII value for "U". This is moved to R0.
  Next, DPTR is incremented to make DPTR = 201H.
  A is set to 0 again to get the contents of the next ROM location 201H, which holds character "S".
  After this program is run, we have R0 = 55H, R1 = 53H, and R2 = 41H,
  the ASCII values for the characters "U", "S" and "A".

# Example

| Letter | ASCII Code | Binary |
|---|---|---|
| A | 065 | 01000001 |
| B | 066 | 01000010 |
| C | 067 | 01000011 |
| D | 068 | 01000100 |
| E | 069 | 01000101 |
| F | 070 | 01000110 |
| G | 071 | 01000111 |
| H | 072 | 01001000 |
| I | 073 | 01001001 |
| J | 074 | 01001010 |
| K | 075 | 01001011 |
| L | 076 | 01001100 |
| M | 077 | 01001101 |
| N | 078 | 01001110 |
| O | 079 | 01001111 |
| P | 080 | 01010000 |
| Q | 081 | 01010001 |
| R | 082 | 01010010 |
| S | 083 | 01010011 |
| T | 084 | 01010100 |
| U | 085 | 01010101 |
| V | 086 | 01010110 |
| W | 087 | 01010111 |
| X | 088 | 01011000 |
| Y | 089 | 01011001 |
| Z | 090 | 01011010 |

41H

55H

53H

# UNCONDITIONAL JUMP

- The unconditional jump is a jump in which control is transferred unconditionally to the target location
- LJMP (long jump)
  - 3-byte instruction
  - First byte is the opcode
  - Second and third bytes represent the 16-bit target address
  - Any memory location from 0000 to FFFFH
- SJMP (short jump)
  - 2-byte instruction
  - First byte is the opcode
  - Second byte is the relative target address
    - 00 to FFH (forward +127 and backward
    - -128 bytes from the current PC)

# CALLS

- Call instruction is used to call subroutine
- Subroutines are often used to perform tasks that need to be performed frequently
- This makes a program more structured in addition to saving memory space
- LCALL (long call)  SYNTAX- LCALL 16 bit address
  - 3-byte instruction
  - First byte is the opcode
  - Second and third bytes are used for address of target subroutine
- Subroutine is located anywhere within 64K byte address space
- ACALL (absolute call)  SYNTAX- ACALL 11bit address
  - 2-byte instruction

# CALLS

- The only difference between ACALL and LCALL is
- The target address for LCALL can be anywhere within the 64K byte address
- The target address of ACALL must be within a 2K-byte range

  **Only 11 bits of 2 bytes is used for address in ACALL**

  Syntax LCALL addr16

  ACALL addr11

- The use of ACALL instead of LCALL can save a number of bytes of program ROM space

# CALLS

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

# CALLS

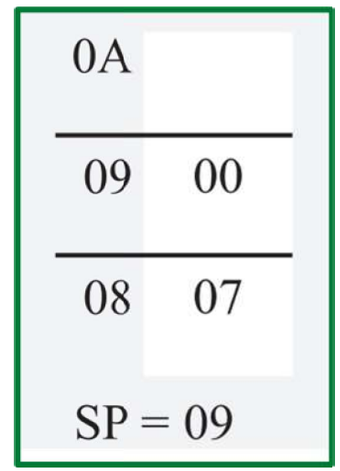
- Write a program to toggle all the bits of port 1 by sending to it the values 55H and AAH continuously. Put a time delay in between each issuing of data to port 1.

```
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
```

# CALLS

- Analyze the stack contents after the execution of the first LCALL in the following.

```
001    0000            ORG    0
002    0000   7455   BACK:   MOV    A,#55H          ;load A with 55H
003    0002   F590           MOV    P1,A            ;send 55H to port 1
004    0004   120300         LCALL  DELAY           ;time delay
005    0007   74AA           MOV    A,#0AAH         ;load A with AAH
006    0009   F590           MOV    P1,A            ;send AAH to port 1
007    000B   120300         LCALL  DELAY
008    000E   80F0           SJMP   BACK            ;keep doing this
009    0010
010    0010  ;————————this is the delay subroutine
011    0300                  ORG 300H
012    0300          DELAY:
013    0300   7DFF           MOV    R5,#0FFH        ;R5=255
014    0302   DDFE    AGAIN: DJNZ   R5,AGAIN        ;stay here
015    0304   22             RET                    ;return to caller
016    0305                  END                    ;end of asm file
```



