8051 MICROCONTROLLER

- What is Microcontroller?
- Why the name Microcontroller?
- Why it is required?
- Where it is used?

Microprocessors and Microcontrollers

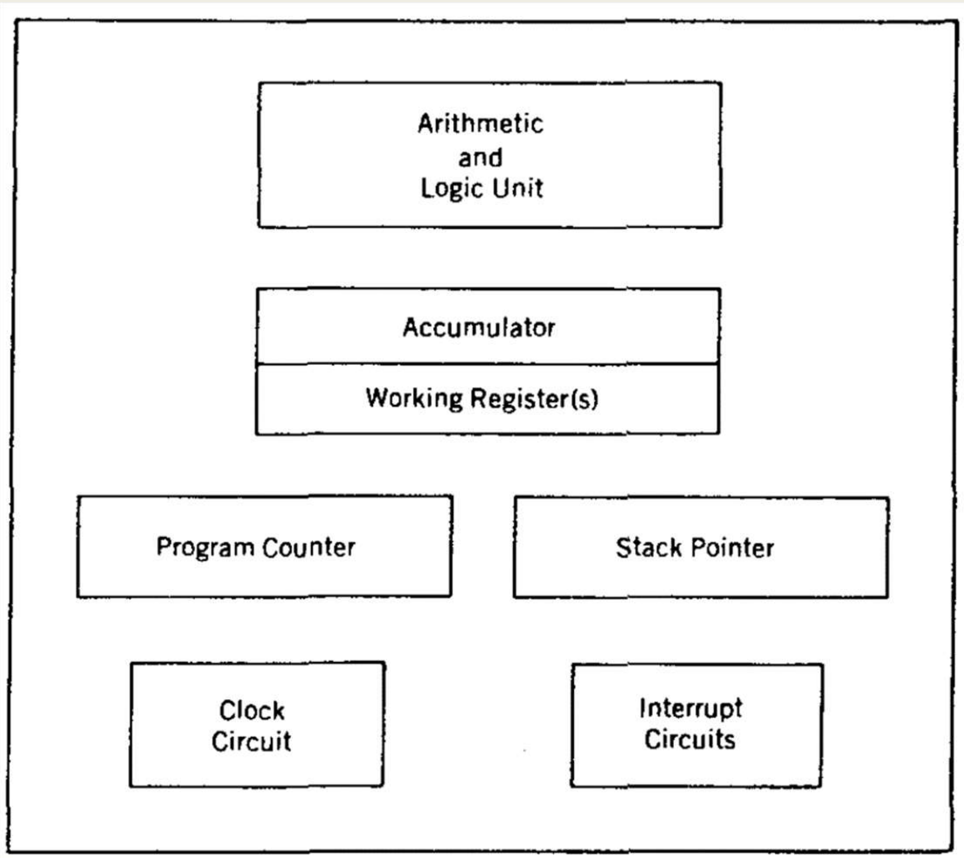
Microprocessors

- A microprocessor is a general purpose digital computer CPU.
- It contains an ALU, a program counter, a stack pointer, some registers, a clock timing circuit, and interrupt circuits.
- To make a complete computer. memory(RAM & ROM), memory decoders, an oscillator, and a number of I/O devices, such as parallel and serial data ports are required.
- The prime use of a microprocessor is to read data, perform extensive calculations on that data, and store the results in memory or storage device.

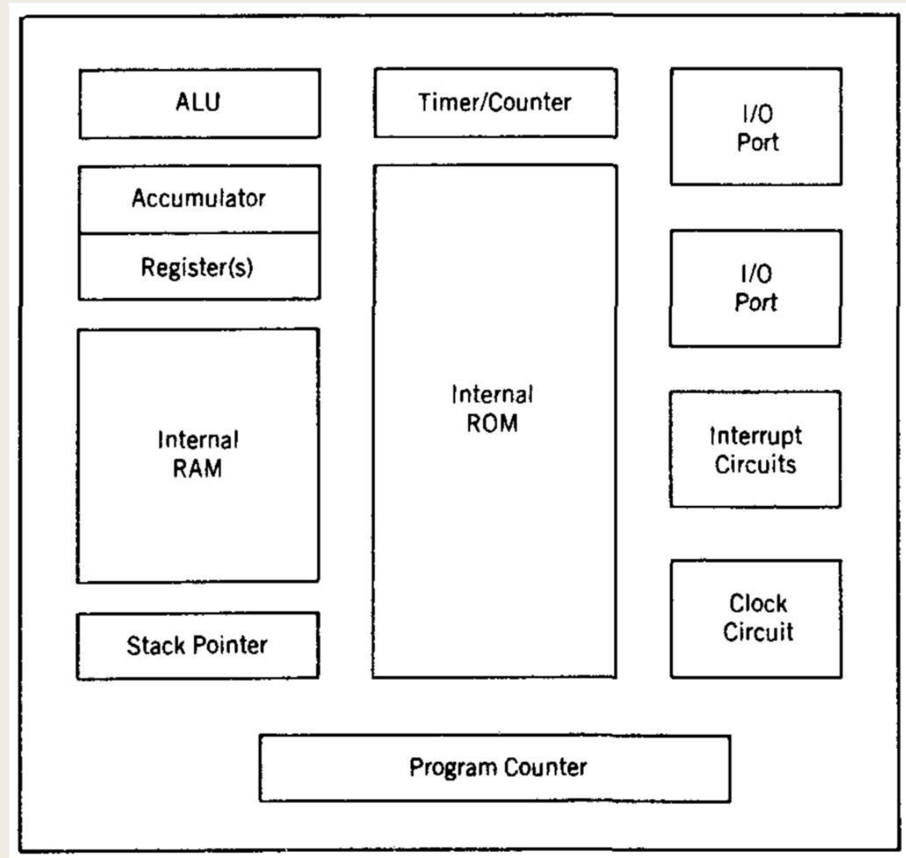
Microcontrollers

- A microcontroller is also called as computer on a chip.
- It is actually a microprocessor + ROM, RAM, parallel I/O, serial I/O, counters, and a clock circuit.
- The prime use of a microcontroller is to control the operation of a machine using a fixed program that is stored in ROM and that does not change over the lifetime of the system.

MICROPROCESSOR



MICROCONTROLLER

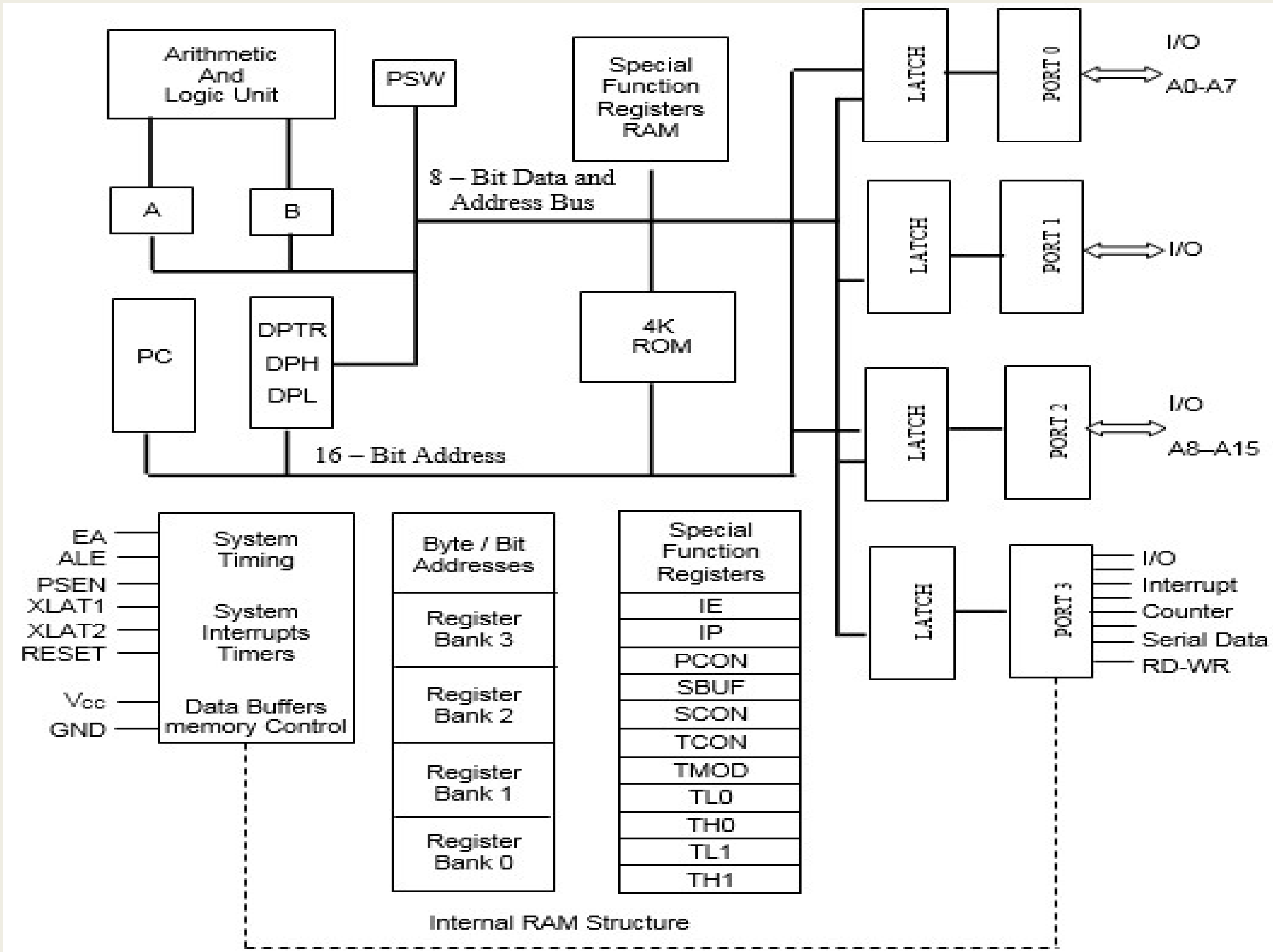


No.	Microprocessor	Microcontroller
1.	Microprocessor contains ALU, control unit (clock and timing circuit), different register and interrupt circuit.	Microcontroller contains microprocessor, memory (ROM and RAM), I/O interfacing circuit and peripheral devices such as A/D converter, serial I/O, timer etc.
2.	It has many instructions to move data between memory and CPU.	It has one or two instructions to move data between memory and CPU.
3.	It has one or two bit handling instructions.	It has many bit handling instructions.
4.	Access times for memory and I/O devices are more.	Less access times for built-in memory and I/O devices.
5.	Microprocessor based system requires more hardware.	Microcontroller based system requires less hardware reducing PCB size and increasing the reliability.
6.	Microprocessor based system is more flexible in design point of view.	Less flexible in design point of view.
7.	It has single memory map for data and code.	It has separate memory map for data and code.
8.	Less number of pins are multifunctioned.	More number pins are multifunctioned.

Features of 8051

- 4KB on-chip program memory (ROM/EPROM).
- 128 bytes on-chip data memory.
- Four register banks.
- 64KB each program and external RAM addressability.
- One microsecond instruction cycle with 12MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports.
- Multiple modes, high-speed programmable serial port (UART).
- 16-bit Timers/Counters.
- Direct byte and bit addressability.

8051 Block Diagram



- Oscillator and clock generator: All operations in a microcontroller are synchronized by the help of an oscillator clock. The oscillator clock generates the clock pulses by which all internal operations are synchronized.
- ALU: It is 8 bit unit. It performs arithmetic operation as addition, subtraction, multiplication, division, increment and decrement. It performs logical operations like AND, OR and EX-OR. It manipulates 8 bit and 16 bit data. It calculates address of jump locations in relative branch instruction. It performs compare, rotate and compliment operations. It consists of Boolean processor which performs bit, set, test, clear and compliment. 8051 micro controller contains 34 general purpose registers or working registers.2 of them are called math registers A & B and 32 are bank of registers.
- Accumulator(A-reg):It is 8 bit register. Its address is E0H and it is bit and byte accessible. Result of arithmetic & logic operations performed by ALU is accumulated by this register. Therefore it is called accumulator register. It is used to store 8 bit data and to hold one of operand of ALU units during arithmetical and logical operations. Most of the instructions are carried out on accumulator data. It is most versatile of 2 CPU registers.
- B-register: It is special 8 bit math register. It is bit and byte accessible. It is used in conjunction with A register as I/P operand for ALU. It is used as general purpose register to store 8 bit data.
- PSW: It is 8 bit register. It has 4 conditional flags or math flags which sets or resets according to condition of result. It has 3 control flags, by setting or resetting bit required operation or function can be achieved.

- The format of flag register is as shown below:
- i. MATH FLAG:1. Carry Flag(CY): During addition and subtraction any carry or borrow is generated then carry flag is set otherwise carry flag resets. It is used in arithmetic, logical, jump, rotate and Boolean operations.
- 2. Auxiliary carry flag(AC): If during addition and subtraction any carry or borrow is generated from lower 4 bit to higher 4 bit then AC sets else it resets. It is used in BCD arithmetic operations.
- 3. Overflow flag(OV): If in signed arithmetic operations result exceeds more than 7 bit than OV flag sets else resets.It is used in signed arithmetic operations only.
- 4. Parity flag(P): If in result, even no. Of ones "1" are present than it is called even parity and parity flag sets. In result odd no. Of ones "1"are present than it is called odd parity and parity flag resets.
- ii. CONTROL FLAGS:1. FO: It is user defined flag. The user defines the function of this flag. The user can set ,test n clear this flag through software.
- 2. RS1 and RS0: These flags are used to select bank of register by resetting those flags which are as shown in table :

- Program counter(PC): The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory.
- Data pointer register(DTPR): It is a 16 bit register used to hold address of external or internal RAM where data is stored or result is to be stored. It is used to store 16 bit data. It is divided into2- 8bit registers, DPH-data pointer higher order (83H) and DPL-data pointer lower order (82H).
- Stack pointer(SP): It is 8-bit register. It is byte addressable. Its address is 81H. It is used to hold the internal RAM memory location addresses which are used as stack memory. When the data is to be placed on stack by push instruction, the content of stack pointer is incremented by 1, and when data is retrieved from stack, content of stack of stack pointer is decremented by 1.

Special function Registers(SFR)

- The 8051 microcontroller has 11 SFR divided in 4 groups:
- A. Timer/Counter register: 8051 microcontroller has 2-16 bit Timer/counter registers called Timer-reg-T0 And Timer/counter Reg-T1.Each register is 16 bit register divide into lower and higher byte register as shown below: These registers are used to hold initial no. of count. All of the 4 registers are byte addressable.

Timer control register: 8051 microcontroller has two 8-bit timer control register i.e. TMOD and TCON register.

- TMOD Register: it is 8-bit register. Its address is 89H. It is byte addressable. It used to select mode and control operation of time by writing control word.
- TCON register: It is 8-bit register. Its address is 88H. It is byte addressable. Its MSB 4-bit are used to control operation of timer/ counter and LSB 4-bit are used for external interrupt control.

Serial data register: 8051 micro controller has two serial data register viz. SBUF and SCON.

- Serial buffer register (SBUF): it is 8-bit register. It is byte addressable .Its address is 99H. It is used to hold data which is to be transferred serially.
- Serial control register (SCON): it is 8-bit register. It is bit/byte addressable. Its address is 98H. The 8-bit loaded into this register controls the operation of serial communication.

