Microcontrollers and Embedded Systems

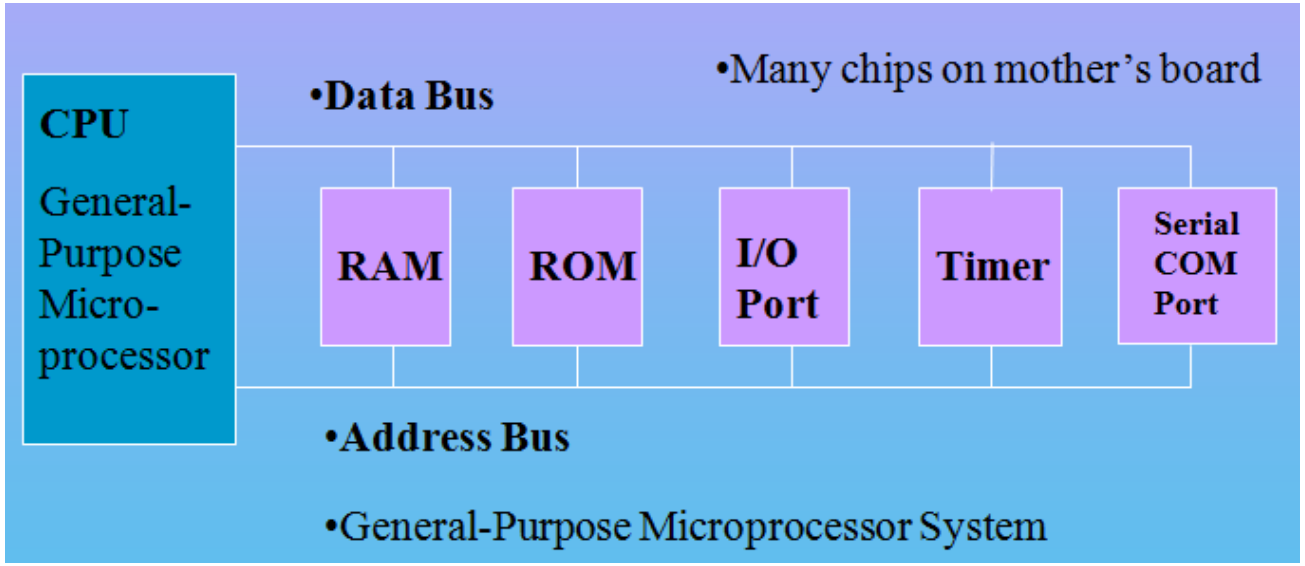
Unit I: Introduction to MCS-51

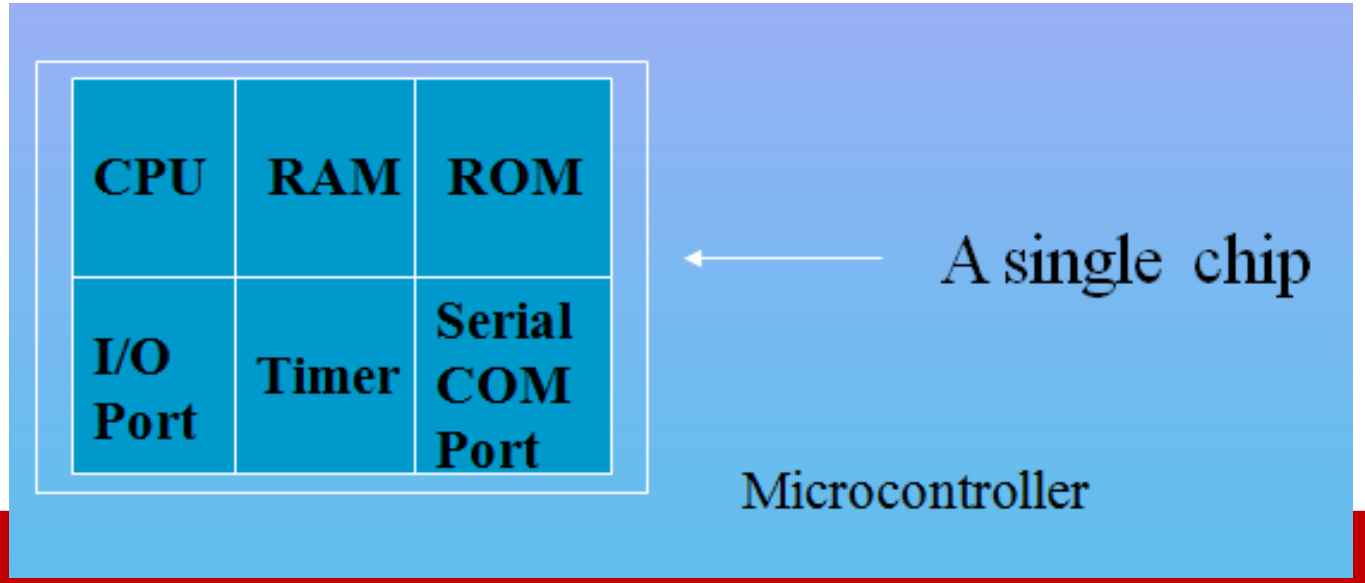
Symbiosis Institute of Technology, Nagpur

Symbiosis Institute of Technology, Nagpur Campus (SIT-N)

The 8051 Microcontroller: Microcontrollers and embedded processors, overview of the 8051 family, 8051 architecture-on chip resources, internal and external memory configuration, 8051 register banks, PSW, clock generator, other special function registers and their purpose, 8051 pin description

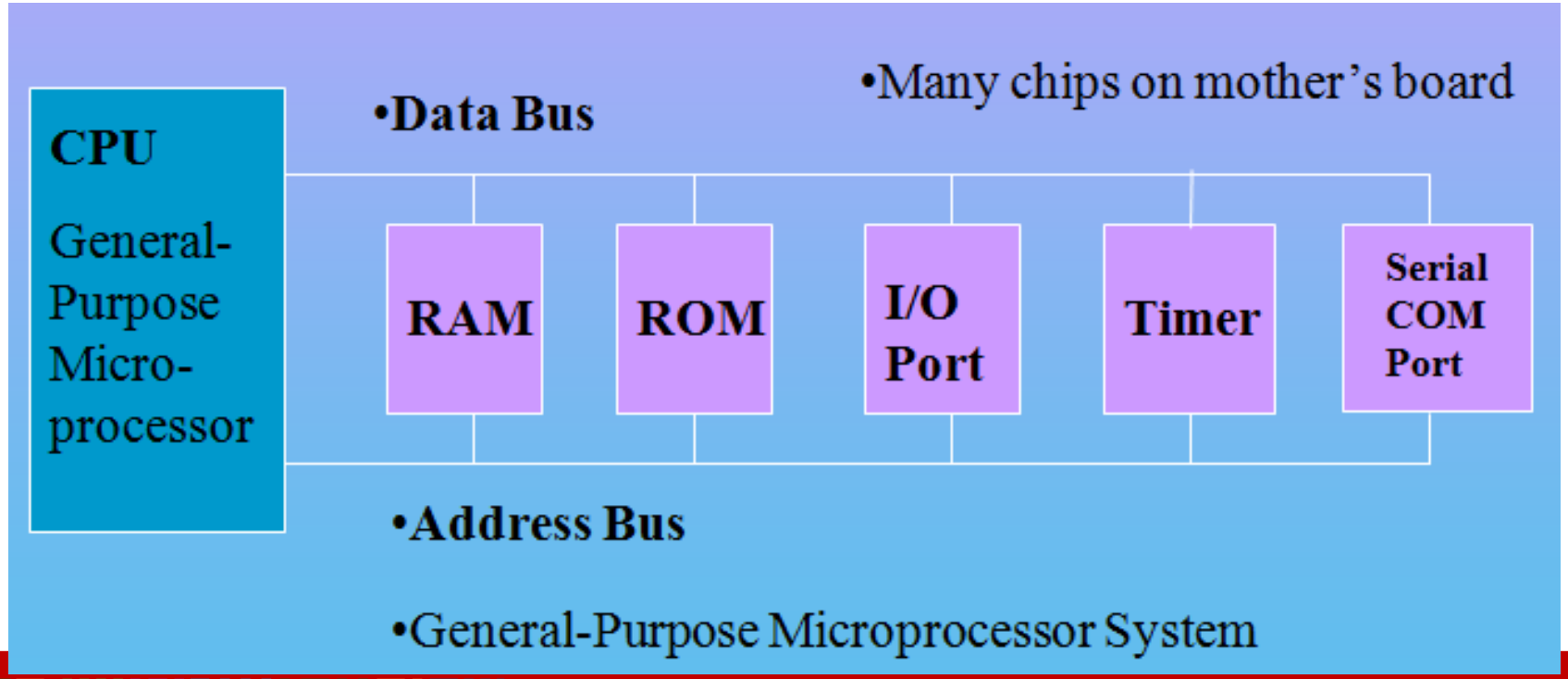
Microprocessor Vs Microcontroller





Microprocessor Based System

- CPU
- External RAM, ROM, I/O
- (No internal RAM, ROM, I/O ports in the CPU)

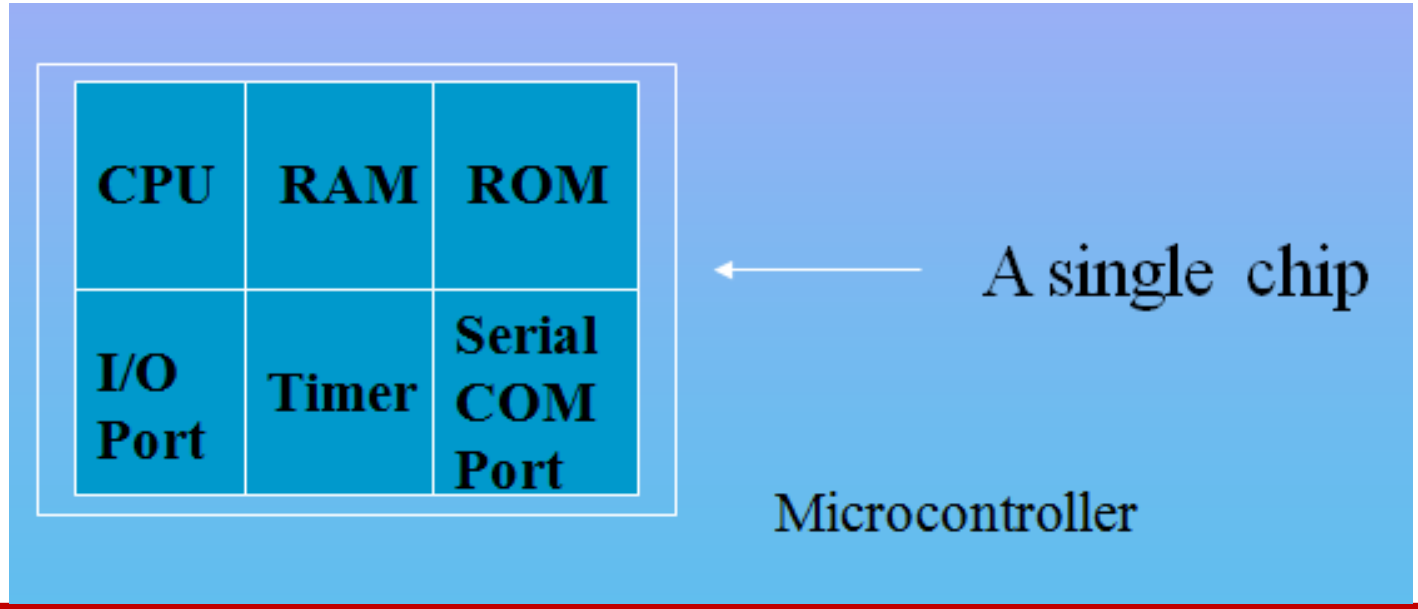


Microcontroller

A smaller computer on a CHIP

On-chip RAM, ROM, I/O Ports, Timer, Serial Controller...

Example: Motorola's 6811, Intel's 8051, Atmel 32



Difference between Microprocessor and Microcontroller

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Microprocessor	Microcontroller
CPU is stand-alone, RAM, ROM, I/O, timer are separate	CPU, RAM, ROM, I/O and timer are all on a single chip
Designer can decide on the amount of ROM, RAM and I/O ports.	Fixed amount of on-chip ROM, RAM, I/O ports
Expansive	Not Expansive
Versatility	Single-purpose
General-purpose	Special Purpose

Types of Microcontrollers

memory bits and instruction sets

Bit

Based on bit configuration, the microcontroller is further divided into three categories.

8-bit microcontroller – This type of microcontroller is used to execute arithmetic and logical operations like addition, subtraction, multiplication division, etc. For example, Intel 8031 and 8051 are 8 bits microcontroller.

16-bit microcontroller – This type of microcontroller is used to perform arithmetic and logical operations where higher accuracy and performance is required. For example, Intel 8096 is a 16-bit microcontroller.

32-bit microcontroller – This type of microcontroller is generally used in automatically controlled appliances like automatic operational machines, medical appliances, etc.