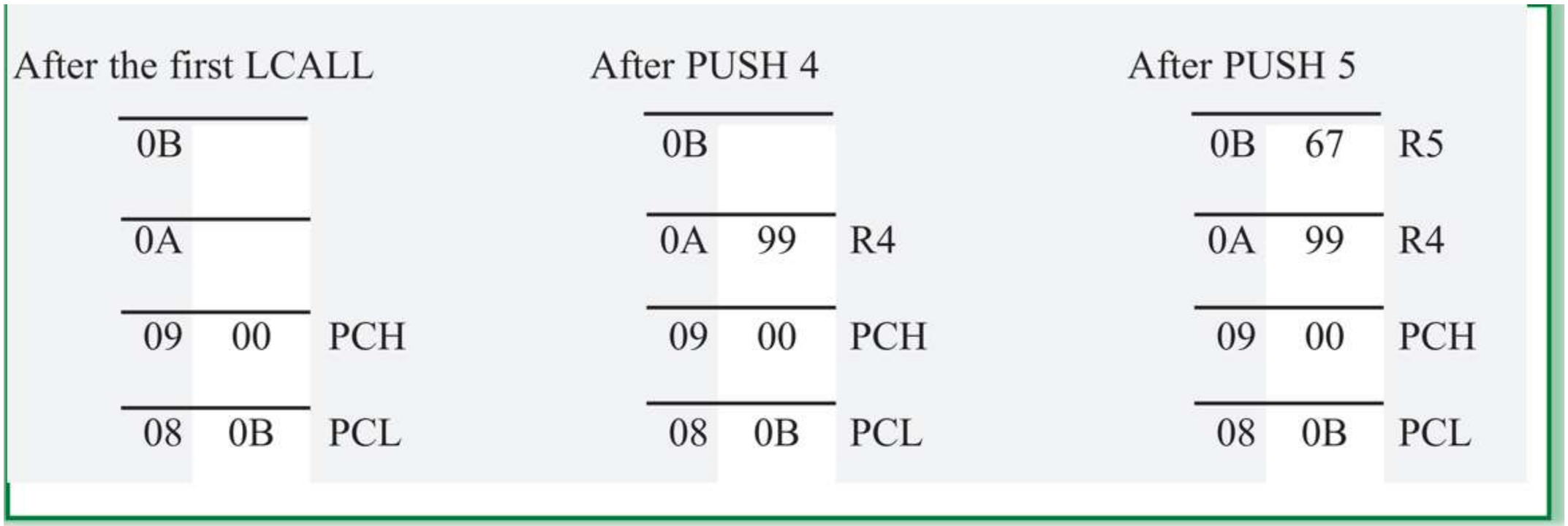
# CALL INSTRUCTION

- 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 port 1
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
```

# CALL INSTRUCTION

- for the PUSH and POP instructions we must specify the direct address of the register being pushed or popped. Here is the stack frame.



# CALL INSTRUCTION

- Rewrite this program as efficiently as you can
- 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.