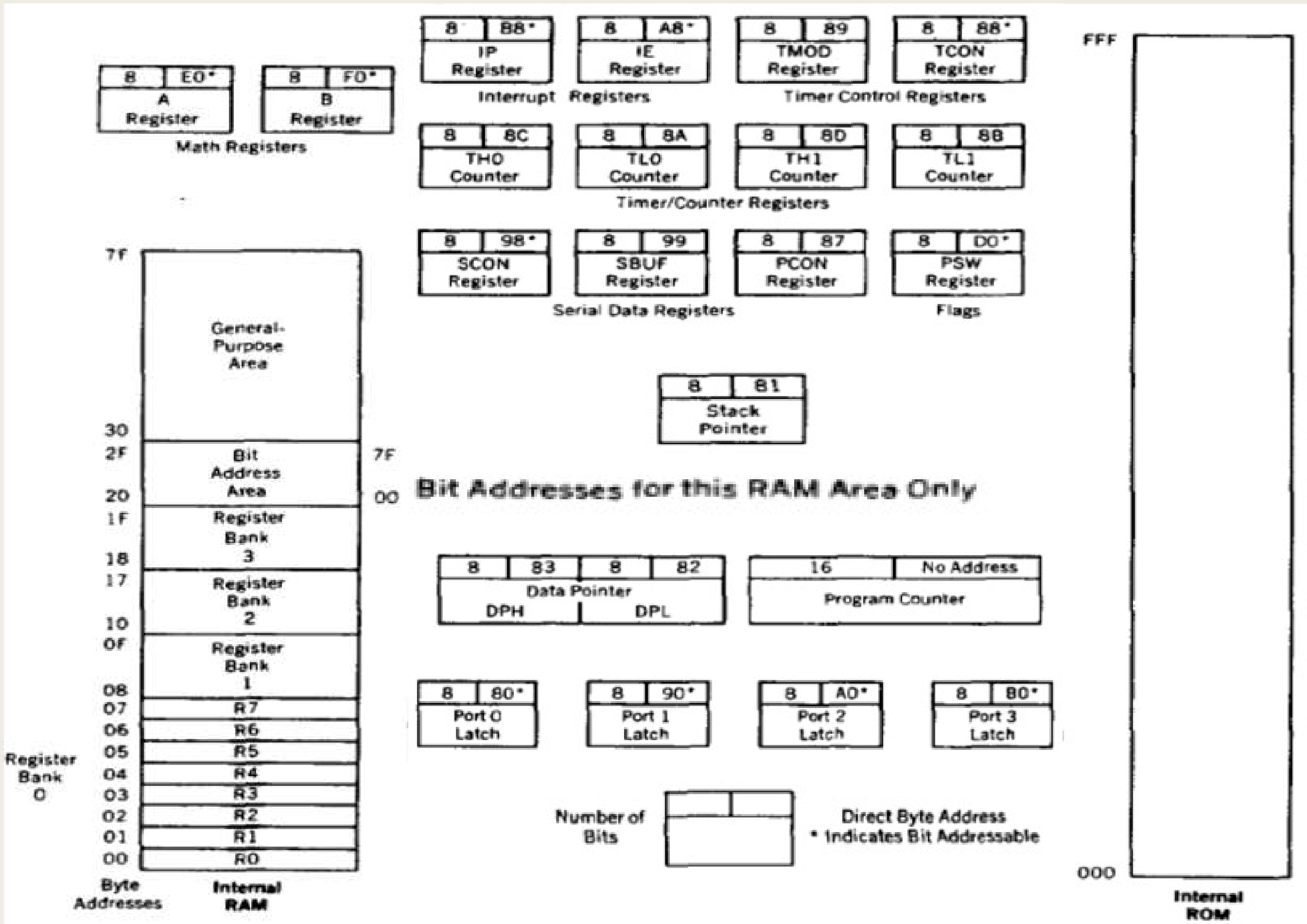
Interrupt register: 8051 µC has two 8-bit interrupt register.

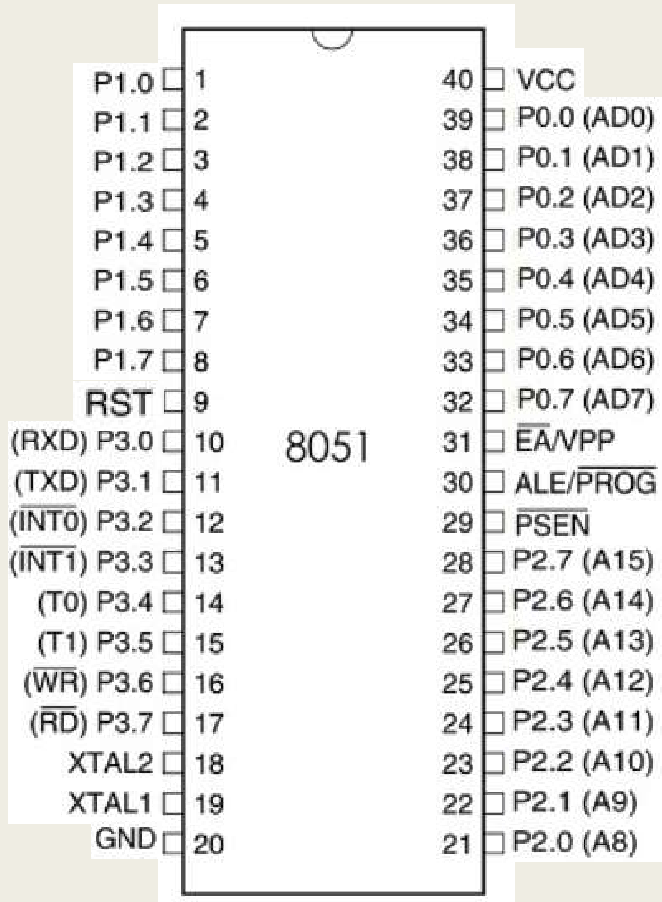
- Interrupt enable register (IE): it is 8-bit register. It is bit/byte addressable. Its address is A8H.it is used to enable and disable function of interrupt.
- Interrupt priority register (IP): It is 8-bit register. It is bit/byte addressable. Its address is B8H. it is used to select low or high level priority of each individual interrupts.

Power control register (PCON): it is 8-bit register.

It is byte addressable .Its address is 87H. its bits are used to control mode of power saving circuit, either idle or power down mode and also one bit is used to modify baud rate of serial communication.

8051 Programming Model




8051
P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(RXD) P3.0 10
(TXD) P3.1 11
($\overline{INT0}$) P3.2 12
($\overline{INT1}$) P3.3 13
(T0) P3.4 14
(T1) P3.5 15
($\overline{WR}$) P3.6 16
($\overline{RD}$) P3.7 17
XTAL2 18
XTAL1 19
GND 20
40 VCC
39 P0.0 (AD0)
38 P0.1 (AD1)
37 P0.2 (AD2)
36 P0.3 (AD3)
35 P0.4 (AD4)
34 P0.5 (AD5)
33 P0.6 (AD6)
32 P0.7 (AD7)
31 $\overline{EA}$/VPP
30 ALE/$\overline{PROG}$
29 $\overline{PSEN}$
28 P2.7 (A15)
27 P2.6 (A14)
26 P2.5 (A13)
25 P2.4 (A12)
24 P2.3 (A11)
23 P2.2 (A10)
22 P2.1 (A9)
21 P2.0 (A8)

8051 MICROCONTROLLER

- What is Microcontroller?
- Why the name Microcontroller?
- Why it is required?
- Where it is used?

Microprocessors and Microcontrollers

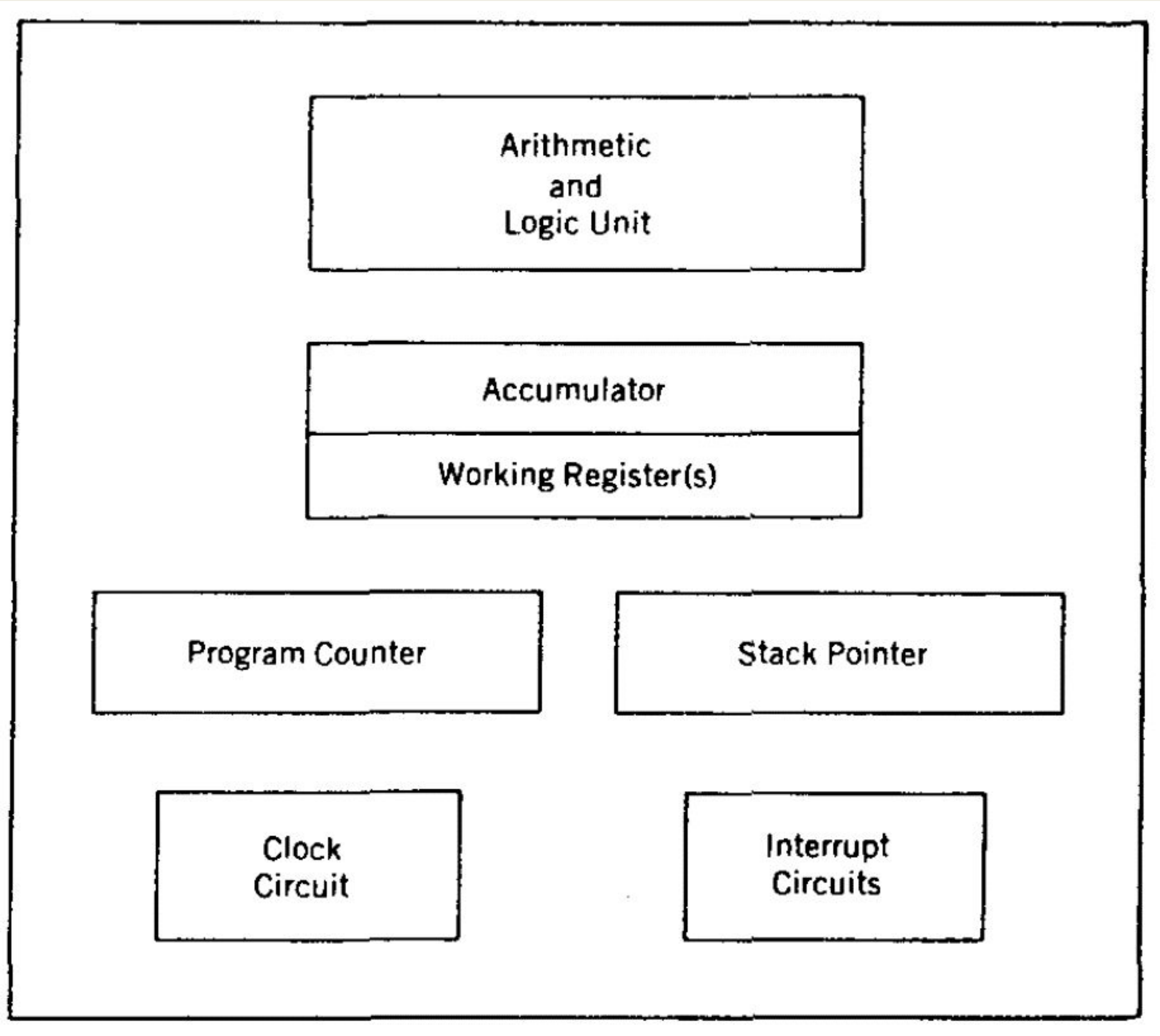
Microprocessors

- A microprocessor is a general purpose digital computer CPU.
- It contains an ALU, a program counter, a stack pointer, some registers, a clock timing circuit, and interrupt circuits.
- To make a complete computer. memory(RAM & ROM), memory decoders, an oscillator, and a number of I/O devices, such as parallel and serial data ports are required.
- The prime use of a microprocessor is to read data, perform extensive calculations on that data, and store the results in memory or storage device.

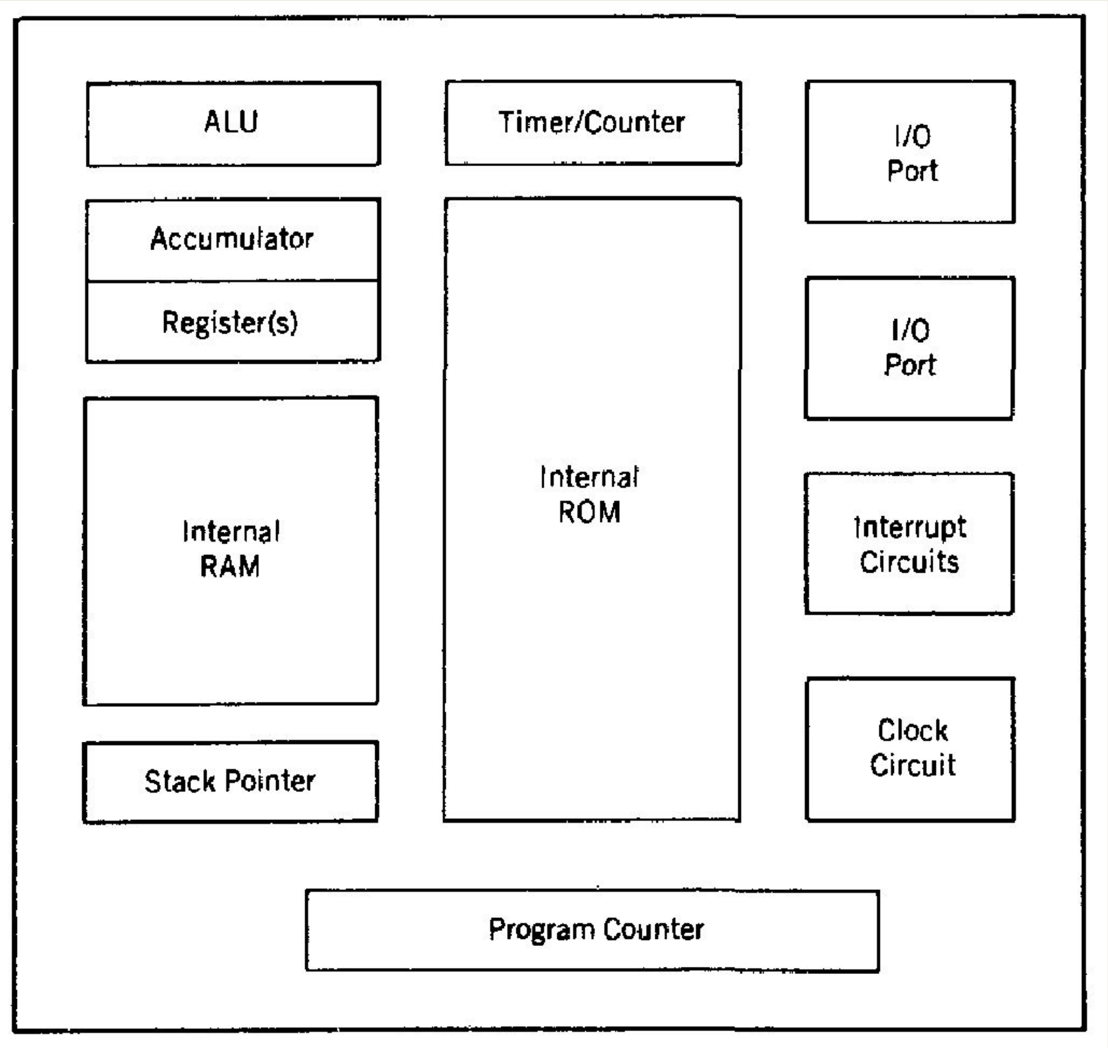
Microcontrollers

- A microcontroller is also called as computer on a chip.
- It is actually a microprocessor + ROM, RAM, parallel I/O, serial I/O, counters, and a clock circuit.
- The prime use of a microcontroller is to control the operation of a machine using a fixed program that is stored in ROM and that does not change over the lifetime of the system.