Example: PIC32MZ DA

Types of Microcontrollers

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Memory

Based on the memory configuration, the microcontroller is divided into two

- **External memory microcontroller** – This type of microcontroller is designed in such a way that they do not have a program memory on the chip. Hence, it is named as external memory microcontroller. For example: Intel 8031 microcontroller.
- **Embedded memory microcontroller** – This type of microcontroller is designed in such a way that the microcontroller has all programs and data memory, counters and timers, interrupts, I/O ports are embedded

Types of Microcontrollers

Instruction Set

Based on the instruction set configuration, the microcontroller is further divided into two categories.

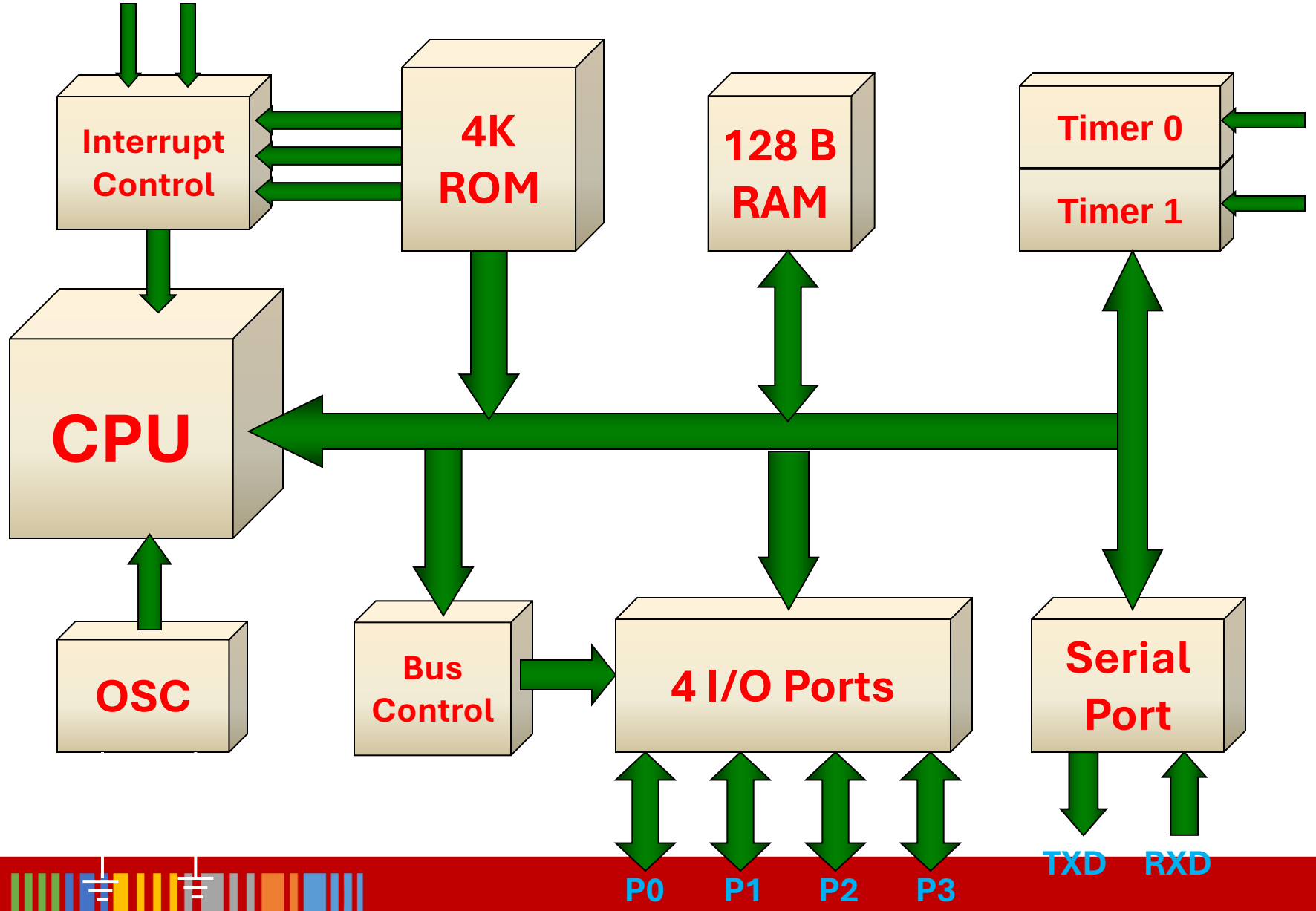
- **CISC** – CISC stands for complex instruction set computer. It allows the user to insert a single instruction as an alternative to many simple instructions.
- **RISC** – RISC stands for Reduced Instruction Set Computers. It reduces the operational time by shortening the clock cycle per instruction.

Intel family of 8-bit microcontrollers

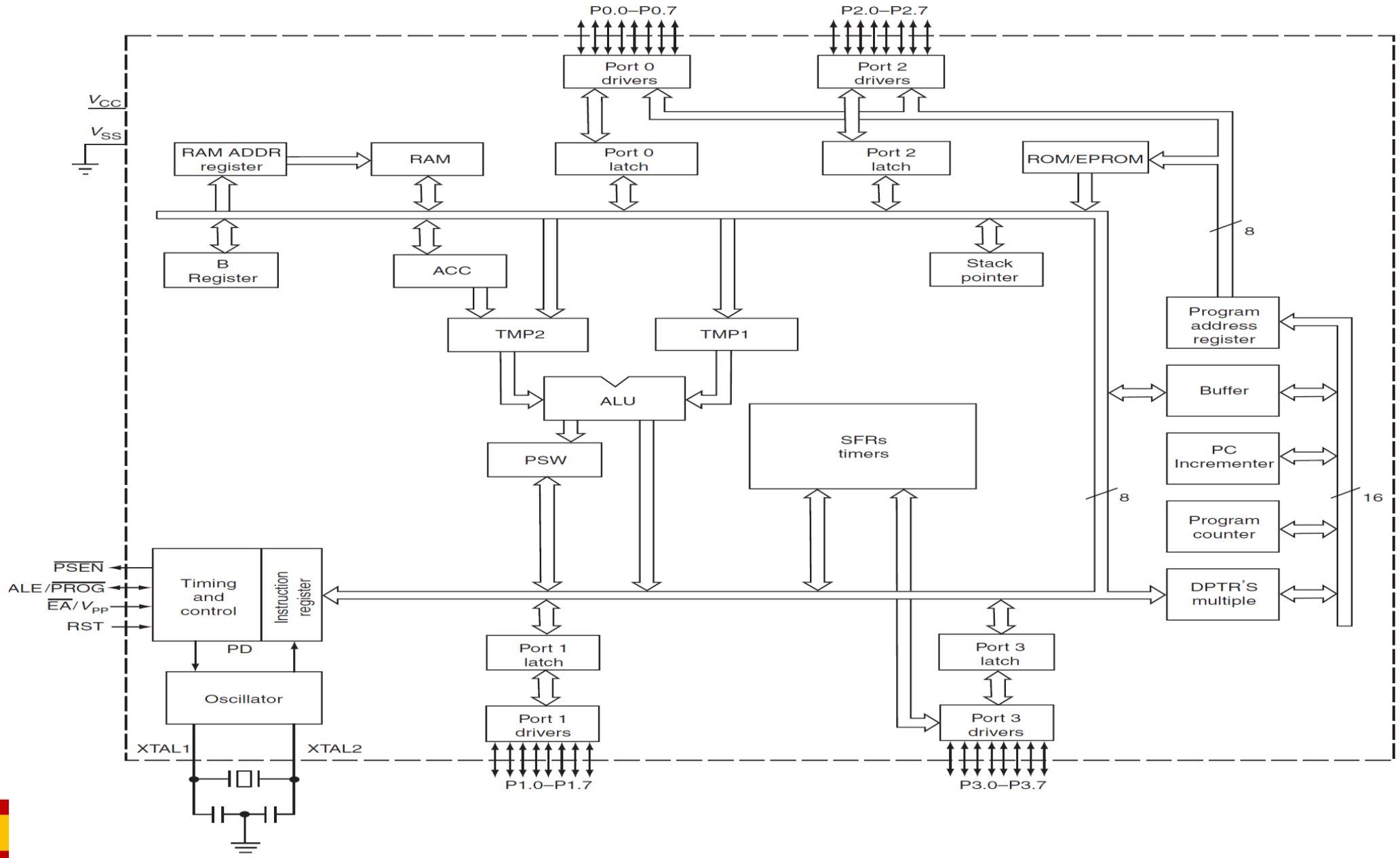
- The 8051 is a subset of the 8052
- The 8031 is a ROM-less 8051
  - Add external ROM to it

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

General Block Diagram of 8051



8051 Architecture



- Oscillators and clock circuits
- 8-bit accumulator and B
- 16-bit program counter and DPTR
- 8-bit program status word
- 8-bit stack pointer
- Internal ROM 4Kbyte
- Internal RAM→128 bytes
- 4 register banks → each contains 8 registers
- 16-bit counters/timers
- Control registers→ TCON,TMOD,SCON,PCON,IP,IE

8051 Architecture

Oscillator and Clock

- Generate the clock pulses by which all internal operations are synchronized.
- **Pulse** – clock frequency establishes the smallest interval of time within the microcontroller.
- **Machine Cycle** – smallest interval of time to accomplish any simple (or) part of complex instruction.
- **State** – basic time interval for discrete operations of microcontroller like fetching, decoding, executing, write back.
- 1 Machine Cycle = 6 States.

8051 Architecture

Oscillator and Clock

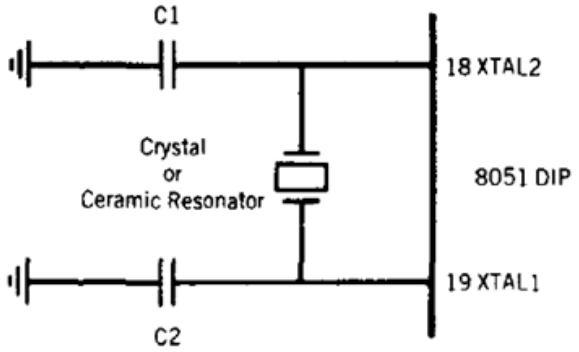
The time period for the machine cycle

$$T = \frac{1}{Timer\ frequency}$$

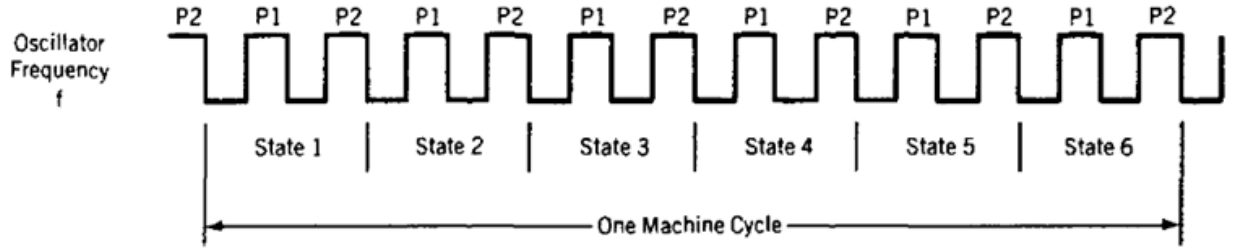
The timer frequency is

$$\frac{\text{Crystal Oscillator frequency}}{12}$$

Every timer need a clock to work and 8051 provides it from an external crystal which is the main clock source of the timer. The internal circuitry of the 8051 microcontroller provides a clock source to the timers that are 1/12th of the frequency of crystal frequency



Crystal or Ceramic Resonator Oscillator Circuit



To calculate the execution time of an instruction

$$T_{inst} = \frac{C * 12}{\text{Clock Frequency}}$$

C- no of cycles

A and B CPU Registers

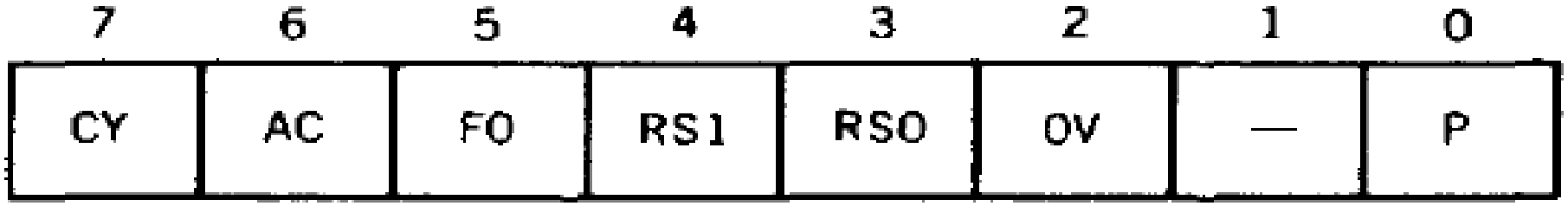
- Totally 34 general purpose registers or working registers.
- Two of these A and B hold results of many instructions, particularly math and logical operations of 8051 CPU.
- The other 32 are in four banks,B0 – B3 of eight registers each.
- A(accumulator) is used for addition, subtraction, multiplication, division, boolean bit manipulation and for data transfers.
- But B register can only be used multiplication and division for operations.