```
        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 he
        MOV    P1,A            ;send AAH to port 1
        LCALL  DELAY
        SJMP   BACK            ;keep doing this indef
;-------        this is the delay subroutine
        ORG    300H            ;put time delay at add
DELAY:  MOV    R5,#0FFH        ;R5 = 255(FF in hex),t
AGAIN:  DJNZ   R5,AGAIN        ;stay here until R5 be
        RET                    ;return to caller (whe
        END                    ;end of asm file
```

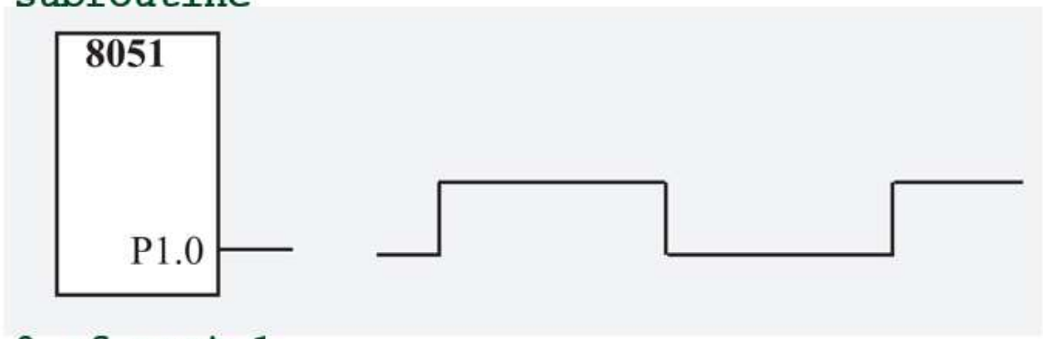
```
                ORG     0
                MOV     A,#55H      ;load A with 55H
BACK:           MOV     P1,A        ;issue value in reg A to port 1
                ACALL   DELAY       ;time delay
                CPL     A           ;complement reg A
                SJMP    BACK        ;keep doing this indefinitely

;----------this is the delay subroutine
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
                END                 ;end of asm file
```

# Use of call INSTRUCTION in I/O port programming for specific dutycycle.

- 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.

**(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.**

```
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
```




```
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
```

# Use of call INSTRUCTION in I/O port programming for specific dutycycle.

- 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.

**(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
```

# MACHINE CYCLE

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

# MACHINE CYCLE

- 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

**(a) 11.0592 MHz/12 = 921.6 kHz; machine cycle is 1/921.6 kHz = 1.085 µs (microsecond)**
**(b) 16 MHz/12 = 1.333 MHz; machine cycle (MC) = 1/1.333 MHz = 0.75 µs**
**(c) 20 MHz/12 = 1.66 MHz; MC = 1/1.66 MHz = 0.60 µs**

# MACHINE CYCLE

- For an 8051 system of 11.0592 MHz, find how long it takes to execute each of the following instructions.
- (a)MOV R3,#55 (b)DEC R3 (c)DJNZ R2,target
- (d)LJMP (e)SJMP (f)NOP (no operation) (g)MUL AB

The machine cycle for a system of 11.0592 MHz is 1.085 µs.

| Instruction | Machine cycles | Time to execute |
|---|---|---|
| (a) MOV R3,#55 | 1 | 1x1.085 µs = 1.085 µs |
| (b) DEC R3 | 1 | 1x1.085 µs = 1.085 µs |
| (c) DJNZ R2,target | 2 | 2x1.085 µs = 2.17 µs |
| (d) LJMP | 2 | 2x1.085 µs = 2.17 µs |
| (e) SJMP | 2 | 2x1.085 µs = 2.17 µs |
| (f) NOP | 1 | 1x1.085 µs = 1.085 µs |
| (g) MUL AB | 4 | 4x1.085 µs = 4.34 µs |

# MACHINE CYCLE- Delay calculation

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

```
                MOV     A,#55H  ;load A with 55H
AGAIN:  MOV     P1,A            ;issue value in reg A to port 1
                ACALL DELAY     ;time delay
                CPL     A       ;complement reg A
                SJMP    AGAIN   ;keep doing this indefinitely
;----Time delay
DELAY:  MOV     R3,#200         ;load R3 with 200
HERE:   DJNZ    R3,HERE         ;stay here until R3 become 0
                RET             ;return to caller
```

| | | | Machine Cycle |
|---|---|---|---|
| DELAY: | MOV | R3,#200 | 1 |
| HERE: | DJNZ | R3,HERE | 2 |
| | | RET | 2 |

Therefore, we have a time delay of [(200 x 2) + 1 + 2] x 1.085 µs = 436.255 µs.