MICROPROCESSOR



MICROCONTROLLER

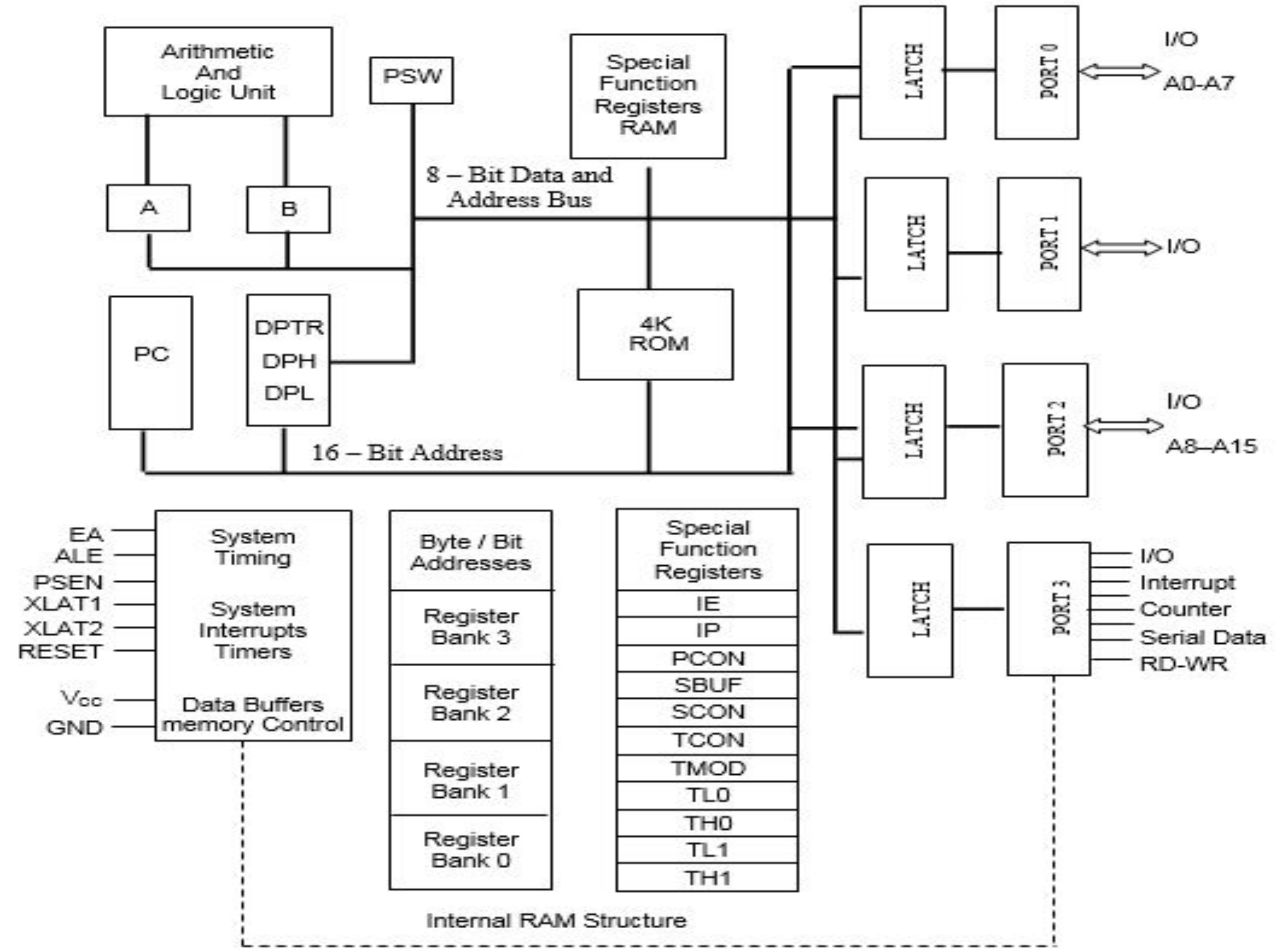


No.	Microprocessor	Microcontroller
1.	Microprocessor contains ALU, control unit (clock and timing circuit), different register and interrupt circuit.	Microcontroller contains microprocessor, memory (ROM and RAM), I/O interfacing circuit and peripheral devices such as A/D converter, serial I/O, timer etc.
2.	It has many instructions to move data between memory and CPU.	It has one or two instructions to move data between memory and CPU.
3.	It has one or two bit handling instructions.	It has many bit handling instructions.
4.	Access times for memory and I/O devices are more.	Less access times for built-in memory and I/O devices.
5.	Microprocessor based system requires more hardware.	Microcontroller based system requires less hardware reducing PCB size and increasing the reliability.
6.	Microprocessor based system is more flexible in design point of view.	Less flexible in design point of view.
7.	It has single memory map for data and code.	It has separate memory map for data and code.
8.	Less number of pins are multifunctioned.	More number pins are multifunctioned.

Features of 8051

- 4KB on-chip program memory (ROM/EPROM).
- 128 bytes on-chip data memory.
- Four register banks.
- 64KB each program and external RAM addressability.
- One microsecond instruction cycle with 12MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports.
- Multiple modes, high-speed programmable serial port (UART).
- 16-bit Timers/Counters.
- Direct byte and bit addressability.

8051 Block Diagram



- **Oscillator and clock generator**: All operations in a microcontroller are synchronized by the help of an oscillator clock. The oscillator clock generates the clock pulses by which all internal operations are synchronized.
- **ALU:** It is 8 bit unit. It performs arithmetic operation as addition, subtraction, multiplication, division, increment and decrement. It performs logical operations like AND, OR and EX-OR. It manipulates 8 bit and 16 bit data. It calculates address of jump locations in relative branch instruction. It performs compare, rotate and compliment operations. It consists of Boolean processor which performs bit, set, test, clear and compliment. 8051 micro controller contains 34 general purpose registers or working registers.2 of them are called math registers A & B and 32 are bank of registers.
- **Accumulator(A-reg):**It is 8 bit register. Its address is E0H and it is bit and byte accessible. Result of arithmetic & logic operations performed by ALU is accumulated by this register. Therefore it is called accumulator register. It is used to store 8 bit data and to hold one of operand of ALU units during arithmetical and logical operations. Most of the instructions are carried out on accumulator data. It is most versatile of 2 CPU registers.
- **B-register:** It is special 8 bit math register. It is bit and byte accessible. It is used in conjunction with A register as I/P operand for ALU. It is used as general purpose register to store 8 bit data.
- **PSW:** It is 8 bit register. It has 4 conditional flags or math flags which sets or resets according to condition of result. It has 3 control flags, by setting or resetting bit required operation or function can be achieved.

- The format of flag register is as shown below:
- i. **MATH FLAG**:1. Carry Flag(CY): During addition and subtraction any carry or borrow is generated then carry flag is set otherwise carry flag resets. It is used in arithmetic, logical, jump, rotate and Boolean operations.
- 2. Auxiliary carry flag(AC): If during addition and subtraction any carry or borrow is generated from lower 4 bit to higher 4 bit then AC sets else it resets. It is used in BCD arithmetic operations.
- 3. Overflow flag(OV): If in signed arithmetic operations result exceeds more than 7 bit than OV flag sets else resets.It is used in signed arithmetic operations only.
- 4. Parity flag(P): If in result, even no. Of ones "1" are present than it is called even parity and parity flag sets. In result odd no. Of ones "1"are present than it is called odd parity and parity flag resets.
- ii. **CONTROL FLAGS**:1. FO: It is user defined flag. The user defines the function of this flag. The user can set ,test n clear this flag through software.
- 2. RS1 and RS0: These flags are used to select bank of register by resetting those flags which are as shown in table :

- **<u>Program counter(PC):</u>** The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory.
- **<u>Data pointer register(DTPR)</u>**: It is a 16 bit register used to hold address of external or internal RAM where data is stored or result is to be stored. It is used to store 16 bit data. It is divided into2- 8bit registers, DPH-data pointer higher order (83H) and DPL-data pointer lower order (82H).
- **<u>Stack pointer(SP):</u>** It is 8-bit register. It is byte addressable. Its address is 81H. It is used to hold the internal RAM memory location addresses which are used as stack memory. When the data is to be placed on stack by push instruction, the content of stack pointer is incremented by 1, and when data is retrieved from stack, content of stack of stack pointer is decremented by 1.

Special function Registers(SFR)

- **The 8051 microcontroller has 11 SFR divided in 4 groups:**
- A. **Timer/Counter register:** 8051 microcontroller has 2-16 bit Timer/counter registers called Timer-reg-T0 And Timer/counter Reg-T1.Each register is 16 bit register divide into lower and higher byte register as shown below: These registers are used to hold initial no. of count. All of the 4 registers are byte addressable.

Timer control register: 8051 microcontroller has two 8-bit timer control register i.e. TMOD and TCON register.

- TMOD Register: it is 8-bit register. Its address is 89H. It is byte addressable. It used to select mode and control operation of time by writing control word.
- TCON register: It is 8-bit register. Its address is 88H. It is byte addressable. Its MSB 4-bit are used to control operation of timer/ counter and LSB 4-bit are used for external interrupt control.

Serial data register: 8051 micro controller has two serial data register viz. SBUF and SCON.

- Serial buffer register (SBUF): it is 8-bit register. It is byte addressable .Its address is 99H. It is used to hold data which is to be transferred serially.
- Serial control register (SCON): it is 8-bit register. It is bit/byte addressable. Its address is 98H. The 8-bit loaded into this register controls the operation of serial communication.

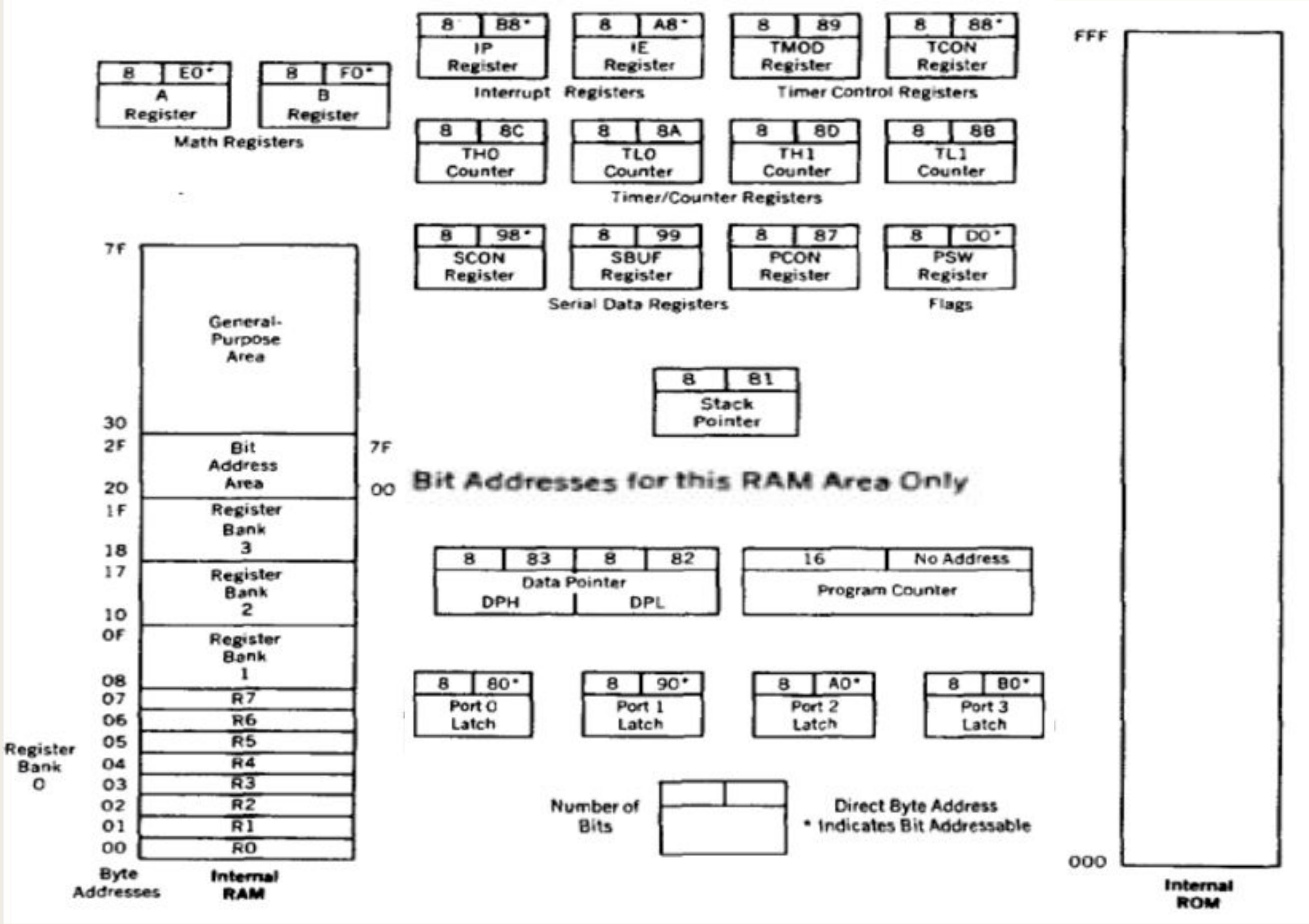
Interrupt register: 8051 µC has two 8-bit interrupt register.

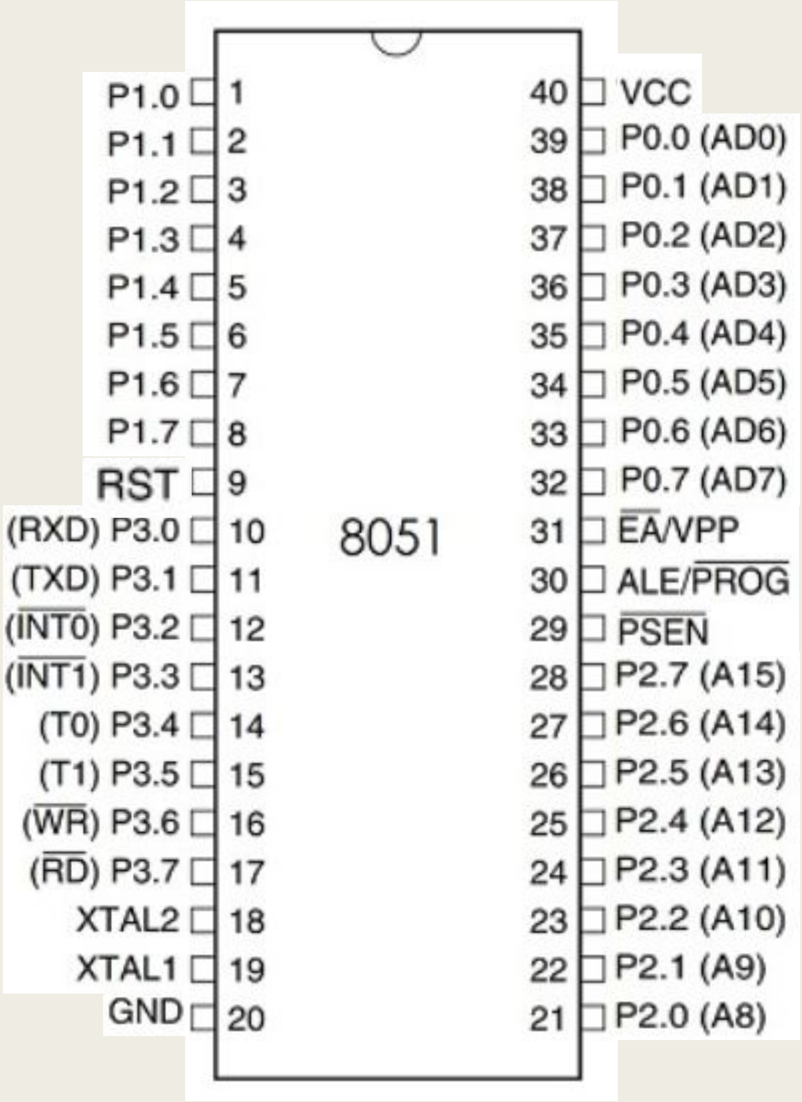
- Interrupt enable register (IE): it is 8-bit register. It is bit/byte addressable. Its address is A8H.it is used to enable and disable function of interrupt.
- Interrupt priority register (IP): It is 8-bit register. It is bit/byte addressable. Its address is B8H. it is used to select low or high level priority of each individual interrupts.