Program counter (PC) and data pointer (DPTR)

- They are both 16-bit registers.
- Each is to hold the address of a byte in memory
- PC contains the address of the next instruction to be executed.(0000h to FFFFh)
- DPTR is made up of two 8-bit register DPH and DPL
- DPTR contains the address of internal & external code and data that has to be accessed.

Flag and the Program Status Word

- Flags→1 bit registers provided to store the results of certain program instruction
- Flag bits are grouped inside the PSW and PCON(power control)
- 8051 has
  - ✓4 math flags→ CY, AC, OV, P
  - ✓3 general purpose flags→ F0,RS0,RS1



Flag and the Program Status Word

Bit	Symbol	Function
7	CY	Carry flag; used in arithmetic, JUMP, ROTATE, and BOOLEAN instructions
6	AC	Auxilliary carry flag; used for BCD arithmetic
5	F0	User flag 0
4	RS1	Register bank select bit 1
3	RS0	Register bank select bit 0
2	OV	Overflow flag; used in arithmetic instructions
1	—	Reserved for future use
0	P	Parity flag; shows parity of register A: 1 = Odd Parity

RS1	RS0	
0	0	Select register bank 0
0	1	Select register bank 1
1	0	Select register bank 2
1	1	Select register bank 3

Parity Bit (P)

•This parity flag bit is used to show the number of 1s in the accumulator only. If the accumulator register contains an odd number of 1s, then this flag set to 1.

•If accumulator contains even number of 1s, then this flag cleared to 0.

Auxiliary Carry Flag (AC)

•It is used in association with BCD arithmetic. This flag is set when there is a carry out of the D3 bit of the accumulator.

Carry Flag (CY)

This flag is used to indicate the carry generated after arithmetic operations. It can also be used as an accumulator, to store one of the data bits for bit-related Boolean instructions.

Flag and the Program Status Word

Example

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

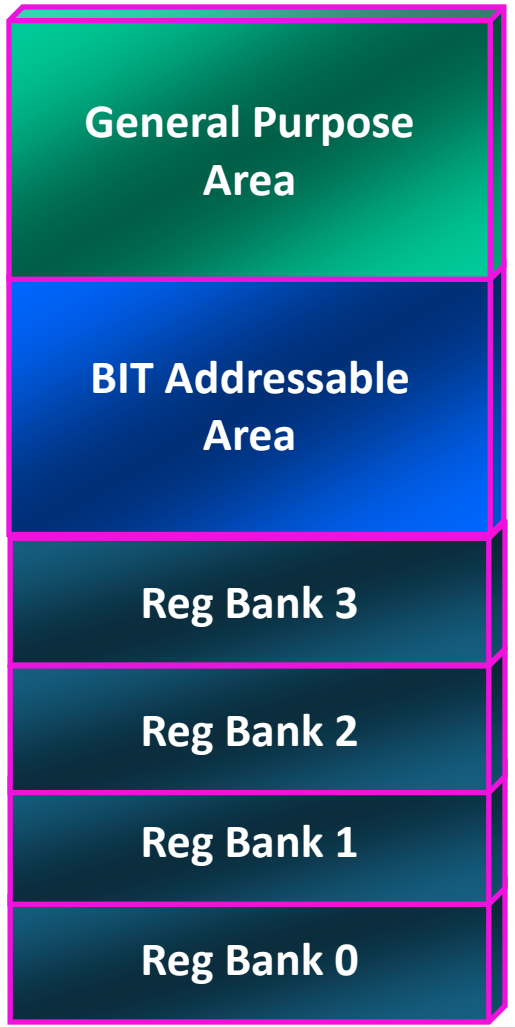
MOV A, #9CH
ADD A, # 64H

Solution:	9C	10011100
	+64	01100100
	100	00000000

Internal memory

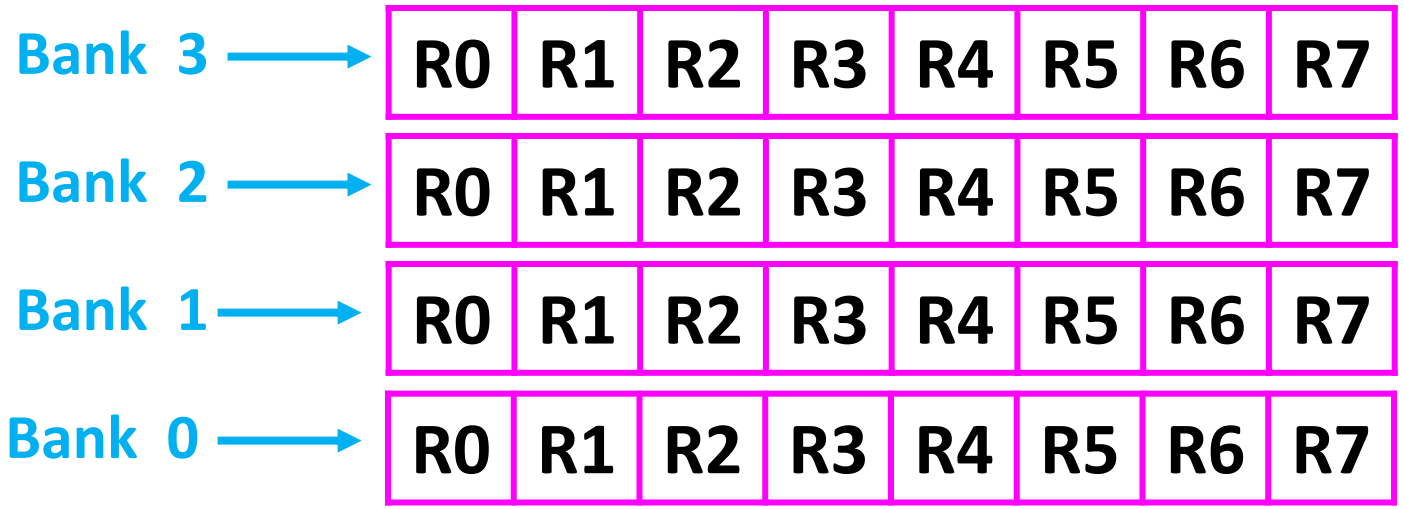
128 Byte RAM

- There are 128 bytes of RAM in the 8051.
  - Assigned addresses 00 to 7FH
- The 128 bytes are divided into 3 different groups as follows:
  1. A total of **32 bytes** from locations 00 to 1F hex are set aside for ***register banks and 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.***

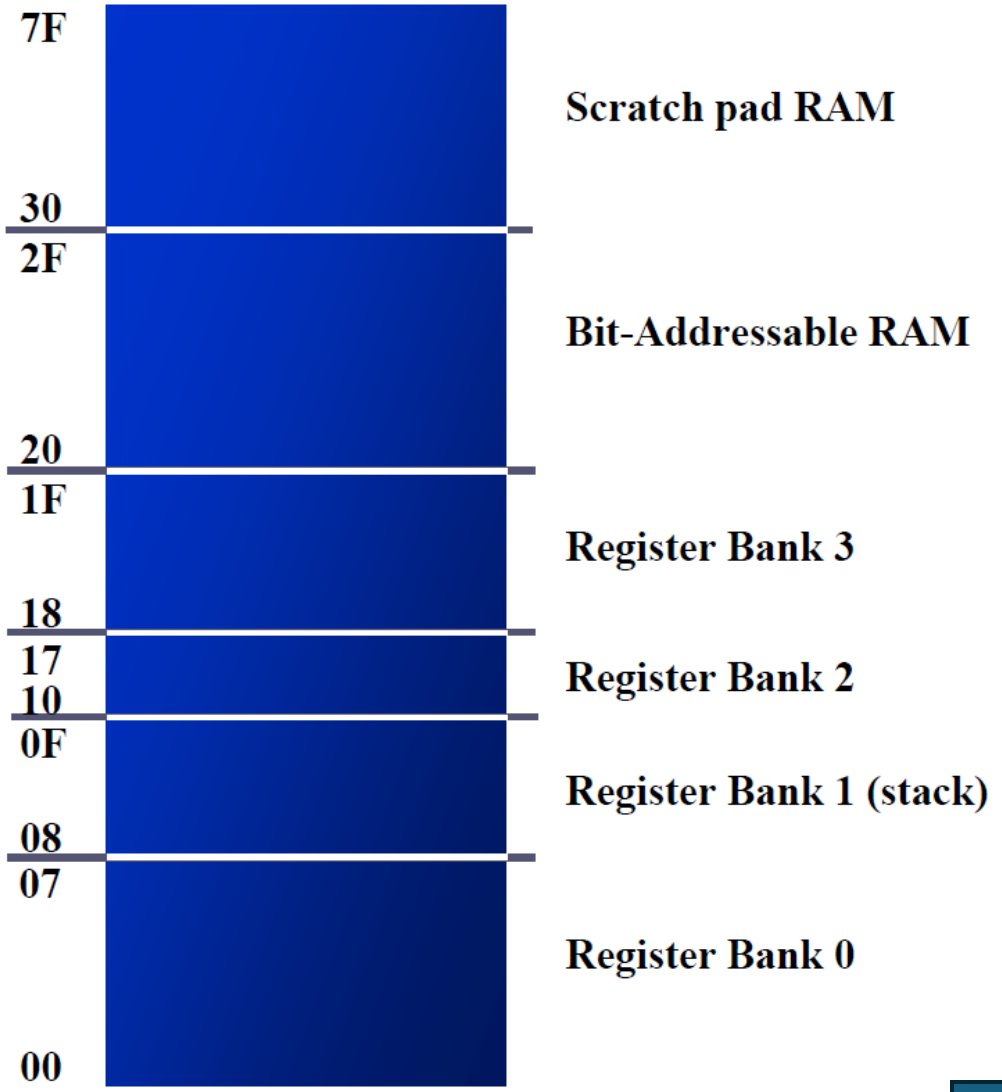


Internal memory

128 Byte RAM



8051 Register Bank Structure

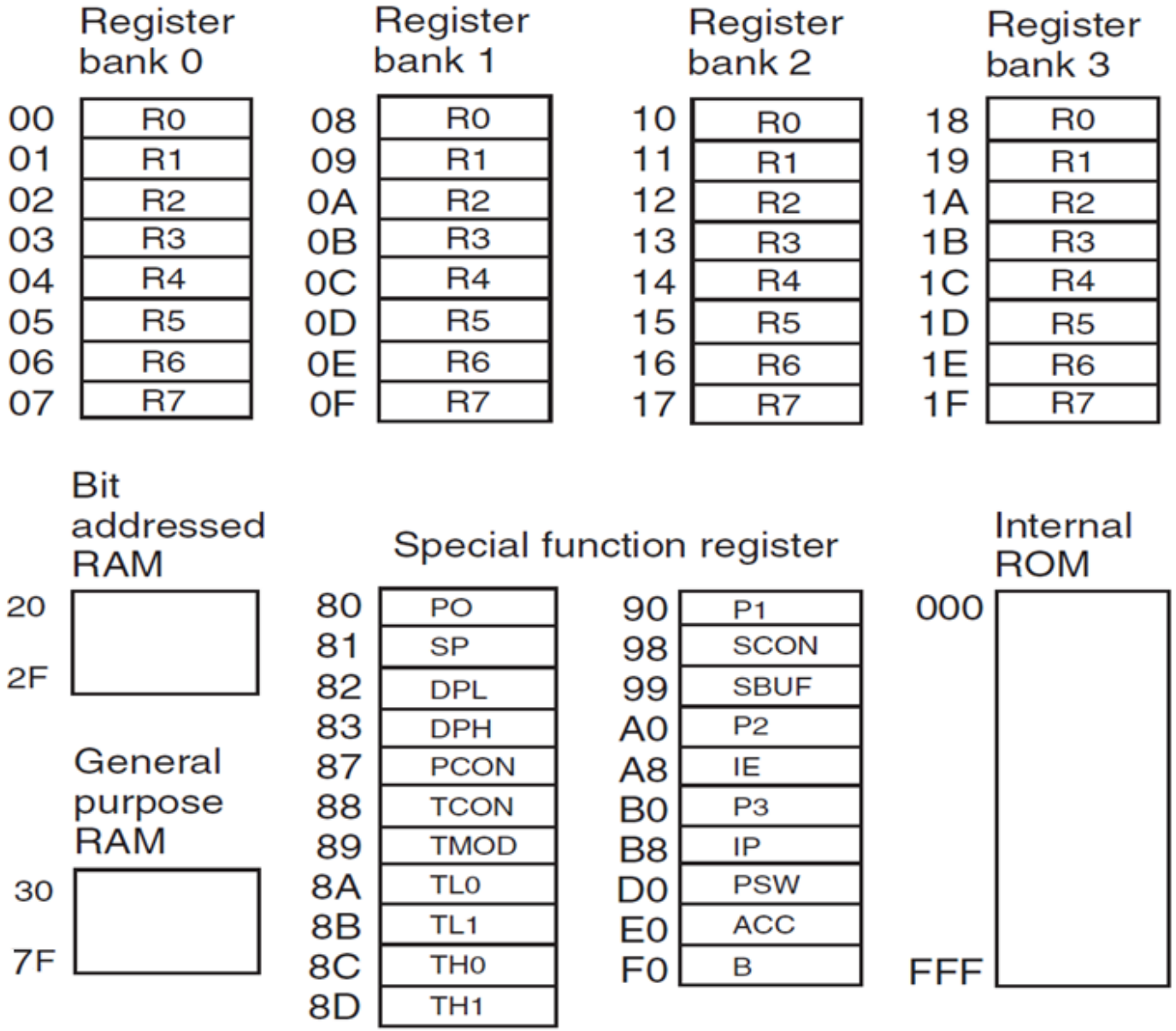


8051 RAM with addresses

Internal memory

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

128 Byte RAM



Internal memory

4K ROM

- Block of program code →stores in ROM
- Code address space is 0000h to 0FFFh.
- Program addresses higher than 0FFFh, automatically 8051 fetches codes from external memory(0000h to FFFFh)