# MACHINE CYCLE- Delay calculation

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

```
                MOV     A,#55H  ;load A with 55H
AGAIN:  MOV     P1,A            ;issue value in reg A to port 1
                ACALL DELAY     ;time delay
                CPL     A       ;complement reg A
                SJMP    AGAIN   ;keep doing this indefinitely
;----Time delay
DELAY:  MOV     R3,#200         ;load R3 with 200
HERE:   DJNZ    R3,HERE         ;stay here until R3 become 0
                RET             ;return to caller
```

| | | | Machine Cycle |
|---|---|---|---|
| DELAY: | MOV | R3,#200 | 1 |
| HERE: | DJNZ | R3,HERE | 2 |
| | | RET | 2 |

Therefore, we have a time delay of [(200 x 2) + 1 + 2] x 1.085 µs = 436.255 µs.

# MACHINE CYCLE- Delay calculation

- **For an 8051 system of 11.0592 MHz, find the time delay for the following subroutine:**

```
                                    Machine Cycle
DELAY: MOV   R3,#250                    1

HERE:  NOP                              1
       NOP                              1
       NOP                              1
       NOP                              1
       DJNZ  R3,HERE                    2

       RET                              2
```

# MACHINE CYCLE- Delay calculation

- **The time delay inside the HERE loop is [250(1 + 1 + 1 + 1 + 2)] x 1.085 µs = 1500 x 1.085 µs = 1627.5 µs.**

- **Adding the two instructions outside the loop we have 1627.5 µs + 3 x 1.085 µs = 1630.755 µs.**

# MACHINE CYCLE- Delay calculation-Homework

- **For a machine cycle of 1.085 ms, find the time delay in the following subroutine**

```
DELAY:
        MOV   R2,#200
AGAIN:  MOV   R3,#250
HERE:   NOP
        NOP
        DJNZ  R3,HERE
        DJNZ  R2,AGAIN
        RET
```

# CLOCK CYCLES OF VARIOUS VENDORS

| Chip/Maker | Clocks per Machine Cycle |
|---|---|
| AT89C51 Atmel | 12 |
| P89C54X2 Philips | 6 |
| DS5000 Dallas Semi | 4 |
| DS89C430/40/50 Dallas Semi | 1 |

**Find the period of the machine cycle (MC) in each case if XTAL = 11.0592 MHz, and discuss the impact on performance. (a) AT89C51 (b) P89C54X2 (c) DS5000 (d) DS89C4x0**

**(a) 11.0592 MHz/12 = 921.6 kHz; MC is 1/921.6 kHz = 1.085 µs (microsecond) = 1085 ns**
**(b) 11.0592 MHz/6 = 1.8432 MHz; MC is 1/1.8432 MHz = 0.5425 µs = 542 ns**
**(c) 11.0592 MHz/4 = 2.7648 MHz; MC is 1/2.7648 MHz = 0.36 µs = 360 ns**
**(d) 11.0592 MHz/1 = 11.0592 MHz; MC is 1/11.0592 MHz = 0.0904 µs = 90 ns**

# Home work

For an AT8051 and DS89C430/40/50 system of 11.0592 MHz, find how long it takes to execute each of the following instructions.

**MOV R3,#55** **(b) DEC R3** **(c) DJNZ R2,target** **(d) LJMP** **(e) SJMP**
**(f) NOP (no operation)** **(g) MUL AB**

TIME DELAY

| Instruction | 8051 | DS89C4x0 |
|---|---|---|
| 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 |

8051 DS89C4x0

# MACHINE CYCLE- Delay calculation

- **Find the time delay for the following subroutine if it is run on a DS89C430 chip, assuming a crystal frequency of 11.0592 MHz.**

```
                                                DS89C430 Machine Cycle
DELAY:  MOV     R3,#250
HERE:   NOP                                     1
        NOP                                     1
        NOP                                     1
        NOP                                     1
        DJNZ    R3,HERE                         4
        RET
```

The time delay inside the HERE loop is [250(1 + 1 + 1 + 1 + 4)] x 90 ns = 2000 x 90 ns = 180 µs.