Power control register (PCON): it is 8-bit register.

It is byte addressable .Its address is 87H. its bits are used to control mode of power saving circuit, either idle or power down mode and also one bit is used to modify baud rate of serial communication.

8051 Programming Model




P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(RXD) P3.0 10
(TXD) P3.1 11
(INT0) P3.2 12
(INT1) P3.3 13
(T0) P3.4 14
(T1) P3.5 15
(WR) P3.6 16
(RD) P3.7 17
XTAL2 18
XTAL1 19
GND 20
8051
40 VCC
39 P0.0 (AD0)
38 P0.1 (AD1)
37 P0.2 (AD2)
36 P0.3 (AD3)
35 P0.4 (AD4)
34 P0.5 (AD5)
33 P0.6 (AD6)
32 P0.7 (AD7)
31 EA/VPP
30 ALE/PROG
29 PSEN
28 P2.7 (A15)
27 P2.6 (A14)
26 P2.5 (A13)
25 P2.4 (A12)
24 P2.3 (A11)
23 P2.2 (A10)
22 P2.1 (A9)
21 P2.0 (A8)

Data Type

- DB – Define Byte
- The 8051 Microcontroller has only one data type DB.
- It is 8-bits and the size of each register is also 8-bits.
- It is used to define data, the numbers can be in decimal, binary, hex, or ASCII formats.
- Examples:
  - DB 28 ; DECIMAL
  - DB 39H ; HEX
  - DB 00110101B : BINARY
  - DB “2593” : ASCII NUMBERS
  - DB “Hello World” : ASCII CHARACTERS

ASSEMBLER DIRECTIVES

- ORG
- END
- EQU

The Elements of Assembly Language Programming

- Assembler Directives
- Instruction Set
- Addressing Modes

Addressing Modes

- The way of accessing data is called addressing mode. The CPU can access the data in different ways by using addressing modes.
- The 8051 microcontroller consists of five addressing modes such as:
- Immediate Addressing Mode
- Register Addressing Mode
- Direct Addressing Mode
- Indirect Addressing Mode
- Base Index Addressing Mode

Immediate Addressing Mode:

- In this addressing mode, the source must be a value that can be followed by the '#' and destination must be SFR registers, general purpose registers and address. It is used for immediately storing the value in the memory registers.
- Syntax:
- MOV A, #20h //A is an accumulator register, 20 is stored in the A//
- MOV R0,#15 // R0 is a general purpose register; 15 is stored in the R0 register//
- MOV P0, #07h //P0 is a SFR register;07 is stored in the P0//
- MOV 20h,#05h //20h is the address of the register; 05 stored in the 20h//

Register Addressing Mode

- Certain register names may be used as part of the opcode mnemonic as sources or destinations of data. Registers A, DPTR, and RO to R7 may be named as part of the opcode mnemonic.

```
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
MOV DPTR, A ; will give an error
MOV DPTR,#25F5H ;
MOV R7, DPL ;
MOV R6, DPH ;
MOV R4, R7 ; The data transfer between Rn registers is not allowed
```

Direct Addressing Mode

- All 128 bytes of internal RAM and the SFRs may be addressed directly using the single byte address assigned to each RAM location and each special- function register.

  MOV R0, 40H ; save content of 40H in R0

  MOV 56H, A ; save content of A in 56H

  MOV A, 4 ; is same as

  MOV A, R4 ; which means copy R4 into A

- The SFR (Special Function Register) can be accessed by their names or by their addresses:

  MOV 0E0H, #55H ; is the same as

  MOV A, #55h ; load 55H into A

  MOV 0F0H, R0 ; is the same as

  MOV B, R0 ; copy R0 into B

Register Indirect Addressing Mode

- The indirect addressing mode uses a register to hold the actual address that will finally be used in the data move; the register itself is not the address, but rather the number in the register. Indirect addressing for MOV opcodes uses register R0 or R1, often called "data pointers," to hold the address of one of the data locations, which could be a RAM or an SFR address.

  MOV A,@RO ; Copy the contents of the address in RO to the A register

  MOV @R1, # 35h ; Copy the number 35h to the address in R1

  MOV add,@R0 ; Copy the contents of the address in RO to add

  MOV @R1, A ; Copy the contents of A to the address in R1

  MOV @R0, 80h ; Copy the contents of the port 0 pins to the address in R0

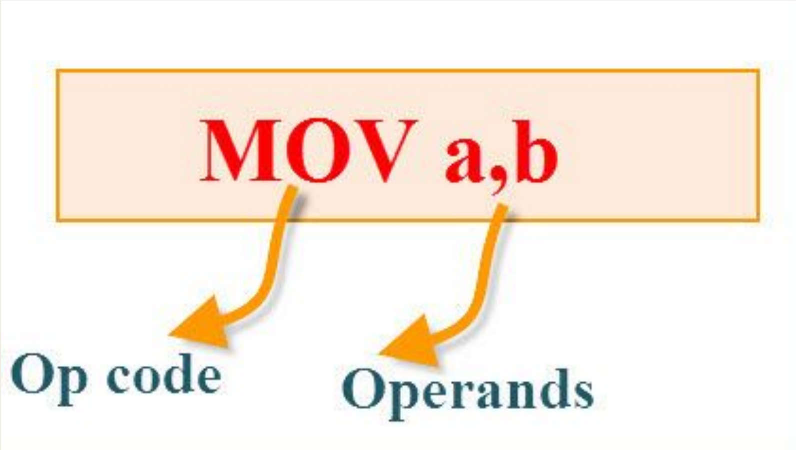
Indexed Addressing

- Indexed addressing uses a base register (either the program counter or the data pointer) and an offset (the accumulator) in forming the effective address for a JMP or MOVC instruction. Jump tables or look-up tables are easily created using indexed addressing.
- MOVC A,@A+DPTR ; Copy the code byte, found at the ROM address formed by adding A and the DPTR, to A
- MOVC A,@A + PC ; Copy the code byte, found at the ROM address formed by adding A and the PC, to A

Rules of Assembly Language

- The assembly code must be written in upper case letters
- The labels must be followed by a colon (label:)
- All symbols and labels must begin with a letter
- All comments are typed in lower case
- The last line of the program must be the END directive

- The assembly language mnemonics are in the form of op-code, such as MOV, ADD, JMP, and so on, which are used to perform the operations.



- Op-code: The op-code is a single instruction that can be executed by the CPU. Here the op-code is a MOV instruction.
- Operands: The operands are a single piece of data that can be operated by the op-code. Example, multiplication operation is performed by the operands that are multiplied by the operand.
- Syntax: MUL a,b;

DATA TRANSFER INSTRUCTIONS

MOV

- **Function: Move Memory**
- Syntax: MOV operand1,operand2
- Description: MOV copies the value of operand2 into operand1. The value of operand2 is not affected. Both operand1 and operand2 must be in Internal RAM.

Instructions	OpCode	Bytes	Cycles	Flags
MOV @R0,#*data*	0x76	2	1	None
MOV @R1,#*data*	0x77	2	1	None
MOV @R0,A	0xF6	1	1	None
MOV @R1,A	0xF7	1	1	None

MOVX

- **Function: Move Data To/From External Memory (XRAM)**
- Syntax: MOVX operand1,operand2
- Description: 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 External Memory into the Accumulator.
- 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.

Instructions	OpCode	Bytes	Cycles	Flags
MOVX @DPTR,A	0xF0	1	2	None
MOVX @R0,A	0xF2	1	2	None
MOVX @R1,A	0xF3	1	2	None
MOVX A,@DPTR	0xE0	1	2	None
MOVX A,@R0	0xE2	1	2	None
MOVX A,@R1	0xE3	1	2	None

- **Operation: MOVC**
- Function: Move Code Byte to Accumulator
- Syntax: MOVC A,@A+register
- Description: 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.

Instructions	OpCode	Bytes	Cycles	Flags
MOVC A,@A+DPTR	0x93	1	2	None
MOVC A,@A+PC	0x83	1	1	None

PUSH

- **Function: Push Value Onto Stack**
- Syntax: PUSH
- Description: PUSH "pushes" the value of the specified iram addr onto the stack. PUSH first increments the value of the Stack Pointer by 1, then takes the value stored in iram addr and stores it in Internal RAM at the location pointed to by the incremented Stack Pointer.

Instructions	OpCode	Bytes	Cycles	Flags
PUSH *iram addr*	0xC0	2	2	None

POP

- **Function:** **Pop Value From Stack**
- Syntax: POP
- Description: POP "pops" the last value placed on the stack into the iram addr specified. In other words, POP will load iram addr with the value of the Internal RAM address pointed to by the current Stack Pointer. The stack pointer is then decremented by 1.

Instructions	OpCode	Bytes	Cycles	Flags
POP *iram addr*	0xD0	2	2	None

XCH

- **Function: Exchange Bytes**
- Syntax: XCH A,register
- Description: Exchanges the value of the Accumulator with the value contained in register.

Instructions	OpCode	Bytes	Cycles	Flags
XCH A,@R0	0xC6	1	1	None
XCH A,@R1	0xC7	1	1	None
XCH A,R0	0xC8	1	1	None
XCH A,R1	0xC9	1	1	None
XCH A,R2	0xCA	1	1	None
XCH A,R3	0xCB	1	1	None

SWAP

- **Function: Swap Accumulator Nibbles**
- Syntax: SWAP A
- Description: 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.

Instructions	OpCode	Bytes	Cycles	Flags
SWAP A	0xC4	1	1	None

ARITHMETIC INSTRUCTIONS

ADD/ ADDC - Add Accumulator/Add Accumulator with Carry

- Function: Add Accumulator, Add Accumulator With Carry
- Syntax: ADD A,operand

  ADDC A,operand
- Description: 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. The Carry bit (C) is set if there is a carry-out of bit 7. The Auxiliary Carry (AC) bit is set if there is a carry-out of bit 3. The Overflow (OV) bit is set if there is a carry-out of bit 6 or out of bit 7, but not both.

Instructions	OpCode	Bytes	Cycles	Flags
ADD A,#data	0x24	2	1	C, AC, OV
ADD A,iram addr	0x25	2	1	C, AC, OV
ADD A,@R0	0x26	1	1	C, AC, OV
ADD A,@R1	0x27	1	1	C, AC, OV
ADD A,R0	0x28	1	1	C, AC, OV
ADD A,R1	0x29	1	1	C, AC, OV
ADD A,R2	0x2A	1	1	C, AC, OV

SUBB - Subtract from Accumulator with Borrow

- **Function: Subtract from Accumulator With Borrow**
- Syntax: SUBB A,operand
- **Description:** SUBB subtract the value of *operand* from the value of the Accumulator, leaving the resulting value 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. In other words, if the unsigned value being subtracted is greater than the Accumulator the Carry Flag is set.
- The **Auxillary Carry (AC)** bit is set if a borrow was required for bit 3, otherwise it is cleared. In other words, 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. In other words, the subtraction of two signed bytes resulted in a value outside the range of a signed byte (-128 to 127). Otherwise it is cleared.

Instructions	OpCode	Bytes	Cycles	Flags
SUBB A,@R0	0x96	1	1	C, AC, OV
SUBB A,@R1	0x97	1	1	C, AC, OV
SUBB A,R0	0x98	1	1	C, AC, OV
SUBB A,R1	0x99	1	1	C, AC, OV

MUL - Multiply Accumulator by B

- **Function: Multiply Accumulator by B**
- Syntax: MUL AB
- Description: 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.
- The Overflow Flag (OV) is set if the result is greater than 255 (if the most-significant byte is not zero), otherwise it is cleared.

Instructions	OpCode	Bytes	Cycles	Flags
MUL AB	0xA4	1	4	C, OV

DIV - Divide Accumulator by B

- **Function: Divide Accumulator by B**
- Syntax: DIV AB
- Description: 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.
- The Overflow flag (OV) is set if division by 0 was attempted, otherwise it is cleared.

Instructions	OpCode	Bytes	Cycles	Flags
DIV AB	0x84	1	1	C, OV

INC - Increment Register

- **Function: Increment Register**
- Syntax: INC register
- Description: 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. Again, the Carry Flag is NOT set when the value of DPTR "rolls over" from 65535 to 0.

Instructions	OpCode	Bytes	Cycles	Flags
INC A	0x04	1	1	None
INC *iram addr*	0x05	2	1	None
INC @R0	0x06	1	1	None
INC @R1	0x07	1	1	None
INC R0	0x08	1	1	None
INC R1	0x09	1	1	None
INC R2	0x0A	1	1	None

DEC - Decrement Register

- **Function: Decrement Register**
- Syntax: DEC register
- Description: 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). Note: The Carry Flag is NOT set when the value "rolls over" from 0 to 255

Instructions	OpCode	Bytes	Cycles	Flags
DEC A	0x14	1	1	None
DEC *iram addr*	0x15	2	1	None
DEC @R0	0x16	1	1	None
DEC @R1	0x17	1	1	None
DEC R0	0x18	1	1	None
DEC R1	0x19	1	1	None
DEC R2	0x1A	1	1	None

LOGICAL INSTRUCTIONS

ANL - Bitwise AND

- **Function: Bitwise AND**
- Syntax: ANL operand1, operand2
- Description: ANL does a bitwise "AND" operation between operand1 and operand2, leaving the resulting value in operand1. The value of operand2 is not affected. A logical "AND" compares the bits of each operand and sets the corresponding bit in the resulting byte only if the bit was set in both of the original operands, otherwise the resulting bit is cleared.

Instructions	OpCode	Bytes	Cycles	Flags
ANL iram addr,A	0x52	2	1	None
ANL iram addr,#data	0x53	3	2	None
ANL A,#data	0x54	2	1	None
ANL A,iram addr	0x55	2	1	None
ANL A,@R0	0x56	1	1	None
ANL A,@R1	0x57	1	1	None
ANL A,R0	0x58	1	1	None
ANL A,R1	0x59	1	1	None
ANL A,R2	0x5A	1	1	None

ORL - Bitwise OR

- **Function: Bitwise OR**
- Syntax: ORL operand1,operand2
- Description: ORL does a bitwise "OR" operation between operand1 and operand2, leaving the resulting value in operand1. The value of operand2 is not affected. A logical "OR" compares the bits of each operand and sets the corresponding bit in the resulting byte if the bit was set in either of the original operands, otherwise the resulting bit is cleared.