Stack and stack pointer

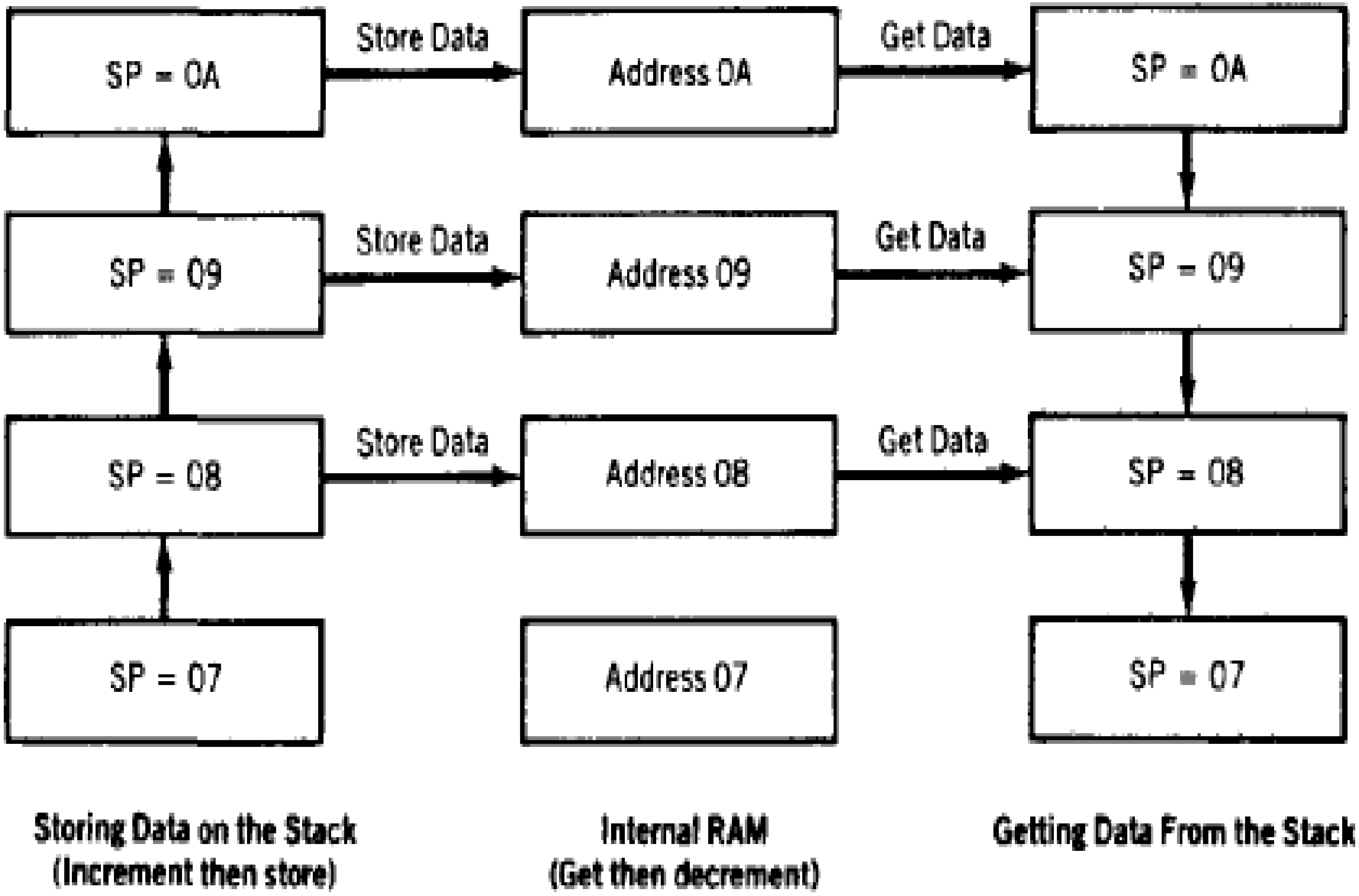
- The **stack** is a section of RAM used by the CPU to store information **temporarily.**
  - This information could be data or an address
- The register used to access the stack is called the **SP (stack pointer) register**
  - The stack pointer in the 8051 is **only 8 bit wide**, which means that it can take value of 00 to FFH
  - SP is pointing to the last used location of the stack
  - When the 8051 is powered up, the SP register contains value 07
  - RAM location 08 is the first location being used for the stack by the 8051

Stack and stack pointer

- The storing of a CPU register in the stack is called a **PUSH**
  - As we push data onto the stack, the **SP is incremented** by one
  - This is **different** from many microprocessors
- Loading the contents of the stack back into a CPU register is called a **POP**
  - With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is **decremented** once

Placing data on to stack → SP incremented
Retrieving data from stack → SP decremented

Stack and stack pointer



Special function register

NAME	FUNSCONCTION	INTERNAL RAM ADDRESS
A	ACCUMUATOR	0E0
B	Athematic	OFO
DPH	Addressing external memory	83
DPL	Addressing external memory	82
IE	Interrupt enable control	Oa8
IP	Interrupt priority	0b8
PO	i/o port latch	80
P1	i/o port latch	90
P2	i/o port latch	A0
P3	i/o port latch	0bo
PCON	Power control	87
PSW	Program status word	ODO
SCON	Serial port control	98
SBUF	Serial port data buffer	99

Special function register

NAME	FUNSCONCTION	INTERNAL RAM ADDRESS
SP	Stack pointer	81
TMOD	Timer/counter mode control	89
TCON	Timer/counter control	88
TLO	Timer 0 low byte	8A
THO	Timer 0 high byte	8C
TL1	Timer 1 low byte	8B
TH1	Timer 1 high byte	8D

I/O Port of 8051

- 8051 microcontrollers have 4 I/O ports each of 8-bit, which can be configured as input or output.
    - Port 0, Port 1, Port 2 and Port 3.
- Hence, total 32 input/output pins allow the microcontroller to be connected with the peripheral devices.

Features of the four ports of 8051?

- Each port has 8 pins. Thus, the four ports jointly comprise 32 pins.
- All ports are bidirectional.

I/O Port of 8051

Features of the four ports of 8051

- They are constructed with a D type output latch. They have output drivers and input buffers.
- We can modify their functions using software and hardware that they connect to.
- All the ports are configured as input ports on Reset.
- To configure ports as an input port 1 must be written to that port
- To configure it as an output port 0 must be written to it.

I/O Port of 8051

Port 0（ pins 32-39 ）

- Address is 80H
- Port 0 of the 8051 has two main functions: To be used as a simple input-output port and to access external memory in conjunction with Port 2.
- Port 0 is also designated as **AD0-AD7**.
- When connecting an 8051 to an external memory, port 0 provides both address and data

I/O Port of 8051

Port 0（pins 32-39）

- The 8051 multiplexes address and data through port 0 to save pins.
- ALE indicates if P0 has address or data.
  - When ALE=0, it provides data D0-D7
  - When ALE=1, it has address A0-A7
- When the 8051 wants to access external memory, the address for the memory generates due to Port 0 and Port 2
- We get the lower half of the address from Port 0 and the upper half from Port 2.

I/O Port of 8051

Port 1 (pins 1-8)

- Address is 90H
- It has only one function – to act as an Input-Output port
- When Port 1 is functioning as an input port, a digital '1' (FFH) is written to 90H.
- When Port 1 is functioning as an output port, the latch is given a 'LOW' signal (00H).

I/O Port of 8051

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

Port 2（ pins 20-27 ）

- Address is 10H
- Its main functions are similar to Port 0. It can be used as an input-output port. And can access external memory in conjunction with Port 0.

16-bit address

- P0 provides address A0-A7
- P2 provides address A8-A15

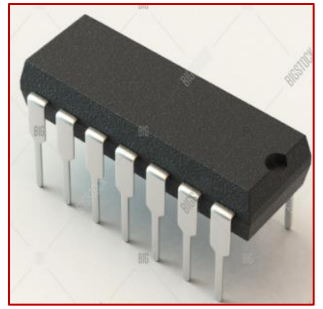
I/O Port of 8051

Port 3（ pins 10-17 ）

- Address is B0H
- Port 3 performs two main functions
  - **I/O port**
  - **Alternate SFR function**

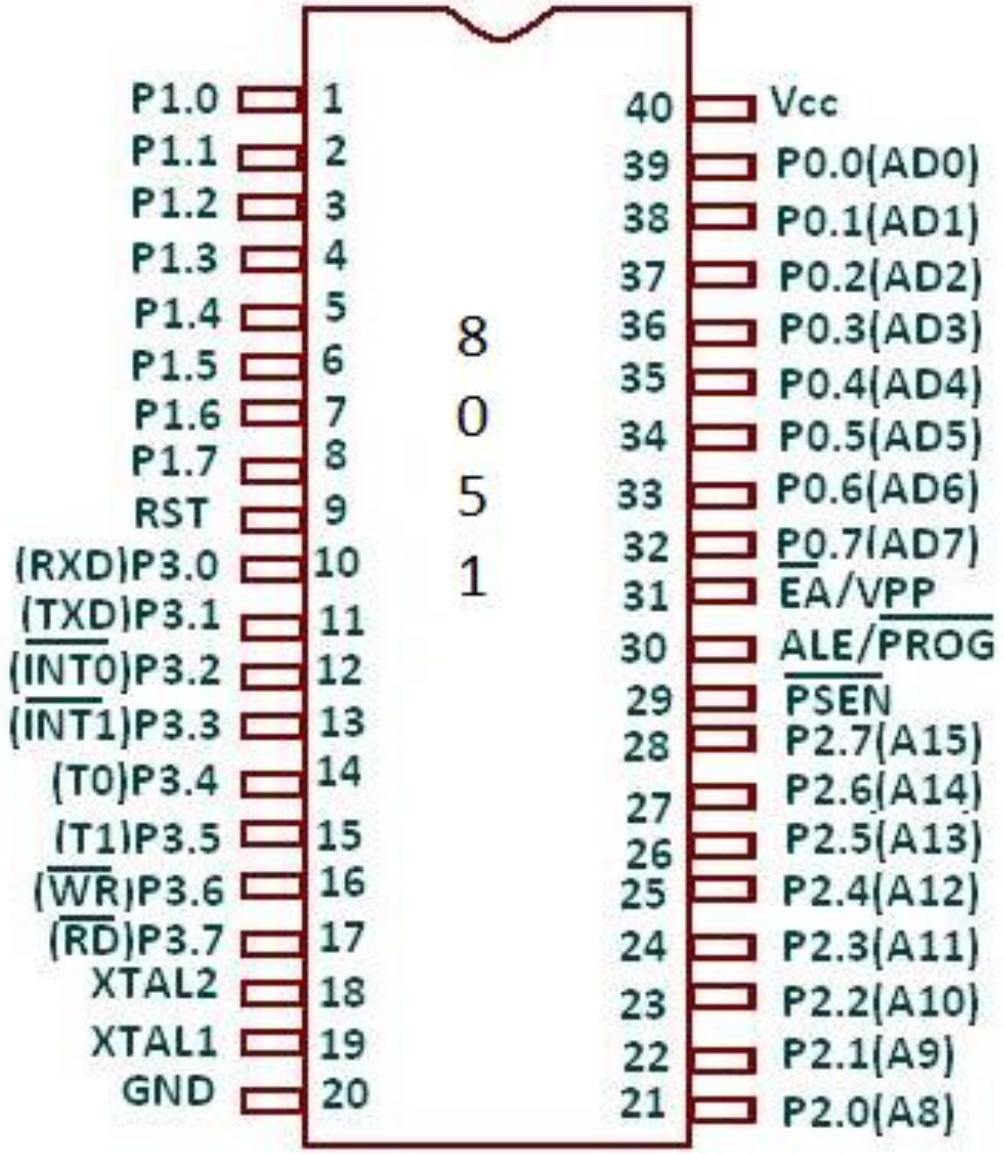
Pin Description of the 8051

- 8051 family members (e.g., 8751, 89C51, 89C52, DS89C4x0)
  - Have **40 pins** dedicated for various functions such as I/O, RD, WR, address, data, and interrupts.
  - Come in different packages, such as
    - *DIP(dual in-line package),*
    - *QFP(quad flat package), and*
    - *LLC(leadless chip carrier)*



- Some companies provide a 20-pin version of the 8051 with a reduced number of I/O ports for less demanding applications

Pin Description of the 8051



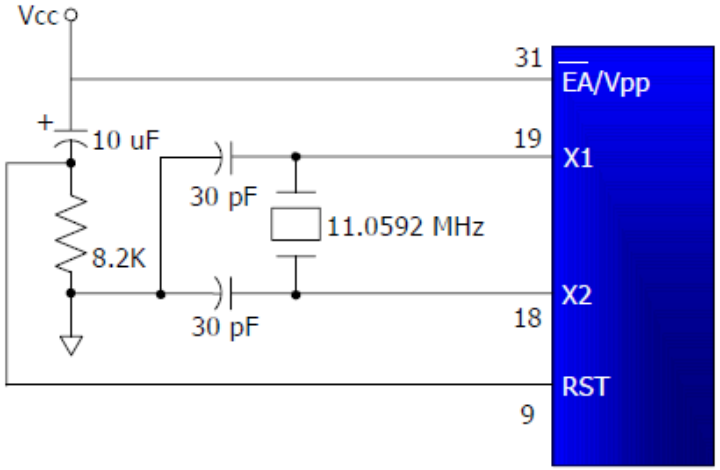
Pin Description of the 8051

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

RST

- RESET pin is an input and is active high (normally low).
- Upon applying a high pulse to this pin, the microcontroller will reset and terminate all activities.
- This is often referred to as a **power-on reset(POR).**
- Activating a power-on reset will cause all values in the registers to be lost.