Instructions	OpCode	Bytes	Cycles	Flags
ORL *iram addr*,A	0x42	2	1	None
ORL *iram addr*,#*data*	0x43	3	2	None
ORL A,#*data*	0x44	2	1	None
ORL A,*iram addr*	0x45	2	1	None
ORL A,@R0	0x46	1	1	None
ORL A,@R1	0x47	1	1	None
ORL A,R0	0x48	1	1	None
ORL A,R1	0x49	1	1	None

XRL - Bitwise Exclusive OR

- **Function: Bitwise Exclusive OR**
- Syntax: XRL operand1,operand2
- Description: XRL does a bitwise "EXCLUSIVE OR" operation between operand1 and operand2, leaving the resulting value in operand1. The value of operand2 is not affected. A logical "EXCLUSIVE OR" compares the bits of each operand and sets the corresponding bit in the resulting byte if the bit was set in either (but not both) of the original operands, otherwise the bit is cleared.

Instructions	OpCode	Bytes	Cycles	Flags
XRL *iram addr*,A	0x62	2	1	None
XRL *iram addr*,#*data*	0x63	3	2	None
XRL A,#*data*	0x64	2	1	None
XRL A,*iram addr*	0x65	2	1	None
XRL A,@R0	0x66	1	1	None
XRL A,@R1	0x67	1	1	None
XRL A,R0	0x68	1	1	None
XRL A,R1	0x69	1	1	None
XRL A,R2	0x6A	1	1	None

CPL - Complement Register

- Function: Complement Register
- Syntax: CPL operand
- Description: CPL complements operand, leaving the result in operand. If operand is a single bit then the state of the bit will be reversed. If operand is the Accumulator then all the bits in the Accumulator will be reversed. This can be thought of as "Accumulator Logical Exclusive OR 255" or as "255-Accumulator." 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.

Instructions	OpCode	Bytes	Cycles	Flags
CPL A	0xF4	1	1	None
CPL C	0xB3	1	1	C
CPL *bit addr*	0xB2	2	1	None

ROTATE AND SWAP INSTRUCTIONS

RL - Rotate Accumulator Left

- Function: Rotate Accumulator Left
- Syntax: RL A
- Description: Shifts the bits of the Accumulator to the left. The left-most bit (bit 7) of the Accumulator is loaded into bit 0.

Instructions	OpCode	Bytes	Cycles	Flags
RL A	0x23	1	1	C

RLC - Rotate Accumulator Left through Carry

- Function: Rotate Accumulator Left Through Carry
- Syntax: RLC A
- Description: Shifts the bits of the Accumulator to the left. The left-most bit (bit 7) of the Accumulator is loaded into the Carry Flag, and the original Carry Flag is loaded into bit 0 of the Accumulator. This function can be used to quickly multiply a byte by 2.

Instructions	OpCode	Bytes	Cycles	Flags
RLC A	0x33	1	1	C

RR - Rotate Accumulator Right

- Function: Rotate Accumulator Right
- Syntax: RR A
- Description: Shifts the bits of the Accumulator to the right. The right-most bit (bit 0) of the Accumulator is loaded into bit 7.

Instructions	OpCode	Bytes	Cycles	Flags
RR A	0x03	1	1	None

RRC - Rotate Accumulator Right through Carry

- Function: Rotate Accumulator Right Through Carry
- Syntax: RRC A
- Description: Shifts the bits of the Accumulator to the right. The right-most bit (bit 0) of the Accumulator is loaded into the Carry Flag, and the original Carry Flag is loaded into bit 7. This function can be used to quickly divide a byte by 2.

Instructions	OpCode	Bytes	Cycles	Flags
RRC A	0x13	1	1	C

JMP - Jump to Address

- **Function: Jump to Data Pointer + Accumulator**
- Syntax: JMP @A+DPTR
- Description: JMP jumps unconditionally to the address represented by the sum of the value of DPTR and the value of the Accumulator.

Instructions	OpCode	Bytes	Cycles	Flags
JMP @A+DPTR	0x73	1	2	None

8051 Timer Programming

- **Mode 1 programming**

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer; therefore, it allows values of 0000 to FFFFH to be loaded into the timer's registers TL and TH.

2. After TH and TL are loaded with a 16-bit initial value, the timer must be started. This is done by the instructions "SETB TRO" for Timer 0 and "SETB TR1″ for Timer 1.

3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH. When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer with the instructions "CLR TRO" or "CLR TR1″, for Timer 0 and Timer 1, respectively. Again, it must be noted that each timer has its own timer flag: TFO for Timer 0, and TF1 for Timer 1.

4. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value, and TF must be reset to 0.

Steps to program in mode 1

To generate a time delay, using the timer's mode 1, the following steps are taken.

1. Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used and which timer mode (0 or 1) is selected.
2. Load registers TL and TH with initial count values.
3. Start the timer.
4. Keep monitoring the timer flag (TF) with the "JNB TFx, target" instructtion to see if it is raised. Get out of the loop when TF becomes high.
5. Stop the timer.
6. Clear the TF flag for the next round.
7. Go back to Step 2 to load TH and TL again.

In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the PI.5 bit. Timer 0 is used to generate the time delay. Analyze the program.

```
            MOV    TMOD,#01        ;Timer 0, mode 1(16-bit mode)
HERE:       MOV    TL0,#0F2H       ;TL0 = F2H, the Low byte
            MOV    TH0,#0FFH       ;TH0 = FFH, the High byte
            CPL    P1.5            ;toggle P1.5
            ACALL DELAY
            SJMP   HERE            ;load TH, TL again
;————————delay using Timer 0
DELAY:
            SETB   TR0             ;start Timer 0
AGAIN:      JNB    TF0,AGAIN       ;monitor Timer 0 flag until
                                   ;it rolls over
            CLR    TR0             ;stop Timer 0
            CLR    TF0             ;clear Timer 0 flag
            RET
```

Solution:

- In the above program notice the following steps.
- TMOD is loaded.
- FFF2H is loaded into THO – TLO.
- P1.5 is toggled for the high and low portions of the pulse.
- The DELAY subroutine using the timer is called.
- In the DELAY subroutine, Timer 0 is started by the “SETB TRO” instruction.
- Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH.
- One more clock rolls it to 0, raising the timer flag (TFO = 1). At that point, the JNB instruction falls through.
- Timer 0 is stopped by the instruction “CLR TRO”. The DELAY subroutine ends, and the process is repeated.

Serial Port Programming

- The 8051 supports a full duplex serial port.
- Three special function registers support serial communication.

1. **SBUF Register**: Serial Buffer (SBUF) register is an 8-bit register. It has separate SBUF registers for data transmission and for data reception. For a byte of data to be transferred via the TXD line, it must be placed in SBUF register. Similarly, SBUF holds the 8-bit data received by the RXD pin and read to accept the received data.

2. **SCON register:** The contents of the Serial Control (SCON) register are shown below. This register contains mode selection bits, serial port interrupt bit (TI and RI) and also the ninth data bit for transmission and reception (TB8 and RB8).

3. **PCON register**: The SMOD bit (bit 7) of PCON register controls the baud rate in asynchronous mode transmission.

Serial Port Control (SCON) Register							
D7	D6	D5	D4	D3	D2	D1	D0
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

- SM0 (SCON.7) : Serial communication mode selection bit
- SM1 (SCON.6) : Serial communication mode selection bit

SM0	SM1	Mode	Description	Baud rate
0	0	Mode 0	8-bit shift register mode	Fosc / 12
0	1	Mode 1	8-bit UART	Variable (set by timer 1)
1	0	Mode 2	9-bit UART	Fosc/ 32 or Fosc/64
1	1	Mode 3	9-bit UART	Variable (set by timer 1)

- SM2 (SCON.5) : Multiprocessor communication bit. In modes 2 and 3, if set this will enable multiprocessor communication.
- REN (SCON.4) : Enable serial reception
- TB8 (SCON.3) : This is 9th bit that is transmitted in mode 2 & 3.
- RB8 (SCON.2) : 9th data bit is received in modes 2 & 3.
- TI (SCON.1) : Transmit interrupt flag, set by hardware must be cleared by software.
- RI (SCON.0) : Receive interrupt flag, set by hardware must be cleared by software.

Power mode Control (PCON) Register							
D7	D6	D5	D4	D3	D2	D1	D0
SMOD	**--**	**--**	**--**	**GF1**	**GF0**	**PD**	**IDL**

- SMD (PCON.7) : Serial rate modify bit. Set to 1 by program to double baud rate using timer 1 for modes 1, 2, and 3. cleared by program to use timer 1 baud rate.

- GF1 (PCON.3) : General Purpose user flag bit.
- GF0 (PCON.2) : General Purpose user flag bit.
- PD (PCON.1) : Power down bit. Set to 1 by program to enter power down configuration for CHMOS processors.
- IDL (PCON.0) : Idle mode bit. Set to 1 by program to enter idle mode configuration for CHMOS processors.

SERIAL COMMUNICATION MODES

- **Mode 0**

  In this mode serial port runs in synchronous mode. The data is transmitted and received through RXD pin and TXD is used for clock output. In this mode the baud rate is 1/12 of clock frequency.

- **Mode 1**

  In this mode SBUF becomes a 10 bit full duplex transceiver. The ten bits are 1 start bit, 8 data bit and 1 stop bit. The interrupt flag TI/RI will be set once transmission or reception is over. In this mode the baud rate is variable and is determined by the timer 1 overflow rate.

  Baud rate = [2smod/32] x Timer 1 overflow Rate

  = [2smod/32] x [Oscillator Clock Frequency] / [12 x [256 – [TH1]]]

- **Mode 2**

  This is similar to mode 1 except 11 bits are transmitted or received. The 11 bits are, 1 start bit, 8 data bit, a programmable 9th data bit, 1 stop bit.

  Baud rate = [2smod/64] x Oscillator Clock Frequency

- **Mode 3**

  This is similar to mode 2 except baud rate is calculated as in mode 1

The 8051

10/27/08

8051

- Today over fifty companies produce variations of the 8051.
- Several of these companies have over fifty versions of the 8051.
- 8051 cores are available for implementations in FPGA’s or ASIC’s.
- Over 100 million 8051’s are sold each year.
- The 8051 has been extremely successful, and has directly influenced many of the more recent microcontroller architectures.

8051 software

- Download the Silicon Labs software form the class web page. Don’t use the new version on the Silicon Labs web page.
- Download my tutorial on using the Silicon Labs University Daughter Card and go through both the C and assembly language tutorials.
- See handout on 8051 instruction set.
- The MCS 51 Family User’s Manual is available on the class web page.
  – Look under Resources - Other

MCS-51

- MCS-51 is Intel's designation for its family of 8051 devices.
- The 8051 is the original member of the MCS-51 family, and is the core for all MCS-51 devices.
- The original 8051 was available in three versions.
  - 8051 – A fixed program in read only memory (ROM) version.
  - 8031 – No internal ROM program stored in external programmable read only memory (PROM) memory.
  - 8751 – Program stored in internal erasable PROM (EPROM). Erased by exposing the chip to high intensity ultraviolet light for several minutes. Eventually EPROM was replaced by EEPROM.

The basic 8051 Core

- 8-bit CPU optimized for control applications
- Capability for single bit Boolean operations.
- Supports up to 64K of program memory.
- Supports up to 64K of program memory.
- 4 K bytes of on-chip program memory.
    - Newer devices provide more.
- 128 or 256 bytes of on-chip data RAM
- Four 8 bit ports.
- Two 16-bit timer/counters
- UART
- Interrupts
- On-chip clock oscillator

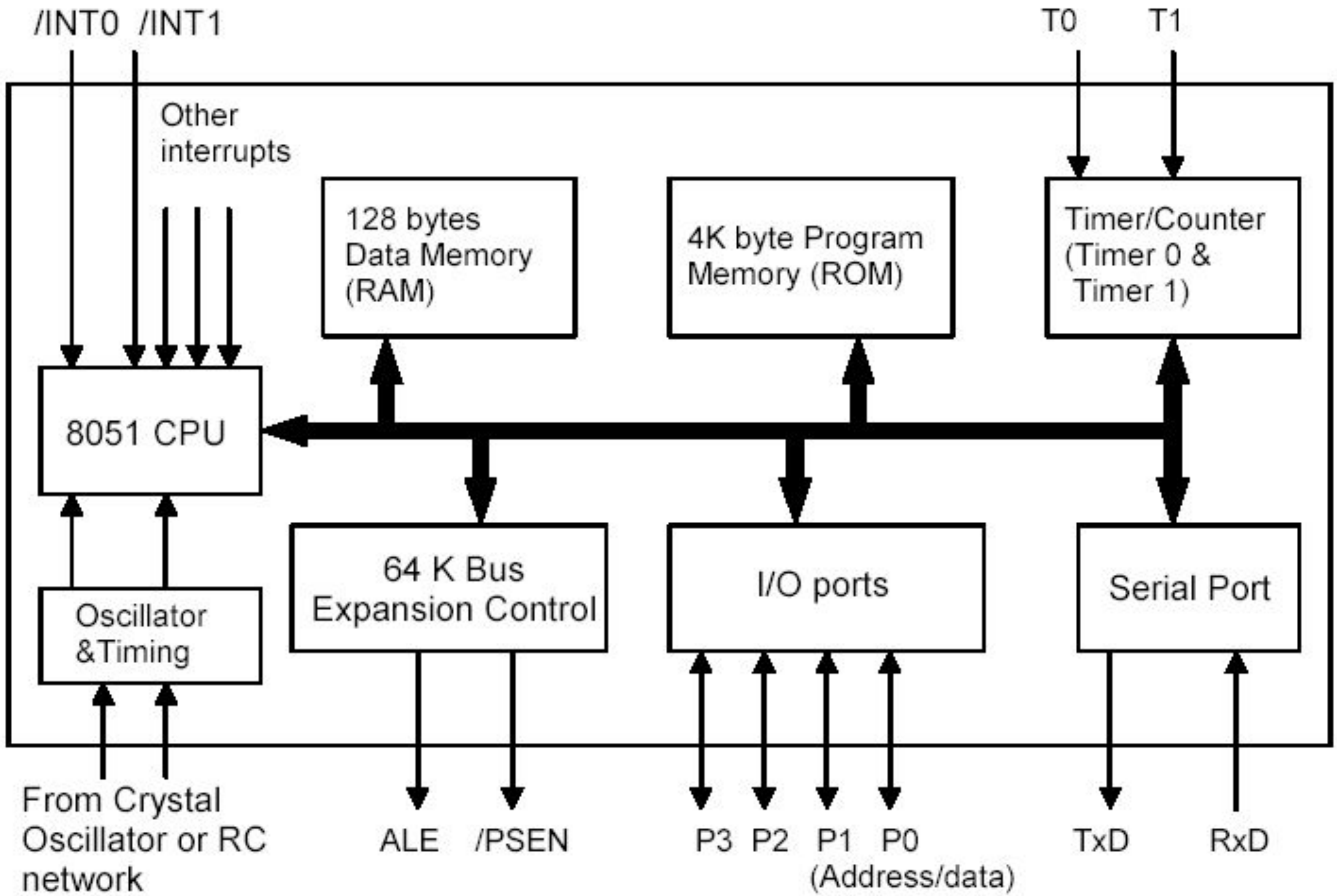
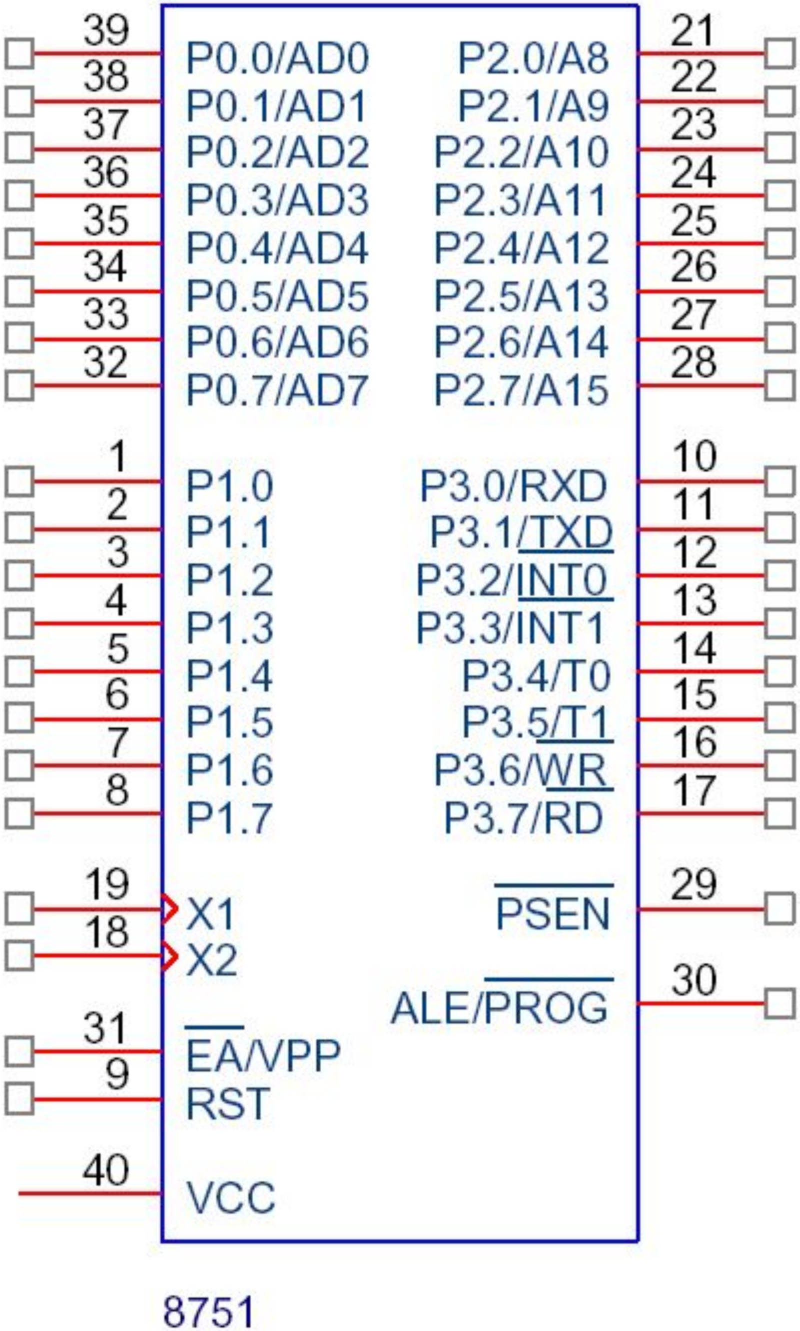