Power-on RESET circuit



RESET value of some 8051 registers

we must place the first line of source code in ROM location 0

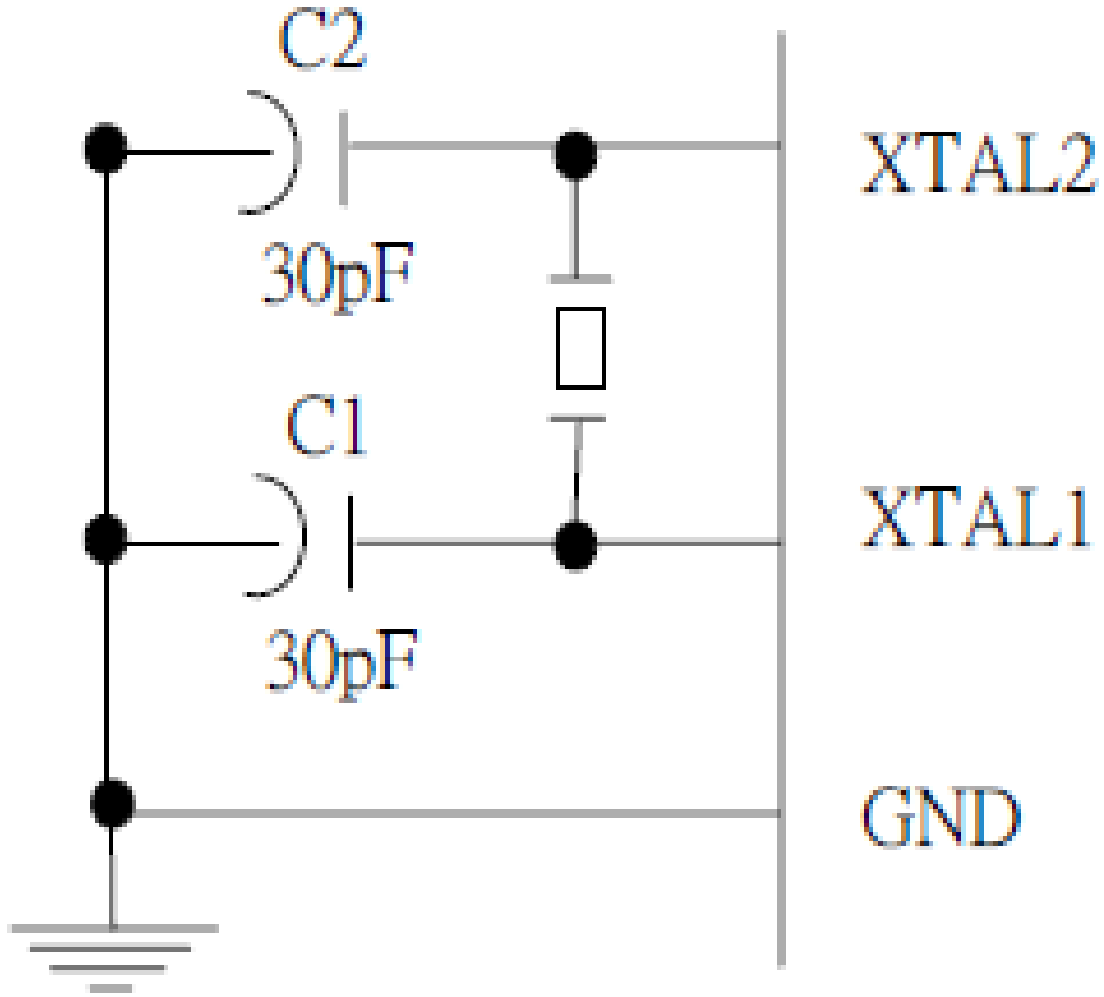
Register	Reset Value
PC	0000
DPTR	0000
ACC	00
PSW	00
SP	07
B	00
P0-P3	FF

Pin Description of the 8051

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

XTAL1 and XTAL2

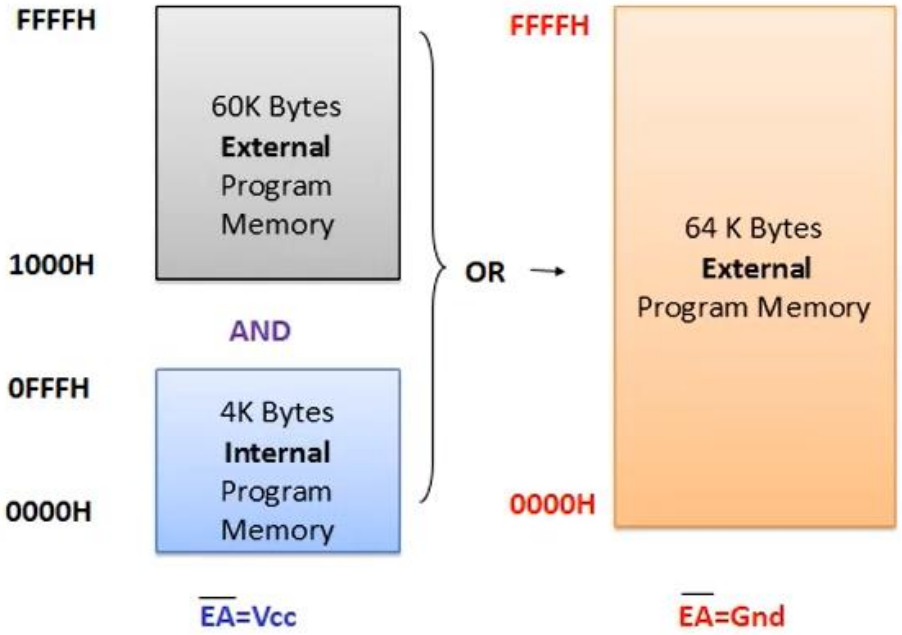
- The 8051 has an on-chip oscillator but requires an external crystal to run it
  - A quartz crystal oscillator is connected to inputs XTAL1 (pin19) and XTAL2 (pin18)
  - The quartz crystal oscillator also needs two capacitors of 30 pF value
  - The original 8051 operates at 12 MHZ



Pin Description of the 8051

$\overline{EA}$

- $\overline{EA}$**"external access"**, is an input pin and must be connected to Vcc or GND
- The 8051 family members all come with on-chip ROM to store programs and also have an external code and data memory.
- Normally EA pin is connected to Vcc (Internal Access)
- EA pin must be connected to GND to indicate that the code or data is stored externally.



Pin Description of the 8051

$\overline{\text{PSEN}}$ and ALE

- $\overline{\text{PSEN}}$, "program store enable", is an output pin
- This pin is connected to the OE pin of the external memory.
- For External Code Memory, $\overline{\text{PSEN}}$ = 0
- For External Data Memory, $\overline{\text{PSEN}}$ = 1
- ALE pin is used for demultiplexing the address and data.
  - When ALE=0, it provides data D0-D7
  - When ALE=1, it has address A0-A7

Pin Description of the 8051

PIN	TYPE	NAME AND FUNCTION
Vss	**I**	**Ground:** 0 V reference.
Vcc	**I**	**Power Supply:** This is the power supply voltage for normal, idle, and power-down operation.
P0.0 - P0.7	**I/O**	**Port 0:** Port 0 is an open-drain, bi-directional I/O port. Port 0 is also the multiplexed low-order address and data bus during accesses to external program and data memory.
P1.0 - P1.7	**I/O**	**Port 1:** Port I is an 8-bit bi-directional I/O port.
P2.0 - P2.7	**I/O**	**Port 2:** Port 2 is an 8-bit bidirectional I/O. Port 2 emits the high order address byte during fetches from external program memory and during accesses to external data memory that use 16 bit addresses.
P3.0 - P3.7	**I/O**	**Port 3:** Port 3 is an 8 bit bidirectional I/O port. Port 3 also serves special features as explained.

Pin Description of the 8051

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

PIN	TYPE	NAME AND FUNCTION
RST	**I**	**Reset:** A high on this pin for two machine cycles while the oscillator is running, resets the device.
ALE	**O**	**Address Latch Enable:** Output pulse for latching the low byte of the address during an access to external memory.
$\overline{\text{PSEN}}$	**O**	**Program Store Enable:** The read strobe to external program memory. When executing code from the external program memory, PSEN* is activated twice each machine cycle, except that two PSEN* activations are skipped during each access to external data memory.
$\overline{\text{EA}}$ **/VPP**	**I**	**External Access Enable/Programming Supply Voltage:** EA* must be externally held low to enable the device to fetch code from external program memory locations. If EA* Is held high, the device executes from internal program memory. This pin also receives the programming supply voltage Vpp during Flash programming. (applies for 89c5x MCU's)

Features of 8051

- 8-bit Processor
- 4KB Internal ROM
- 128 Bytes Internal RAM
- Four 8 BIT I/O PORTS (32 I/O LINES)
- Two 16 Bit Timers/Counters
- On Chip Full Duplex UART for Serial Communication
- 5 Vector Interrupts (2 External, 3 Internal - Timer0,Timer1,Serial)
- On Chip Clock Oscillator
- 16-bit Address bus
- 16-bit program counter to access external Code Memory and
- 16-bit Data Pointer to access external Data Memory
- 128 user defined flags
- 32 General Purpose Registers each of 8 bits

8051 Addressing Modes

- The CPU can access data in various ways, which are called addressing modes

1. Immediate
2. Register
3. Direct
4. Register indirect
5. External Direct

8051 Addressing Modes

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Immediate Addressing Mode

- The source operand is a **constant.**
- The immediate data must be preceded by the sign, **"#"**
- Can load information into **any registers**, including 16-bit DPTR register
  - DPTR can also be accessed as two 8-bit registers, the high byte DPH and low byte DPL

```
MOV A,#25H       ;load 25H into A
MOV R4,#62       ;load 62 into R4
MOV B,#40H       ;load 40H into B
MOV DPTR,#4521H  ;DPTR=4512H
MOV DPL,#21H     ;This is the same
MOV DPH,#45H     ;as above

;illegal!! Value > 65535 (FFFFH)
MOV DPTR,#68975
```

8051 Addressing Modes

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

Register Addressing Mode

- Use registers to hold the data to be ma

```
MOV A,R0     ;copy contents of R0 into A
MOV R2,A     ;copy contents of A into R2
ADD A,R5     ;add contents of R5 to A
ADD A,R7     ;add contents of R7 to A
MOV R6,A     ;save accumulator in R6
```

- The source and destination registers must match in size.