Figure 1.1 Block Diagram of the generic 8051 Microcontroller



The 8751 in a 40-pin DIP
To use internal EPROM /EA (pin 31) is connected to 5 volts. To use external program memory /EA is connected to ground.

MEMORY

CODE MEMORY Internal ROM selected by $\overline{\text{EA}} = 1$

CM = CM(0..0FFFFH) = CM(0..0FFFFH;7..0)[1]

ON CHIP DATA MEMORY

DM = DM(0..7FH) = DM(0..7FH;7..0)

ON CHIP BIT ADDRESSIABLE MEMORY

BADM = BADM(0..0FF) = BADM(0..0FF;0) ;BIT ADDR DATA MEMORY

EXTERNAL RAM MEMORY

XM = XM(0..0FFFFH) = XM(0..0FFFFH;7..0)

INTERNAL REGISTERS AND PORTS

PC = PC(15..0)

SP = SP(7..0)

DPTR = DPTR(15..0)

PSW = CY|AC|F0|RS1|RS0|OV|P

TCON = TF1|TR1|TF0|TR0|IE1|IT1|IE0|IT0

Memory Organization

- The 8051 memory organization is rather complex.
- The 8051 has separate address spaces for Program Memory, Data Memory, and external RAM.
- This is refereed to as a Harvard architecture.
  - The early Mark I (1944) computer developed at Harvard was of this type of architecture.
  - Von Neumann at Princeton pointed out that it was not necessary to put instructions and data in separate memories.
  - Most machines have been Princeton architecture.
  - Recently Harvard architecture has been employed to help alleviate the memory bottleneck.
- Both program memory and external data memory are 8 bits wide and use 16 bits of address. The internal data memory is accessed using an 8-bit address.
- Since the same address can refer to different locations the specific location is determined by the type of instruction.

Program or Code Memory

- May consist of internal or external program memory. The amount of internal program memory varies depending on the device.
  - 4K bytes typical in older devices.
  - The Silicon Labs C8051F310 contains 16K of flash memory for programs.
  - The Silicon Labs C8051F020 which is on the University Daughter Card (UDC) contains 4K bytes of program memory.
- The MOVC instruction can be use to read code memory.
- To reference code memory I will use the notation:

CM = CM(0,…,FFFFH) = CM(0,…,FFFFH; 7,…,0)

- This notation can be used to specify particular bits and bytes of code memory.

For example CM(1234H) refers to the byte of code memory at address 1234H. CM(1234H;7) refers to the most significant bit in that address.

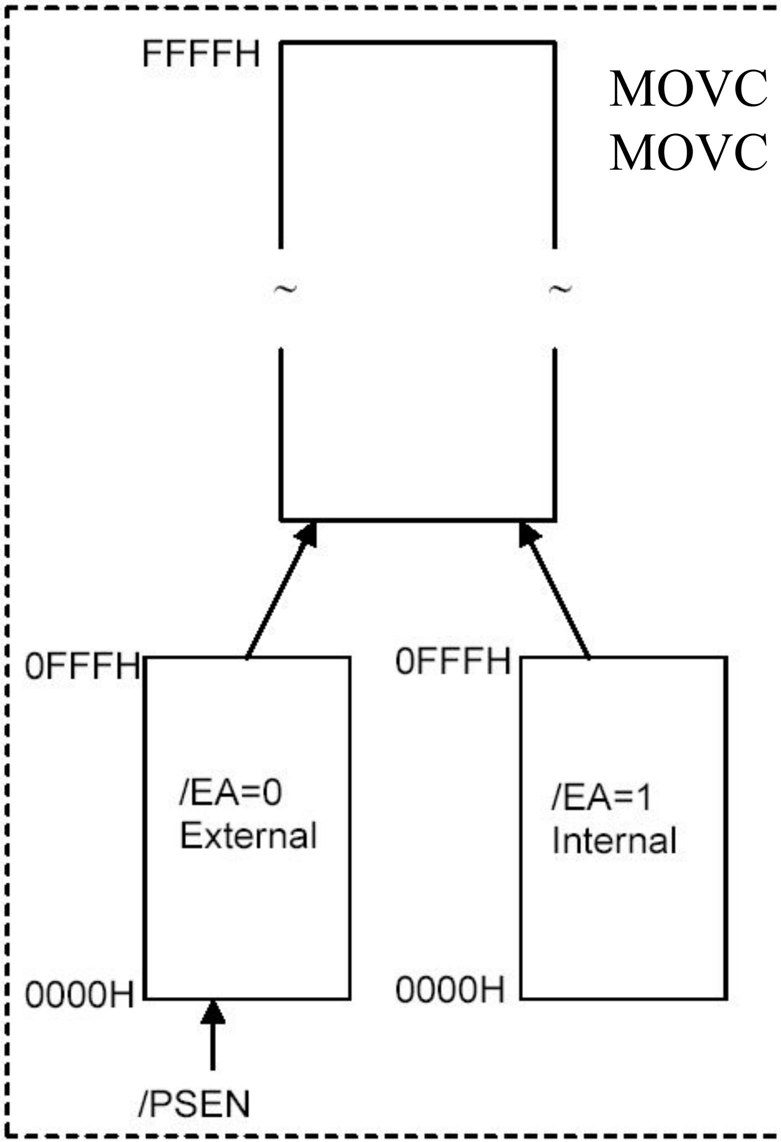


Figure 1.4 Program Memory Organization (Read Only)

MOVC A,@A + DPTR ;A ← CM(A+DPTR)
MOVC A,@A + PC ;A ← CM(A+PC)

CM

PC = PC(15..0)

DPTR = DPTR(15..0)

External Memory

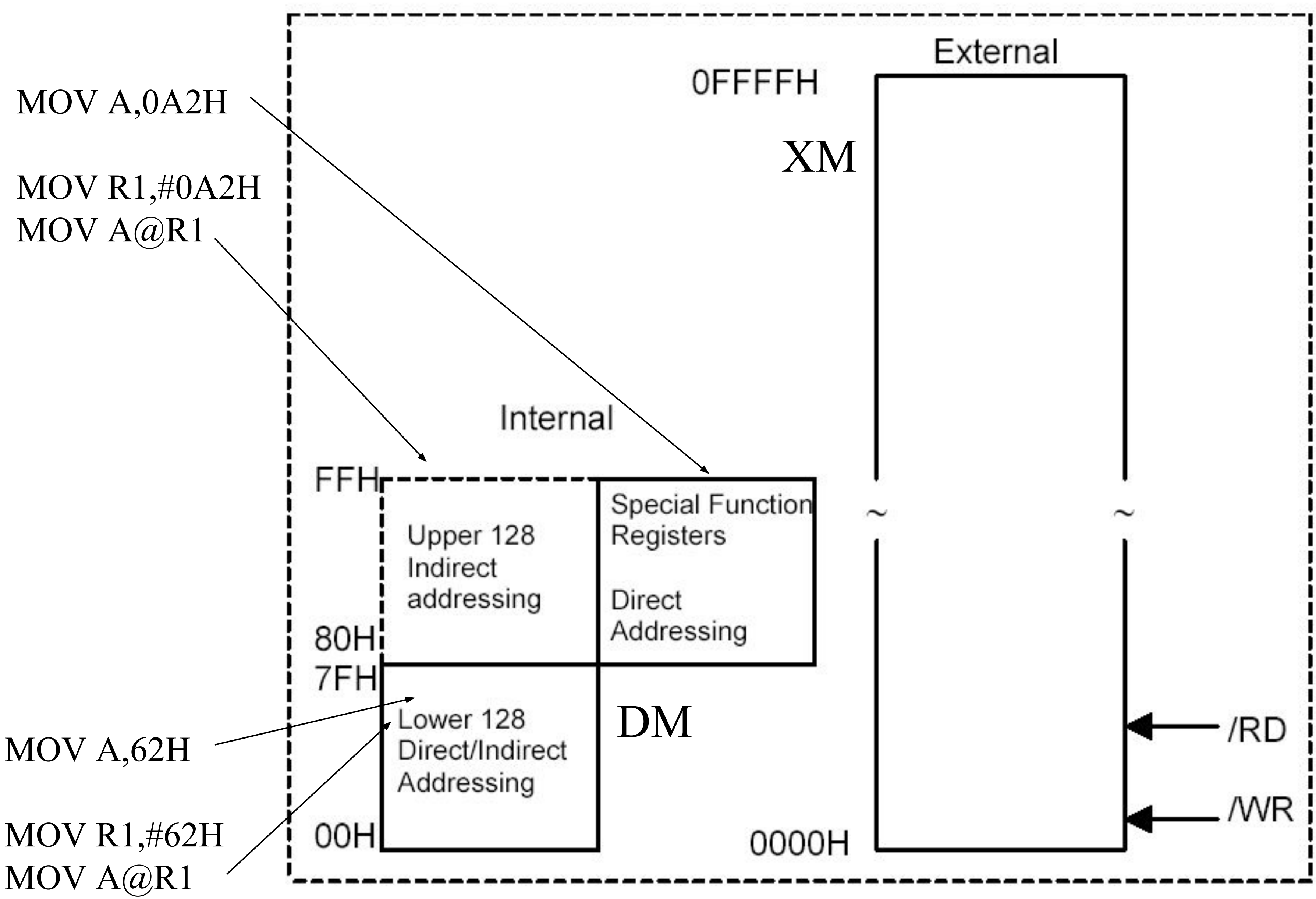
- Supports up to 64K bytes external memory.
  - XM(0000,…,FFFF)

    = XM(0000,…,FFFF; 7,…,0)
  - Accessed by using the MOVX instruction.
- On the original using external memory reduces number of available I/O ports.
- On some new devices this is not the case.
  - For example in C8051F020 64K bytes of external memory has been included in the chip.
  - The 4 standard 8051 ports are available and three additional ports have been added.

```
MOVX  A,@DPTR    ;A ← XM(DPTR)
MOVX  A,@Rn      ;A ← XM(P2|Rn)
MOVX  @DPTR,A    ;XM(DPTR) ← A
MOVX  @Rn,A      ;XM(P2|Rn) ← A
```

Data Memory

- The original 8051 had 128 bytes of on-chip data RAM.
  - This memory includes 4 banks of general purpose registers at DM(00..1F)
  - Only one bank can be active at a time.
  - If all four banks are used, DM(20..7F) is available for program data.
  - DM(20..2F) is bit addressable as BADM(00..7F).
- DM(80,…,FF) contains the special function registers such as I/O ports, timers, UART, etc.
  - Some of these are bit addressable using BADM(80..FF)
- On newer versions of the 8051, DM(80,…,FF) is also use as data memory. Thus, the special functions registers and data memory occupy the same address space. Which is accessed is determined by the instruction being used.


MOV A,0A2H
MOV R1,#0A2H
MOV A@R1
MOV A,62H
MOV R1,#62H
MOV A@R1
External
0FFFFH
XM
Internal
FFH
80H
7FH
00H
Upper 128
Indirect
addressing
Special Function
Registers
Direct
Addressing
Lower 128
Direct/Indirect
Addressing
DM
/RD
/WR
0000H

Data memory

Byte Address	Bit Address							
7F	General Purpose RAM							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F	Bank 3							
18								
17	Bank 2							
10								
0F	Bank 1							
08								
07	Default Register Bank for R0 – R7							
00								

Bit Addressable

Byte Address	Bit Address								
FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	-	D0	PSW
B8	-	-	-	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	-	-	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	Not bit-addressable								SBUF
98	9F	96	95	94	93	92	91	90	SCON
90	97	96	95	94	93	92	91	90	P1
8D	Not bit-addressable								TH1
8C	Not bit-addressable								TH0
8B	Not bit-addressable								TL1
8A	Not bit-addressable								TL0
89	Not bit-addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	Not bit-addressable								PCON
83	Not bit-addressable								DPH
82	Not bit-addressable								DPL
81	Not bit-addressable								SP
80	87	86	85	84	83	82	81	80	P0

Data Memory (DM)

Table 1

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
*+T2CON	Timer/Counter 2 Control	0C8H
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 Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

* = Bit addressable
+ = 8052 only

SFR MEMORY MAP

8 Bytes

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

↑
Bit
Addressable

Figure 5

PSW: PROGRAM STATUS WORD. BIT ADDRESSABLE.

CY	AC	F0	RS1	RS0	OV	—	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

NOTE:

1. The value presented by RS0 and RS1 selects the corresponding register bank.

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

INTERNAL REGISTERS AND PORTS

PC = PC(15..0)

SP = SP(7..0)

DPTR = DPTR(15..0)

PSW = CY|AC|F0|RS1|RS0|OV|P

TCON = TF1|TR1|TF0|TR0|IE1|IT1|IE0|IT0

BADM(87H..80H) = **P0** = P0(7..0) = BADM(87H..80H)

BADM(8FH..88H) = **TCON** =BADM(8FH..88H)

BADM(97H..90H) = **P1** = P1(7..0) =BADM(97H..90H)

BADM(9FH..98H) = **SCON** =BADM(9FH..98H)

BADM(A7H..A0H) = **P2** = P2(7..0) =BADM(A7H..A0H)

BADM(AFH..A8H) = **IE** =BADM(AFH..A8H)

BADM(B7H..B0H) = **P3** = P3(7..0) =BADM(B7H..B0H)

BADM(BFH..B8H) = **IP** =BADM(BFH..B8H)

BADM(C7H..C0H) = BADM(C7H..C0H)

BADM(CFH..C8H) = BADM(CFH..C8H)

BADM(D7H..D0H) = **PSW** =BADM(D7H..D0H)

BADM(DFH..D8H) = BADM(DFH..D8H)

BADM(E7H..E0H) = **ACC** = **A** = A(7..0) = BADM(E7H..E0H)

BADM(EFH..E8H) = BADM(EFH..E8H)

BADM(F7H..F0H) = **B** =BADM(F7H..F0H)

BADM(FFH..F8H) = BADM(FFH..F8H)

RESET

PC $\leftarrow$ 0, A $\leftarrow$ 0, B $\leftarrow$ 0, PSW $\leftarrow$ 0, SP $\leftarrow$ 7H, SPTR $\leftarrow$ 0, P0-P3 $\leftarrow$ 0FFH,
IP $\leftarrow$ XXX00000B, IE $\leftarrow$ 0XX00000B, TMOD $\leftarrow$ 0, TCON $\leftarrow$ 0, TH0 $\leftarrow$ 0, TL0 $\leftarrow$ 0,
TH1 $\leftarrow$ 0, TL1 $\leftarrow$ 0, SCON $\leftarrow$ 0, PCON $\leftarrow$ 0XXXXXXXB, DPTR $\leftarrow$ 0000H

INTERNAL DATA MEMORY

00H-07H	RB0	Register Bank 0
08H-0FH	RB1	Register Bank 1
10H-17H	RB2	Register Bank 2
18H-1FH	RB3	Register Bank 3
20H-27H	BAM(00H)-BAM(3FH)	Bit addressable memory
28H-2FH	BAM(40H)-BAM(7FH)	Bit addressable memory
30H-37H		Available to user.
38H-3FH		Available to user.
40H-47H		Available to user.
48H-4FH		Available to user.
50H-57H		Available to user.
58H-5FH		Available to user.
60H-67H		Available to user.
68H-6FH		Available to user.
70H-77H		Available to user.
78H-7FH		Available to user.
80H	P0	

78H-7FH		Available to user.
80H	P0	
81H	SP	
82H	DPL	DPTR
83H	DPH	
84H		
85H		
86H		
87H	PCON	
88H		
89H	TMOD	Timer/Counter Mode Control
8AH	TL0	
8BH	TL1	
8CH	TH0	
8DH	TH1	
98H	SCON	Serial Port control
99H	SBUF	

NOTATION DEFINITIONS

$n \in \{0,1\}$, $i \in \{0,...,7\}$, $bit \in \{0,1\}$, $byte \in \{0,...,255\}$, $dbyte \in \{0,...,255\}$

$short \in \{0,...,03FFH\}$, $addr \in \{0,...,0FFFFH\}$

$Rn \in \{R0, R1\}$, $Ri \in \{R0, R1, ..., R7\}$

$Ri = DM(i+8*RBANK)$

$RBANK = RS1*2+RS0$

	CY	AC	FO	RS1	RS0	OV		P	
D0H	D7	D6	D5	D4	D3	D2	D1	D0	**PSW**

ARITHMETIC INSTRUCTIONS

ADD	A,#byte	;A $\leftarrow$ A + byte
ADD	A,@Rn	;A $\leftarrow$ A + DM(Rn)
ADD	A,Ri	;A $\leftarrow$ A + Ri
ADD	A,byte	;A $\leftarrow$ A + DM(byte)
ADDC	A,#byte	;A $\leftarrow$ A + byte + CY
ADDC	A,@Rn	;A $\leftarrow$ A + DM(Rn) + CY
ADDC	A,Ri	;A $\leftarrow$ A + Ri + CY
ADDC	A,byte	;A $\leftarrow$ A + DM(byte) + CY

Examples:

ADD A,#7FH ;A □ 7FH

ADD A,7FH ;A □ DM(7FH)

LOGICAL INSTRUCTIONS

ANL	A,#byte	;$A \leftarrow A \wedge byte$
ANL	A,@Rn	;$A \leftarrow A \wedge DM(Rn)$
ANL	A,Ri	;$A \leftarrow A \wedge Ri$
ANL	A,byte	;$A \leftarrow A \wedge DM(byte)$
ANL	dbyte,#byte	;$DM(dbyte) \leftarrow DM(dbyte) \wedge byte$
ANL	byte,A	;$DM(byte) \leftarrow A \wedge DM(byte)$
CLR	A	;$A \leftarrow 0$
CPL	A	;$A \leftarrow \overline{A}$
RL	A	;$A \leftarrow A(6..0)\|A(7)$
RLC	A	;$CY\|A \leftarrow A\|CY$
RR	A	;$A \leftarrow A(0)\|A(7..1)$
RRC	A	;$A\|CY \leftarrow CY\|A$
SWAP	A	;$A \leftarrow A(3..0)\|A(7..4)$

DATA MOVE INSTRUCTIONS

MOV	A,#byte	;A ← byte
MOV	A,@Rn	;A ← DM(Rn)
MOV	A,Ri	;A ← Ri
MOV	A,byte	;A ← DM(byte)
MOV	@Rn,A	;DM(Rn) ← A
MOV	@Rn,#byte	;DM(Rn) ← byte
MOV	@Rn,byte	;DM(Rn) ← DM(byte)
MOV	Ri,A	;Ri ← A
MOV	Ri,#byte	;Ri ← byte
MOV	Ri,byte	;Ri ← DM(byte)
MOV	byte,A	;DM(byte) ← A

MOV	dbyte,#byte	;DM(dbyte) $\leftarrow$ byte
MOV	byte,@Rn	;DM(byte) $\leftarrow$ DM(Rn)
MOV	byte,Ri	;DM(byte) $\leftarrow$ Ri
MOV	dbyte,byte	;DM(dbyte) $\leftarrow$ DM(byte)
MOV	DPTR,#addr	;DPTR $\leftarrow$ addr
MOVC	A,@A + DPTR	;A $\leftarrow$ CM(A+DPTR)
MOVC	A,@A + PC	;A $\leftarrow$ CM(A+PC)
MOVX	A,@DPTR	;A $\leftarrow$ XM(DPTR)
MOVX	A,@Rn	;A $\leftarrow$ XM(P2|Rn)
MOVX	@DPTR,A	;XM(DPTR) $\leftarrow$ A
MOVX	@Rn,A	;XM(P2|Rn) $\leftarrow$ A

MOVE INSTRUCTIONS

```
  MOV     A,#-1              ;LOAD A WITH 0FFH
  MOV     A,@R1              ;LOAD A WITH DATA MEM POINTED TO BY R1
  MOV     A,R1               ;LOAD A WITH CONTENTS OF R1
  MOV     A,2*30H            ;LOAD A WITH CONTENTS OF DATA MEM 60H
  MOV     A,P2               ;LOAD A WITH CONTENTS OF PORT 2
  MOV     @R0,A              ;STORE A IN DATA MEM POINTED TO BY R0
  MOV     @R1,#0FFH          ;PUT ALL 1'S IN LOC POINTED TO BY R1
  MOV     @R0,7FH            ;READ DATA MEMORY 7FH AND STORE IN
                             ;DATA MEMORY POINTED TO BY R0.
  MOV     R7,A               ;STORE A IN REGISTER R7
  MOV     R6,#01010101B      ;PUT 55H IN R6
  MOV     R0,#50H            ;LOAD R0 WITH 50H = 80
  MOV     R7,#01010001B      ;LOAD R7 WITH 51H = 81
  MOV     R5,55              ;PUT CONTENTS OF LOCATION 55 IN R5
  MOV     55,A               ;PUT A IN DATA MEM LOC 55
  MOV     55,#0AAH           ;PUT AAH IN DM LOC 55 DECIMAL
  MOV     P1,#0FFH           ;SET P1 TO ALL 1'S
  MOV     P2,@R1             ;SET P2 TO DATA MEM POINTED TO BY R1
  MOV     P0,P3              ;READ PORT 3 AND OUTPUT IT TO P0
  MOV     0,P0               ;READ PORT 0 AND SAVE IN DATA MEM 0
  MOV     R2,P2              ;READ P2 AND SAVE IT IN R2
  MOV     DPTR,#A_TABLE      ;POINT DPTR TO LOOK UP TABLE
  MOV     A,#0AH             ;PUT HEX B = 1011 = 11 IN ACC
  MOVC    A,@A+DPTR          ;CONVERT IT TO ASCII (A = 'B')
  CLR     A                  ;LOAD A WITH 0
  MOVC    A,@A+PC            ;PUT OPCODE OF PC+0 (NEXT INST) IN A
```

Push, Pop, and exchange

PUSH	byte	;DM(SP+1) $\leftarrow$ DM(byte), SP $\leftarrow$ SP + 1
POP	byte	;DM(byte) $\leftarrow$ DM(SP), SP $\leftarrow$ SP - 1
XCH	A,Ri	;A $\leftrightarrow$ Ri
XCH	A,byte	;A $\leftrightarrow$ DM(byte)
XCH	A,@Rn	;A $\leftrightarrow$ DM(R_i)
XCHD	A,@Rn	;A(3..0) $\leftrightarrow$ DM(Rn;3..0)

PROGRAM AND MACHINE CONTROL

CALL	**Note: Assembler translates CALL to ACALL or LCALL**	
ACALL	short	;DM(SP+2)|DM(SP+1) $\leftarrow$ PC+2,
		;PC(10..0) $\leftarrow$ short
LCALL	addr	;DM(SP+2)|DM(SP+1) $\leftarrow$ PC+2,
		;PC(10..0) $\leftarrow$ addr
RET		;PC $\leftarrow$ DM(SP)|DM(SP-1), SP $\leftarrow$ SP-2
RETI		;PC $\leftarrow$ DM(SP)|DM(SP-1), SP $\leftarrow$ SP-2
		;Reenable equal or lower priority INT
JMP	**Note: JMP is translated to AJMP, LJMP, or SJMP.**	
AJMP	short	;PC(10..0) $\leftarrow$ short
LJMP	addr	;PC(15..0) $\leftarrow$ addr
SJMP	byte	;PC $\leftarrow$ PC + 2 + byte(7)..byte(7)|byte
JMP	@A+DPTR	;PC $\leftarrow$ DPTR + A
JZ	byte	;IF A = 0 THEN
		;PC $\leftarrow$ PC + 2 + byte(7)..byte(7)|byte
		;ELSE PC $\leftarrow$ PC+2
JNZ	byte	;IF A $\neq$ 0 THEN
		;PC $\leftarrow$ PC + 2 + byte(7)..byte(7)|byte
		;ELSE PC $\leftarrow$ PC+2

CJNE	A,dbyte,byte	;IF A $\neq$ DM(dbyte) THEN
		;PC $\leftarrow$ PC + 2 + byte(7)..byte(7)|byte
		;IF A < DM(dbyte) THEN CY $\leftarrow$ 1
		;ELSE CY $\leftarrow$ 0
CJNE	A,#dbyte,byte	;IF A $\neq$ dbyte THEN
		;PC $\leftarrow$ PC + 2 + byte(7)..byte(7)|byte
		;IF A < dbyte THEN CY $\leftarrow$ 1
		;ELSE CY $\leftarrow$ 0
CJNE	Rn,#dbyte,byte	;IF Rn $\neq$ dbyte THEN
		;PC $\leftarrow$ PC + 2 + byte(7)..byte(7)|byte
		;IF A < DM(dbyte) THEN CY $\leftarrow$ 1
		;ELSE CY $\leftarrow$ 0
CJNE	@Rn,#dbyte,byte	;IF DM(Rn) $\neq$ dbyte THEN
		;PC $\leftarrow$ PC + 2 + byte(7)..byte(7)|byte
		;IF A < dbyte THEN CY $\leftarrow$ 1
		;ELSE CY $\leftarrow$ 0

DJNZ	Rn,byte	;Rn ← Rn-1, IF (Rn-1) ≠ 0 THEN
		;PC ← PC + 2 + byte(7)..byte(7)|byte
		;ELSE PC ← PC + 2
DJNZ	dbyte,byte	;DM(dbyte) ← DM(dbyte)-1,
		;IF (DM(dbyte)-1) ≠ 0 THEN
		;PC ← PC + 3 + byte(7)..byte(7)|byte
		;ELSE PC ← PC + 3
NOP		;PC ← PC + 1

BIT MANIPULATION INSTRUCTIONS

CLR	C	;CY $\leftarrow$ 0
CLR	byte	;BADM(byte) $\leftarrow$ 0
SETB	C	;CY $\leftarrow$ 1
SETB	byte	;BADM(byte) $\leftarrow$ 1
CPL	C	;CY $\leftarrow \overline{CY}$
CPL	byte	;BADM(byte) $\leftarrow \overline{\text{BADM(byte)}}$
ANL	C,byte	;CY $\leftarrow$ CY $\wedge$ BADM(byte)
ANL	C,/byte	;CY $\leftarrow$ CY $\wedge \overline{\text{BADM(byte)}}$
ANL	byte,bit	;BADM(byte) $\leftarrow$ BADM(byte) $\wedge$ bit
ORL	C,byte	;CY $\leftarrow$ CY $\vee$ BADM(byte)
ORL	C,/byte	;CY $\leftarrow$ CY $\vee \overline{\text{BADM(byte)}}$
ORL	byte,bit	;BADM(byte) $\leftarrow$ BADM(byte) $\vee$ bit
MOV	C,byte	;CY $\leftarrow$ BADM(byte)
MOV	byte,C	;BADM(byte) $\leftarrow$ CY