  MOV DPTR,A will give an error

```
MOV DPTR,#25F5H
MOV R7,DPL
MOV R6,DPH
```

- The movement of data between Rn registers is not allowed

  MOV R4,R7 is invalid

8051 Addressing Modes

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Direct Addressing Mode

- It is most often used the direct addressing mode to access RAM locations 30 – 7FH.
- The entire 128 bytes of RAM can be accessed.
- Contrast this with immediate addressing mode, there is no "#" sign in the operand.

```
MOV R0,40H  ;save content of 40H in R0
MOV 56H,A   ;save content of A in 56H
```

8051 Addressing Modes

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Direct Addressing Mode

SFR Registers & their Addresses

```
MOV   0E0H,#55H    ;is the same as
MOV   A,#55H       ;which means load 55H into A (A=55H)

MOV   0F0H,#25H    ;is the same as
MOV   B,#25H       ;which means load 25H into B (B=25H)

MOV   0E0H,R2      ;is the same as
MOV   A,R2         ;which means copy R2 into A

MOV   0F0H,R0      ;is the same as
MOV   B,R0         ;which means copy R0 into B
```

8051 Addressing Modes

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Example 5-1

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

Solution:

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

(b) From Table 5-1, P1 address = 80H; P2 address = A0H

```
        MOV  A,#55H        ;A=55H
        MOV  80H,A         ;P1=55H
        MOV  0A0H,A        ;P2=55H
```

8051 Addressing Modes

Stack and Direct Addressing Mode

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

Solution:

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

8051 Addressing Modes

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

Register Indirect Addressing Mode

- 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
```

8051 Addressing Modes

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

Register Indirect Addressing Mode

- Write a program to copy the value 55H into RAM memory locations 40H to 41H using **(a) direct addressing mode**, **(b) register indirect addressing mode without a loop**, and **(c) with a loop.**

Solution:

(a)

```
    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
```

(b)

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

(c)

```
        MOV A,#55H     ;A=55H
        MOV R0,#40H    ;load pointer.R0=40H,
        MOV R2,#02     ;load counter, R2=3
AGAIN:  MOV @R0,A      ;copy 55 to RAM R0 points to
        INC R0         ;increment R0 pointer
        DJNZ R2,AGAIN  ;loop until counter = zero
```

8051 Addressing Modes

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

Register Indirect Addressing Mode

- **Looping is not possible in direct addressing mode.**

- Write a program to clear 16 RAM locations starting at RAM address 60H.

```
        CLR A           ;A=0
        MOV R1,#60H     ;load pointer. R1=60H
        MOV R7,#16      ;load counter, R7=16
AGAIN:  MOV @R1,A       ;clear RAM R1 points to
        INC R1          ;increment R1 pointer
        DJNZ R7,AGAIN   ;loop until counter=zero
```

8051 Addressing Modes

External Direct or Indexed Addressing

- External Memory is accessed.
- There are only two commands that use External Direct addressing mode:
  - **MOVX A, @DPTR**
    MOVX @DPTR, A
- DPTR must first be loaded with the address of external memory.

8051 Instruction Sets

- An instruction is an order or command given to a processor by a computer program.
- All commands are known as instruction set and set of instructions is known as program.
  - Types Of Instructions
    - Instructions are divided into 3 types
      - One/single byte instruction.
      - Two/double byte instruction.
      - Three/triple byte instruction

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

One/single byte instruction.

- In 1-byte instruction, the opcode and the operand of an instruction are represented in one byte.
- Example MOV A, R_n

Two/Double byte instruction.

- Two-byte instruction is the type of instruction in which the first 8 bits indicates the opcode, and the next 8 bits indicates the operand.
- Example MOV A, #25H

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

Three/Triple byte instruction.

- Three-byte instruction is the type of instruction in which the first 8 bits indicates the opcode, and the next two bytes specify the 16-bit address.
- The low-order address is represented in second byte and the high-order address is represented in the third byte.
- Example MOV DPTR, #23E3H

8051 Instruction Sets

8051 has simple instruction set in different groups. There are:

- Arithmetic instructions
- Logical instructions & Rotate instructions
- Data transfer instructions
- Branching and looping instructions
- Bit control instructions

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

DATA TRANSFER	*ARITHMETIC*	*LOGICAL & Rotation Instruction*	*Bit Control Instruction*	*Branching and looping instructions*
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 Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Data transfer instructions
  - These instructions move the content of one register to another one (Source to Destination).
  - Data can be transferred to stack with the help of PUSH and POP instructions.

DATA TRANSFER
MOV
MOVC
MOVX
PUSH
POP
XCH
XCHD

8051 Instruction Sets

- Data transfer instructions
  - Instruction: MOV
  - Syntax: MOV destination, source
    - MOV copies the value of source into destination. The value of source is not affected.
    - Both destination and source must be in Internal RAM.
    - No flags are affected unless the instruction is moving the value of a bit into the carry bit

8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Data transfer instructions
  - MOV
    - 1-bit data transfer involving the CY flag
      - MOV C, bit
      - MOV bit, C
  - MOV
    - 16-bit data transfer involving the DPTR
      - MOV DPTR, #data

Other Examples

- MOV A, Rn
- MOV A, @Rn
- MOV A,#X
- MOV Rn, A
- MOV Rn, #X

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Data transfer instructions
  - Instruction: MOVC
  - Function: Move Code Byte to Accumulator
  - Syntax: MOVC A,@A+register
    - MOVC moves a byte from Code Memory into the Accumulator.
    - The Code Memory address from which the byte will be moved is calculated by summing the value of the Accumulator with either DPTR or the Program Counter (PC).
    - In the case of the Program Counter, PC is first incremented by 1 before being summed with the Accumulator.
    - Must use indexed addressing
      - MOVC A, @A+DPTR
      - MOVC A, @A+PC

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Data transfer instructions
  - Instruction: MOVX
  - Function: Move Data To/From External Memory (XRAM)
  - Syntax: MOVX operand1,operand2
    - MOVX moves a byte to or from External Memory into or from the Accumulator.
    - If operand1 is @DPTR, the Accumulator is moved to the 16-bit External Memory address indicated by DPTR. This instruction uses both P0 (port 0) and P2 (port 2) to output the 16-bit address and data.
    - If operand2 is DPTR then the byte is moved from
    - If operand1 is @R0 or @R1, the Accumulator is moved to the 8-bit External Memory address indicated by the specified Register. This instruction uses only P0 (port 0) to output the 8-bit address and data. P2 (port 2) is not affected.
    - If operand2 is @R0 or @R1 then the byte is moved from External Memory into the Accumulator

8051 Instruction Sets

- Data transfer instructions
  - MOVX
    - Data transfer between the accumulator and a byte from external data memory.
      - MOVX A, @Ri
      - MOVX A, @DPTR
      - MOVX @Ri, A
      - MOVX @DPTR, A

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Data transfer instructions
  - Instruction: XCH
  - Function: Exchange Bytes
  - Syntax: XCH A, register
    - Exchanges the value of the Accumulator with the value contained in register.
    - XCH A, Rn
    - XCH A, direct

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Data transfer instructions
  - Instruction: **SWAP**
  - Function: Swap Accumulator Nibbles.
  - Syntax: SWAP A
    - SWAP swaps bits 0-3 of the Accumulator with bits 4-7 of the Accumulator. This instruction is identical to executing "RR A" or "RL A" four times.

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Data transfer instructions
  - Instruction: **PUSH**
  - Function: Push Value Onto Stack
  - Syntax: PUSH
    - PUSH "pushes" the value of the specified internal RAM (iram) address onto the stack.
    - PUSH first increments the value of the Stack Pointer by 1, then takes the value stored in iram address and stores it in Internal RAM at the location pointed to by the incremented Stack Pointer.
      - PUSH DPL
      - PUSH DPH
      - PUSH Rn
      - PUSH @Ri (R0 or R1)

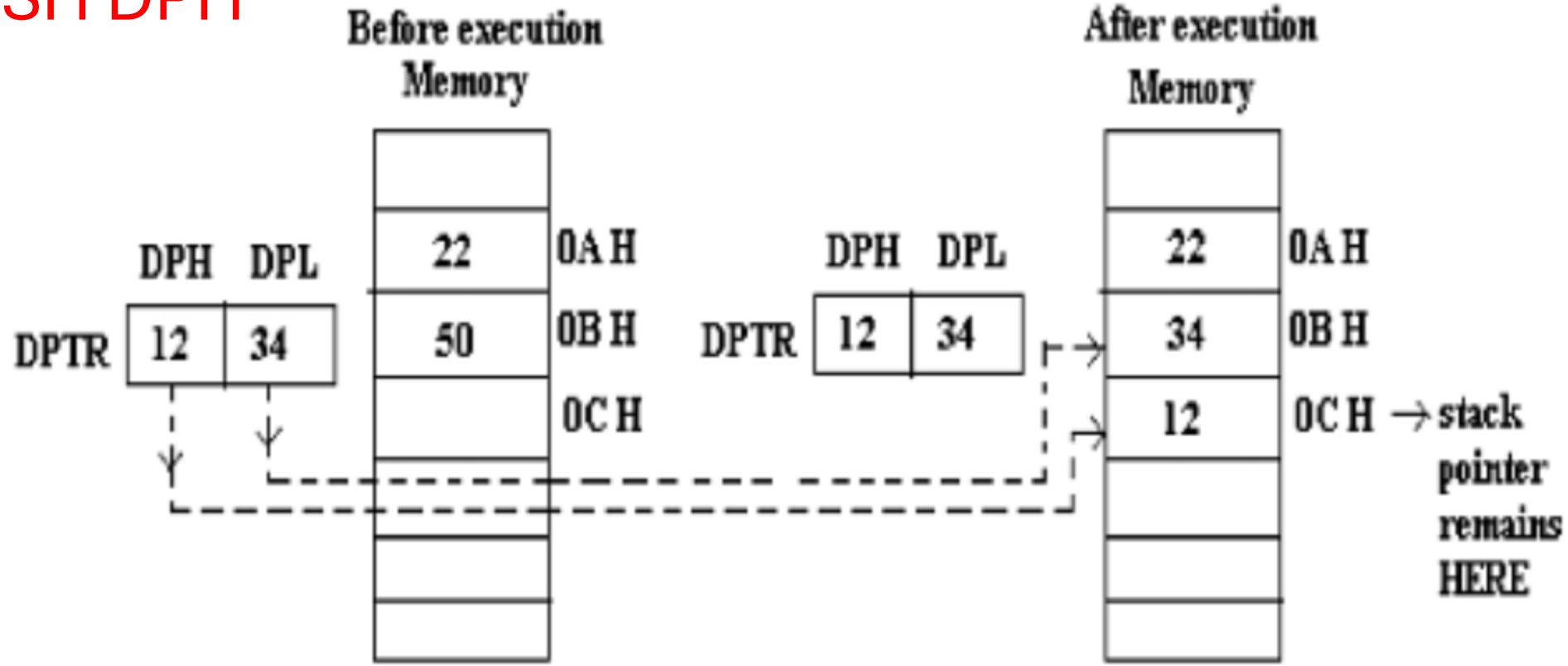
8051 Instruction Sets

- Data transfer instructions

Let SP =0AH and data pointer = 1234H

PUSH DPL

PUSH DPH



8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

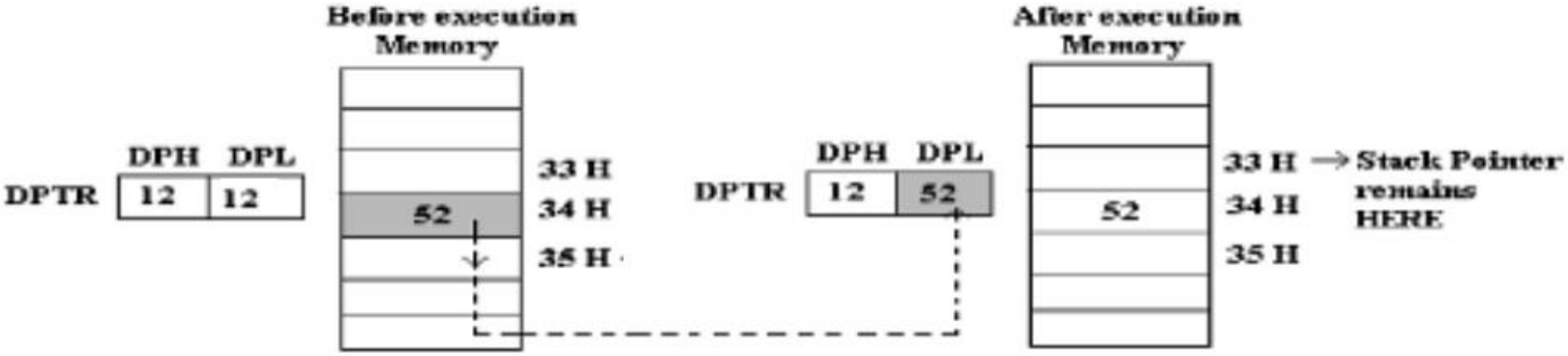
- Data transfer instructions
  - Instruction: **POP**
  - Function: Pop Value From Stack
  - Syntax: POP
    - POP "pops" the last value placed on the stack into the Internal RAM (iram) address specified.
    - POP will load iram address with the value of the Internal RAM address pointed to by the current Stack Pointer. The stack pointer is then decremented by 1.
      - POP 40H
      - POP DPH
      - POP Rn
      - POP @Ri (R0 or R1)

8051 Instruction Sets

- Data transfer instructions

Let SP = 34H and data at internal RAM locations 34H be 52H, then instruction POP DPL will move 52 H to DPL and then SP decremented by 1

POP DPL



8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

DATA TRANSFER	*ARITHMETIC*	*LOGICAL & Rotation Instruction*	*Bit Control Instruction*	*Branching and looping instructions*
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
SWAP	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
			ORL	ACALL
			CPL	RET
				RETI
				JMP

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - These instructions perform several basic operations. After execution, the result is stored in the first operand.
  - 8-bit addition, subtraction, multiplication, increment-decrement instructions can be p

ARITHMETIC
ADD
ADDC
SUBB
INC
DEC
MUL
DIV
DA A

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - Instruction: ADD, ADDC
  - Function: Add Accumulator, Add Accumulator With Carry
  - Syntax: ADD A, Operand ; ADDC A, Operand
    - ADD and ADDC both add the value operand to the value of the Accumulator, leaving the resulting value in the Accumulator.
    - The value operand is not affected.
    - ADD and ADDC function identically except that ADDC adds the value of operand as well as the value of the Carry flag whereas ADD does not add the Carry flag to the result.

8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - ADD
    - ADD A, R1 (Add the content of register1 to Accumulator) A+ R1
  - ADDC
    - ADDC A,#2 (Add 2 to accumulator with carry) A + 2 + C

Other Examples

- ADD A, # X — A = A + Byte
- ADDC A, Rn — A = A + Rn + C
- ADDC A , Rx (Direct) — A = A + Rx + C

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - Instruction: SUBB
  - Function: Subtract from Accumulator With Borrow
  - Syntax: SUBB A, Operand
    - The SUBB instruction subtracts the operand and the carry flag from the accumulator. The result is stored in the accumulator.
    - The value operand is not affected.
    - The Carry Bit (C) is set if a borrow was required for bit 7, otherwise it is cleared.
    - That is, if the unsigned value being subtracted is greater than the Accumulator the Carry Flag is set.

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - Instruction: SUBB
  - Function: Subtract from Accumulator With Borrow
  - Syntax: SUBB A, Operand
    - The Auxiliary Carry (AC) bit is set if a borrow was required for bit 3, otherwise it is cleared.
    - That is, the bit is set if the low nibble of the value being subtracted was greater than the low nibble of the Accumulator.
    - The Overflow (OV) bit is set if a borrow was required for bit 6 or for bit 7, but not both.
    - The subtraction of two signed bytes resulted in a value outside the range of a signed byte (-128 to 127). Otherwise, it is cleared.

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - SUBB A, @R1
    - A = A – (R1)-1

Other Examples

- SUBB A, Rn A = A – Rn – 1
- SUBB A, Rx (Direct) A = A – Rx – 1
- SUBB A, @ Ri A = A – Ri – 1
- SUBB A, # X A = A – Byte – 1

8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - Instruction: MUL
  - Function: Multiply Accumulator by B
  - Syntax: MUL AB
    - Multiples the unsigned value of the Accumulator by the unsigned value of the "B" register.
    - The least significant byte of the result is placed in the Accumulator and the most-significant-byte is placed in the "B" register.
    - The Carry Flag (C) is always cleared.

MUL AB B:A = A * B

8051 Instruction Sets

- Arithmetic Instructions
  - Instruction: DIV
  - Function: Divide Accumulator by B
  - Syntax: DIV AB
    - Divides the unsigned value of the Accumulator by the unsigned value of the "B" register.
    - The resulting quotient is placed in the Accumulator and the remainder is placed in the "B" register.
    - The Carry Flag (C) is always cleared.

DIV AB A = A/B

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - Instruction: INC
  - Function: Increment Register
  - Syntax: INC register
    - INC increments the value of register by 1. If the initial value of register is 255 (0xFF Hex), incrementing the value will cause it to reset to 0. (Note: The Carry Flag is NOT set when the value "rolls over" from 255 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 65535 (0xFFFF Hex), incrementing the value will cause it to reset to 0. (The Carry Flag is NOT set when the value of DPTR "rolls over" from 65535 to 0.)

8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Arithmetic Instructions
  - INC A
    - A = A +1

Other Examples

- INC A A = A + 1
- INC Rn Rn = Rn + 1
- INC Rx Rx = Rx + 1
- INC @ Ri Ri = Ri + 1

8051 Instruction Sets

Arithmetic Instructions

- Instruction: DEC
- Function: Decrement Register
- Syntax: DEC register
  - DEC 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). (The Carry Flag is NOT set when the value "rolls over" from 0 to 255).
    - DEC A A = A – 1
    - DEC Rn Rn = Rn – 1
    - DEC Rx Rx = Rx – 1
    - DEC @ Ri (Ri) =(Ri) – 1

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

DATA TRANSFER	*ARITHMETIC*	*LOGICAL & Rotation Instruction*	*Bit Control Instruction*	*Branching and looping instructions*
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
SWAP	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
			ORL	ACALL
			CPL	RET
				RETI
				JMP

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Logical and Rotate Instructions
  - These instructions perform logical operations between two register contents on bit-by-bit basis.
  - After execution, the result is stored in the first operand.

LOGICAL & Rotation Instruction
ANL
ORL
XRL
CLR
CPL
RL
RLC
RR
RRC

8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Logical Instructions
  - Instruction: ANL
  - Function: Bitwise AND
  - Syntax: ANL operand1, operand2
    - ANL does a bitwise "AND" operation between operand1 and operand2, leaving the resulting value in operand1.
    - The value of operand2 is not affected.
      - » ANL A, Rn (A) = (A) ^ (Rn)
      - » ANL A, Rx (A) = (A) ^ (Rx)
      - » ANL A,@ Ri (A) = (A) ^ (Ri)
      - » ANL A, # X (A) = (8 bit data) ^ (A)
      - » ANL Rx, A (Rx) = (A) ^ (Rx)
      - » ANL Rx,# X (Rx) = (8 bit data) ^ (Rx)

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Logical Instructions
  - Instruction: ORL
  - Function: Bitwise OR
  - Syntax: ORL operand1, operand2
    - ORL does a bitwise "OR" operation between operand1 and operand2, leaving the resulting value in operand1.
    - The value of operand2 is not affected.
      - » ORL A, Rn (A) = (A) ! (Rn)
      - » ORL A, Rx (A) = (A) ! (Rx)
      - » ORL A,@ Ri (A) = (A) ! (Ri)
      - » ORL A, # X (A) = (8 bit data) ! (A)
      - » ORL Rx, A (Rx) = (A) ! (Rx)
      - » ORL Rx,# X (Rx) = (8 bit data) ! (Rx)

8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Logical Instructions
  - Instruction: XRL
  - Function: Bitwise Exclusive OR
  - Syntax: XRL operand1, operand2
    - XRL does a bitwise "EXCLUSIVE OR" operation between operand1 and operand2, leaving the resulting value in operand1
    - The value of operand2 is not affected.
      - » XRL A, Rn (A) = (A)⊕ (Rn)
      - » XRL A, Rx (A) = (A) ⊕ (Rx)
      - » XRL A,@ Ri (A) = (A) ⊕ (Ri)
      - » XRL A, # X (A) = (8 bit data) ⊕ (A)
      - » XRL Rx, A (Rx) = (A) ⊕ (Rx)
      - » XRL Rx,# X (Rx) = (8 bit data) ⊕ (Rx)

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Logical Instructions
  - Instruction: CLR
  - Function: Clear Register
  - Syntax: CLR register
  - CLR clears (sets to 0) all the bit(s) of the indicated register. If the register is a bit (including the carry bit), only the specified bit is affected. Clearing the Accumulator sets the Accumulator's value to 0. The value of operand2 is not affected.

CLR A (A) ←0

CLR C (C)← 0 (Carry flag)

CLR P1.0

8051 Instruction Sets

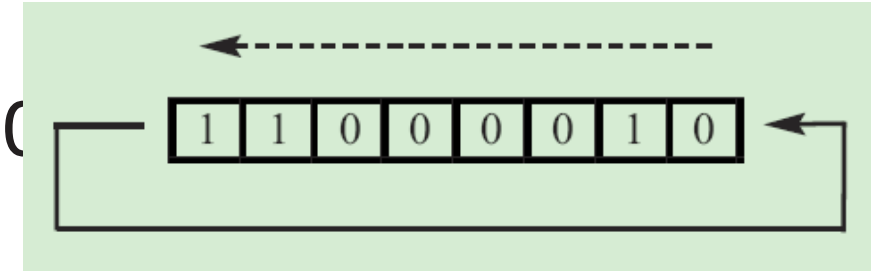
SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Logical Instructions
  - Instruction: CPL
  - Function: Complement Register
  - Syntax: CPL Operand
  - CPL complements operand, leaving the result in operand. The complement action changes 0s to 1s and 1s to 0s.
  - If operand is the Accumulator, then all the bits in the Accumulator will be reversed.
  - If the operand refers to a bit of an output Port, the value that will be complemented is based on the last value written to that bit, not the last value read from it.

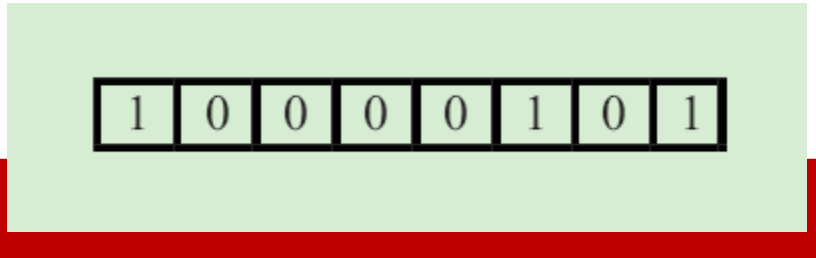
8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Rotate Instructions
  - Instruction: RL
  - Function: Rotate Accumulator Left
  - Syntax: RL A
  - This instruction rotates the left most bit (bit 7)of the accumulator to the left side by one such that leftmost bit that is being rotated is again stored as the rightmost bit (bit 0) in the accumulator
    - **RL A**
      - Before execution: A= C2h (1100001(



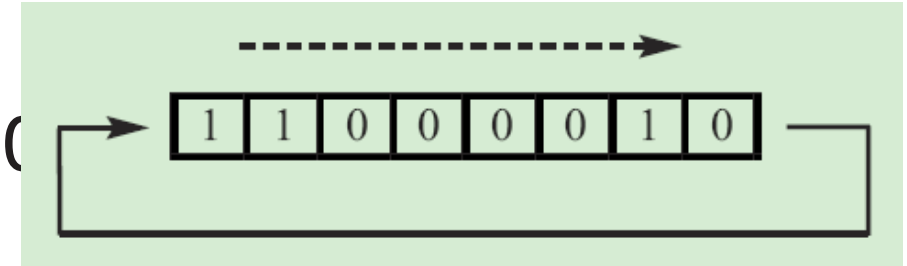
      - After execution: A=85h (10000101)



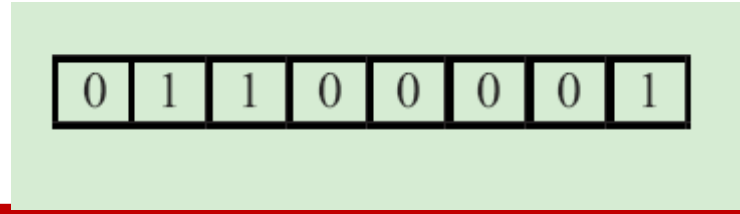
8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Rotate Instructions
  - Instruction: RR
  - Function: Rotate Accumulator Right
  - Syntax: RR A
  - This instruction rotates the right most bit (bit 0)of the accumulator to the right side by one such that right most bit that is being rotated is again stored as the left most bit (bit 0) in the accumulator
    - **RR A**
      - Before execution: A= C2h (1100001(



      - After execution: A= 61h (01100001)



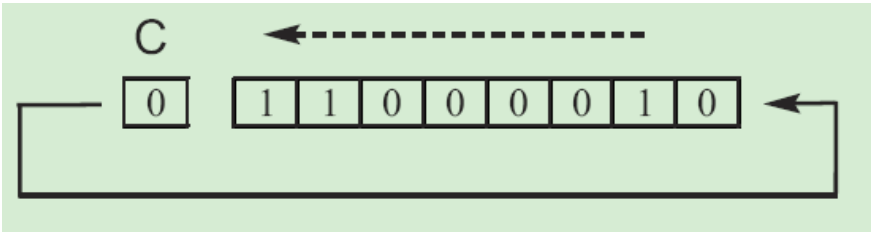
8051 Instruction Sets

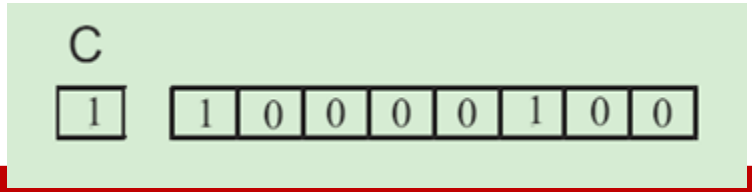
SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Rotate Instructions
  - Instruction: RLC
  - Function: Rotate Accumulator Left Through Carry
  - Syntax: RLC A
  - This instruction rotates the left most (bit 7) bit in the accumulator to the left side by one such that leftmost bit that is being rotated it is stored in the Carry Flag (CF), and the bit in the CF moved to the right most bit(bit 0) in the accumulator.
    - **RLC A**

Before execution: A= C2h (11000010)

After execution: A=84h (10000100) C=1



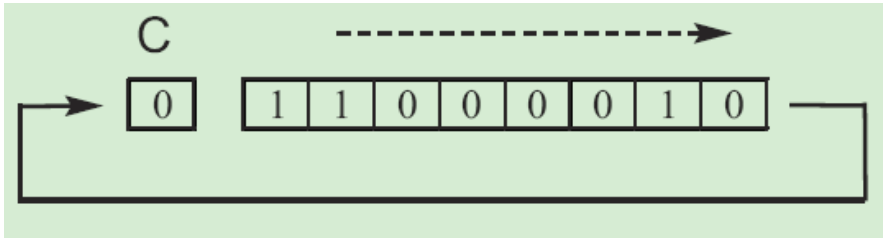


8051 Instruction Sets

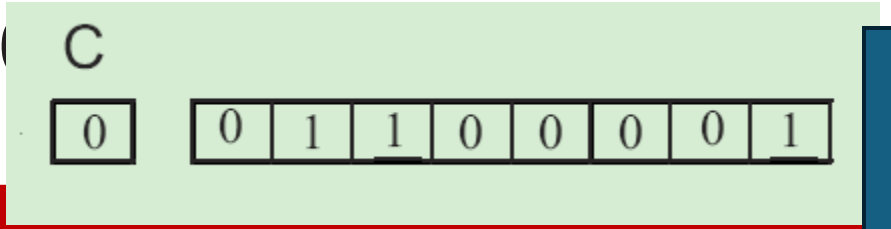
SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Rotate Instructions
  - Instruction: RRC
  - Function: Rotate Accumulator Right Through Carry
  - Syntax: RRC A
  - This instruction rotates the right most (bit 0) bit in the accumulator to the right side by one such that right most bit that is being rotated it is stored in the Carry Flag (CF), and the bit in the CF moved to the left most bit(bit 7) in the accumulator.
    - **RRC A**

Before execution: A= C2h (11000010)



After execution: A= 61h (01100001) C=



8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

DATA SERIALIZATION

- Serializing data is a way of sending a byte of data one bit at a time through a single pin of microcontroller.
- There are two ways to transfer a byte of data serially:
  - Using the serial port. In using the serial port, programmers have very limited control over the sequence of data transfer.
  - The second method of serializing data is to transfer data one bit at a time and control the sequence of data and spaces in between them.

Serializing a byte of data

- Serializing data is one of the most widely used applications of the rotate instruction. Using the rotate instruction, we transfer a byte of data serially (one bit at a time).

```
RRC   A          ;move the bit to CY
MOV   P1.3,C     ;output carry as data bit
```

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

DATA TRANSFER	*ARITHMETIC*	*LOGICAL & Rotation Instruction*	*Bit Control Instruction*	*Branching and looping instructions*
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
SWAP	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
			ORL	ACALL
			CPL	RET
				RETI
				JMP

8051 Instruction Sets

SYMBIOSIS INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control Instructions

  - Like logical instructions, these instructions (Some of) also perform logical operations.
  - The difference is that these operations are performed on single bits.

Bit Control Instruction
CLR
SETB
MOV
JC
JNC
JB
JNB
JBC
ANL
ORL
CPL

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control Instructions
  - CLR C (C = 0)
  - CLR bit clear directly addressed bit
  - SETB C (C = 1)
  - SETB bit Set directly addressed bit
  - CPL C (1 = 0, 0 = 1)
  - CPL bit Complement directly addressed bit
  - MOV C, bit Move directly addressed bit to carry bit.
  - MOV bit, C Move Carry bit to directly addressed bit.
  -

8051 Instruction Sets

- Bit Control Instructions
  - ANL C, bit
    - Logical AND operation between Carry bit and directly addressed bit.
  - ANL C,/bit
    - Logical AND operation between Carry bit and inverted directly addressed bit.
  - ORL C, bit
    - Logical OR operation between Carry bit and directly addressed bit.
  - ORL C,/bit
    - Logical OR operation between Carry bit and inverted directly addressed bit.

8051 Instruction Sets

- Bit Control / Jump Instructions
  - Instruction: JC
  - Function: Jump if Carry Set
  - Syntax: JC Addr
    - JC will branch to the address indicated by "Addr" if the Carry Bit is set.
    - If the Carry Bit is not set program execution continues with the instruction following the JC instruction. The value of operand is not affected.
    - This is a short jump instruction, which means that the address of a new location must be relatively near the current locations relative to the first following instruction

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control / Jump Instructions
  - Instruction: JNC
  - Function: Jump if Carry Not Set
  - Syntax: JNC Addr
    - JNC will branch to the address indicated by “Addr” if the Carry Bit is not set.
    - If the Carry Bit is set program execution continues with the instruction following the JNC instruction.
    - This is a short jump instruction, which means that the address of a new location must be relatively near the current locations relative to the first following instruction

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control / Jump Instructions
  - Instruction: JB
  - Function: Jump if Bit Set
  - Syntax: JB Bit_addr, Addr
    - JB branches to the address indicated by "Aaddr" 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.
    - This is a short jump instruction, which means that the address of a new location must be relatively near the current locations relative to the first following instruction

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control / Jump Instructions
  - Instruction: JNB
  - Function: Jump if Bit is not Set
  - Syntax: JNB Bit_addr, Addr
    - JNB branches to the address indicated by "Aaddr" if the bit indicated by bit_addr is not set. If the bit is set program execution continues with the instruction following the JB instruction.
    - This is a short jump instruction, which means that the address of a new location must be relatively near the current locations relative to the first following instruction

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control / Jump Instructions
  - Instruction: JBC
  - Function: Jump if Bit Set and Clear Bit
  - Syntax: JBC Bit_addr, Addr
    - JBC will branch to the address indicated by “Aaddr” if the bit indicated by bit addr is set. Before branching to reladdr the instruction will clear the indicated bit.
    - If the bit is not set program execution continues with the instruction following the JBC instruction.

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control / Jump Instructions
  - Instruction: JZ
  - Function: Jump if Accumulator Zero
  - Syntax: JZ, Addr
    - JZ will branch to the address indicated by “Aaddr if the Accumulator contains the value 0. If the value of the Accumulator is non-zero program execution continues with the instruction following the JZ instruction

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control / Jump Instructions
  - Instruction: JNZ
  - Function: Jump if Accumulator is not Zero
  - Syntax: JZ, Addr
    - JNZ will branch to the address indicated by “Aaddr” if the Accumulator contains any value except 0. If the value of the Accumulator is zero program execution continues with the instruction following the JZ instruction

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Bit Control / Jump Instructions
  - Instruction: CJNE
  - Function: Compare and Jump If Not Equal
  - Syntax: CJNE operand1,operand2, Addr
    - 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.

8051 Instruction Sets

- Bit Control / Jump Instructions
- Instruction: DJNZ
- Function: Decrement and Jump if Not Zero
- Syntax: DJNZ register, Addr
    - 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 “Addr”.
    - If the new value of register is 0 program flow continues with th instruction following the DJNZ instruction.

8051 Instruction Sets

- Jump/Branch Instructions
- Instruction: LJMP
  - Function: Long Jump
  - Syntax: LJMP code addr
    - LJMP jumps unconditionally to the specified code addr
- Instruction: SJMP
- Function: Short Jump
- Syntax: SJMP addr
    - SJMP jumps unconditionally to the address specified “addr”. “addr” must be within -128 or +127 bytes of the instruction that follows the SJMP instruction.

8051 Instruction Sets

- Jump/Beanch Instructions
- Instruction: LCALL
- Function: Long call
- Syntax: LCALL code address
    - LCALL calls a program subroutine. LCALL increments the program counter by 3 (to point to the instruction following LCALL) and pushes that value onto the stack (low byte first, high byte second).
    - The Program Counter is then set to the 16-bit value which follows the LCALL opcode, causing program execution to continue at tha address.

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Jump/Beanch Instructions
- Instruction: RET
- Function: Return From Subroutine
- Syntax: RET
    - RET is used to return from a subroutine previously called by LCALL or ACALL. Program execution continues at the address that is calculated by popping the topmost 2 bytes off the stack. The most-significant-byte is popped off the stack first, followed by the least-significant-byte.

8051 Instruction Sets

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

- Jump/Beanch Instructions
- Instruction: RETI
- Function: Return From Interrupt
- Syntax: RETI
  - RETI is used to return from an interrupt service routine. RETI first enables interrupts of equal and lower priorities to the interrupt that is terminating. Program execution continues at the address that is calculated by popping the topmost 2 bytes off the stack. The most-significant-byte is popped off the stack first, followed by the least-significant-byte.
  - RETI functions identically to RET if it is executed outside of an interrupt service routine.

Assembler Directives

Assembler directives tell the assembler to do something other than creating the machine code for an

instruction. In assembly language programming, the assembler directives instruct the assembler to

1. Process subsequent assembly language instructions
2. Define program constants
3. Reserve space for variables

The following are the widely used 8051 assembler directives.

ORG (origin)

The ORG directive is used to indicate the starting address. It can be used only when the program counter needs to be changed. The number that comes after ORG can be either in hex or in decimal.

Eg: ORG 0000H ;Set PC to 0000.

Assembler Directives

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

EQU and SET

EQU and SET directives assign numerical value or register name to the specified symbol name.

EQU is used to define a constant without storing information in the memory. The symbol defined with EQU should not be redefined.

SET directive allows redefinition of symbols at a later stage.

DB (DEFINE BYTE)

The DB directive is used to define an 8 bit data. DB directive initializes memory with 8 bit values. The numbers can be in decimal, binary, hex or in ASCII formats. For decimal, the 'D' after the decimal number is optional, but for binary and hexadecimal, 'B' and ‘H’ are required. For ASCII, the number is written in quotation marks (‘LIKE This).

DATA1: DB 40H ; hex

DATA2: DB 01011100B ; b i n a r y

DATA3: DB 48 ; decimal

DATA4: DB ' HELLO W’ ; ASCII

Assembler Directives

END

The END directive signals the end of the assembly module. It indicates the end of the program to the assembler. Any text in the assembly file that appears after the END directive is ignored. If the END statement is missing, the assembler will generate an error message.

Assembler Directives

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

ASSEMBLY LANGUAGE PROGRAMS.

1. Write a program to add the values of locations 50H and 51H and store the result in locations

in 52h and 53H.

```
ORG 0000H ; Set program counter 0000H
MOV A,50H ; Load the contents of Memory location 50H into A ADD
ADD A,51H ; Add the contents of memory 51H with CONTENTS A
MOV 52H,A ; Save the LS byte of the result in 52H
MOV A, #00 ; Load 00H into A
ADDC A, #00 ; Add the immediate data and carry to A
MOV 53H,A ; Save the MS byte of the result in location 53h
END
```

Assembler Directives

SYMBIOSIS
INSTITUTE OF TECHNOLOGY, NAGPUR

2. Write a program to store data FFH into RAM memory locations 50H to 58H using direct addressing mode

```
ORG 0000H ; Set program counter 0000H
MOV A, #0FFH ; Load FFH into A
MOV 50H, A ; Store contents of A in location 50H
MOV 51H, A ; Store contents of A in location 5IH
MOV 52H, A ; Store contents of A in location 52H
MOV 53H, A ; Store contents of A in location 53H
MOV 54H, A ; Store contents of A in location 54H
MOV 55H, A ; Store contents of A in location 55H
MOV 56H, A ; Store contents of A in location 56H
MOV 57H, A ; Store contents of A in location 57H
MOV 58H, A ; Store contents of A in location 58H
END
```

Assembler Directives

3. Write a program to add two 16 bit numbers stored at locations 51H-52H and 55H-56H and store the result in locations 40H, 41H and 42H. Assume that the least significant byte of data and the result is stored in low address and the most significant byte of data or the result is stored in high address.

```
ORG 0000H ; Set program counter 0000H
MOV A,51H ; Load the contents of memory location 51H into A
ADD A,55H ; Add the contents of 55H with contents of A
MOV 40H,A ; Save the LS byte of the result in location 40H
MOV A,52H ; Load the contents of 52H into A
ADDC A,56H ; Add the contents of 56H and CY flag with A
MOV 41H,A ; Save the second byte of the result in 41H
MOV A,#00 ; Load 00H into A
ADDC A,#00 ; Add the immediate data 00H and CY to A
MOV 42H,A ; Save the MS byte of the result in location 42H
END
```

Assembler Directives

4. Write a program to store data FFH into RAM memory locations 50H to 58H using indirect

addressing mode.

```
ORG 0000H ; Set program counter 0000H
MOV A, #0FFH ; Load FFH into A
MOV RO, #50H ; Load pointer, R0-50H
MOV R5, #08H ; Load counter, R5-08H
Start:MOV @RO, A ; Copy contents of A to RAM pointed by R0
INC RO ; Increment pointer
DJNZ R5, start ; Repeat until R5 is zero
END
```