BIT JUMP INSTRUCTIONS

```
JB      dbyte,byte    ;IF BADM(dbyte) = 1 THEN
                      ;PC ← PC + 3 + byte(7)..byte(7)|byte
                      ;ELSE PC ← PC+3
JNB     dbyte,byte    ;IF BADM(dbyte) = 0 THEN
                      ;PC ← PC + 3 + byte(7)..byte(7)|byte
                      ;ELSE PC ← PC+3
JBC     dbyte,byte    ;IF BADM(dbyte) = 1 THEN
                      ;   BADM(dbyte) ← 0
                      ;   PC ← PC + 3 + byte(7)..byte(7)|byt
                      ;ELSE PC ← PC+3
JC      byte          ;IF CY = 1 THEN
                      ;PC ← PC + 3 + byte(7)..byte(7)|byte
                      ;ELSE PC ← PC+3
JNC     byte          ;IF CY = 0 THEN
                      ;PC ← PC + 3 + byte(7)..byte(7)|byte
                      ;ELSE PC ← PC+3
```

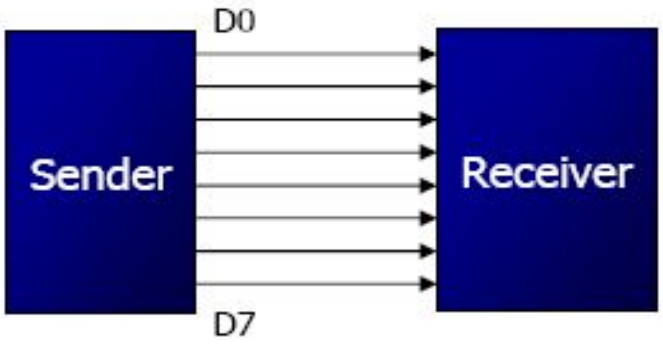
INSTRUCTIONS THAT AFFECT FLAGS (incomplete)

Instruction	CY	OV	AC	Instruction	CY	OV	AC
ADD	X	X	X	CLR C	0	-	-
ADDC	X	X	X	CPL C	X	-	-
SUBB	X	X	X	ANL C,bit	X	-	-
MUL	0	X	-	ANL C,/bit	X	-	-
DIV	0	X	-	ORL C,bit	X	-	-
DA	X	-	-	ORL C,bit	X	-	-
RRC	X	-	-	MOV C,bit	X	-	-
RLC	X	-	-	CJNE	X	-	-
SETB C	1	-	-				

- Computers transfer data in two ways:
  - Parallel
    - Often 8 or more lines (wire conductors) are used to transfer data to a device that is only a few feet away
  - Serial
    - To transfer to a device located many meters away, the serial method is used
    - The data is sent one bit at a time

Serial Transfer

Parallel Transfer

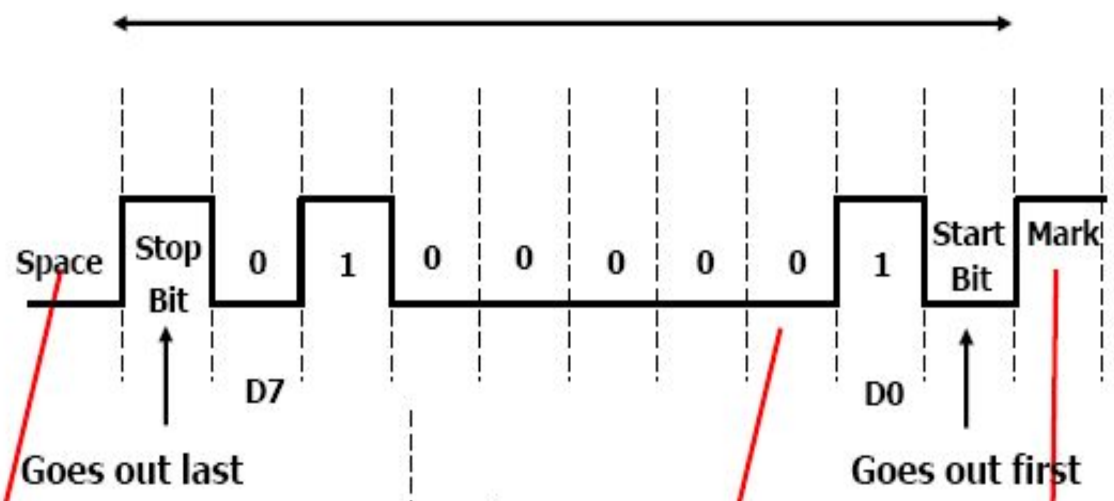


BASICS OF SERIAL COMMUNICA-TION

Start and Stop Bits (cont')

- The start bit is always a 0 (low) and the stop bit(s) is 1 (high)

ASCII character "A" (8-bit binary 0100 0001)



The 0 (low) is referred to as *space*

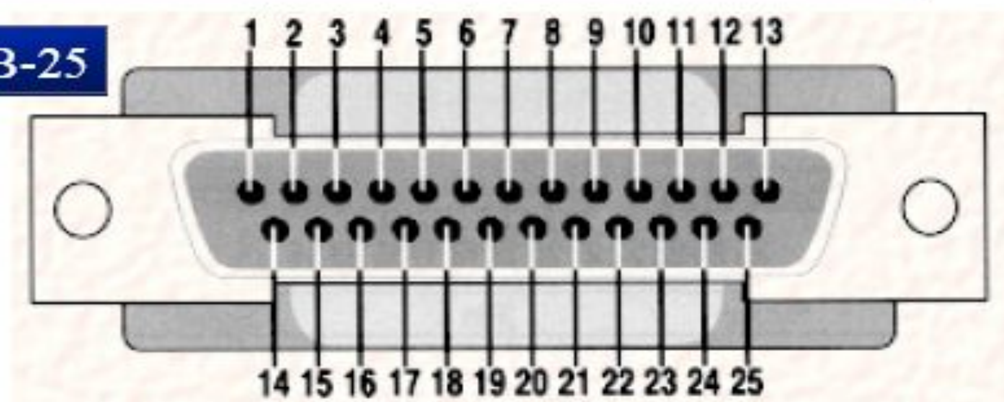
The transmission begins with a start bit followed by D0, the LSB, then the rest of the bits until MSB (D7), and finally, the one stop bit indicating the end of the character

When there is no transfer, the signal is 1 (high), which is referred to as *mark*

RS232 DB-25 Pins

Pin	Description	Pin	Description
1	Protective ground	14	Secondary transmitted data
2	Transmitted data (TxD)	15	Transmitted signal element timing
3	Received data (RxD)	16	Secondary receive data
4	Request to send (-RTS)	17	Receive signal element timing
5	Clear to send (-CTS)	18	Unassigned
6	Data set ready (-DSR)	19	Secondary receive data
7	Signal ground (GND)	20	Data terminal ready (-DTR)
8	Data carrier detect (-DCD)	21	Signal quality detector
9/10	Reserved for data testing	22	Ring indicator (RI)
11	Unassigned	23	Data signal rate select
12	Secondary data carrier detect	24	Transmit signal element timing
13	Secondary clear to send	25	Unassigned

RS232 Connector DB-25



SERIAL COMMUNICATION PROGRAMMING (cont')

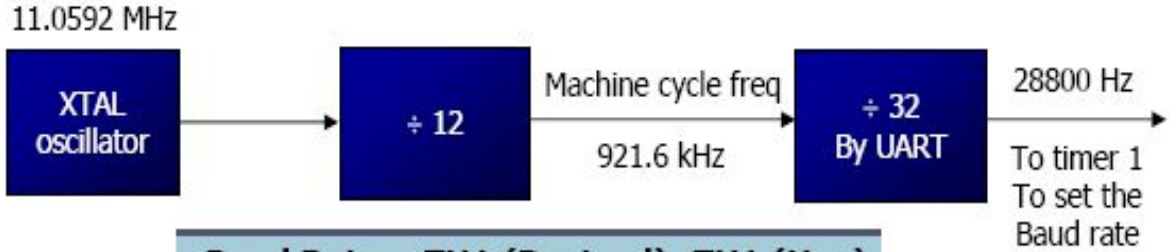
With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates. (a) 9600 (b) 2400 (c) 1200

Solution:

The machine cycle frequency of 8051 = 11.0592 / 12 = 921.6 kHz, and 921.6 kHz / 32 = 28,800 Hz is frequency by UART to timer 1 to set baud rate.

(a) 28,800 / 3 = 9600 where -3 = FD (hex) is loaded into TH1
(b) 28,800 / 12 = 2400 where -12 = F4 (hex) is loaded into TH1
(c) 28,800 / 24 = 1200 where -24 = E8 (hex) is loaded into TH1

Notice that dividing 1/12 of the crystal frequency by 32 is the default value upon activation of the 8051 RESET pin.



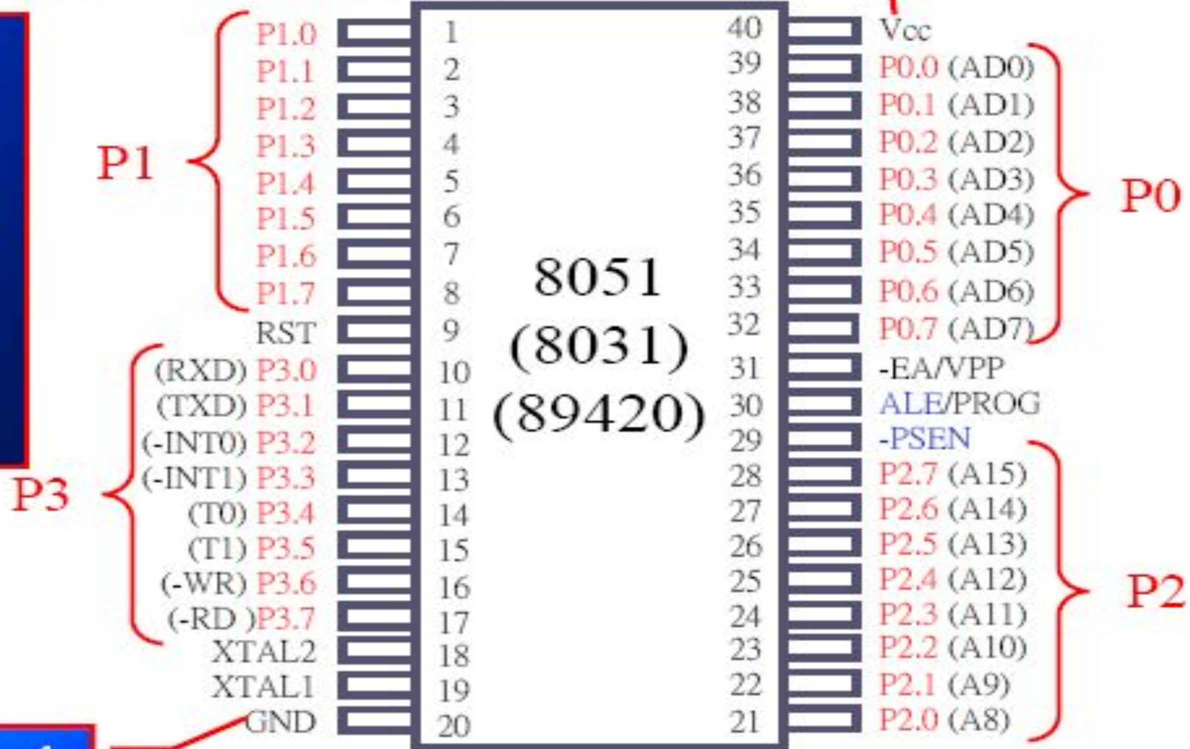
Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

TF is set to 1 every 12 ticks, so it functions as a frequency divider

8051 Pin Diagram

A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins

Provides +5V supply voltage to the chip



Grond

- SBUF is an 8-bit register used solely for serial communication
  - For a byte data to be transferred via the TxD line, it must be placed in the SBUF register
    - The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line
  - SBUF holds the byte of data when it is received by 8051 RxD line
    - When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF

```
MOV SBUF,#'D'    ;load SBUF=44h, ASCII for 'D'
MOV SBUF,A       ;copy accumulator into SBUF
MOV A,SBUF       ;copy SBUF into accumulator
```

- SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW

Note: *Make SM2, TB8, and RB8 =0*

- REN (receive enable)
  - It is a bit-adressable register
    - When it is high, it allows 8051 to receive data on RxD pin
    - If low, the receiver is disable
- TI (transmit interrupt)
  - When 8051 finishes the transfer of 8-bit character
    - It raises TI flag to indicate that it is ready to transfer another byte
    - TI bit is raised at the beginning of the stop bit
- RI (receive interrupt)
  - When 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in SBUF register
    - It raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost
    - RI is raised halfway through the stop bit

- In programming the 8051 to transfer character bytes serially
  1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate
  2. The TH1 is loaded with one of the values to set baud rate for serial data transfer
  3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
  4. TR1 is set to 1 to start timer 1
  5. TI is cleared by `CLR TI` instruction
  6. The character byte to be transferred serially is written into SBUF register
  7. The TI flag bit is monitored with the use of instruction `JNB TI,xx` to see if the character has been transferred completely
  8. To transfer the next byte, go to step 5

Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.

Solution:

```
        MOV  TMOD,#20H   ;timer 1,mode 2(auto reload)
        MOV  TH1,#-6     ;4800 baud rate
        MOV  SCON,#50H   ;8-bit, 1 stop, REN enabled
        SETB TR1         ;start timer 1
AGAIN:  MOV  SBUF,#"A"   ;letter "A" to transfer
HERE:   JNB  TI,HERE     ;wait for the last bit
        CLR  TI          ;clear TI for next char
        SJMP AGAIN       ;keep sending A
```

Write a program for the 8051 to transfer "YES" serially at 9600 baud, 8-bit data, 1 stop bit, do this continuously

Solution:

```
        MOV  TMOD,#20H  ;timer 1,mode 2(auto reload)
        MOV  TH1,#-3    ;9600 baud rate
        MOV  SCON,#50H  ;8-bit, 1 stop, REN enabled
        SETB TR1        ;start timer 1
AGAIN:  MOV  A,#"Y"     ;transfer "Y"
        ACALL TRANS
        MOV  A,#"E"     ;transfer "E"
        ACALL TRANS
        MOV  A,#"S"     ;transfer "S"
        ACALL TRANS
        SJMP AGAIN      ;keep doing it
;serial data transfer subroutine
TRANS:  MOV  SBUF,A     ;load SBUF
HERE:   JNB  TI,HERE    ;wait for the last bit
        CLR  TI         ;get ready for next byte
        RET
```

- The steps that 8051 goes through in transmitting a character via TxD
  1. The byte character to be transmitted is written into the SBUF register
  2. The start bit is transferred
  3. The 8-bit character is transferred on bit at a time
  4. The stop bit is transferred
     - It is during the transfer of the stop bit that 8051 raises the TI flag, indicating that the last character was transmitted
  5. By monitoring the TI flag, we make sure that we are not overloading the SBUF
     - If we write another byte into the SBUF before TI is raised, the untransmitted portion of the previous byte will be lost
  6. After SBUF is loaded with a new byte, the TI flag bit must be forced to 0 by `CLR TI` in order for this new byte to be transferred

- By checking the TI flag bit, we know whether or not the 8051 is ready to transfer another byte
  - It must be noted that TI flag bit is raised by 8051 itself when it finishes data transfer
  - It must be cleared by the programmer with instruction `CLR TI`
  - If we write a byte into SBUF before the TI flag bit is raised, we risk the loss of a portion of the byte being transferred
- The TI bit can be checked by
  - The instruction `JNB TI,xx`
  - Using an interrupt

- In programming the 8051 to receive character bytes serially
  1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate
  2. TH1 is loaded to set baud rate
  3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
  4. TR1 is set to 1 to start timer 1
  5. RI is cleared by `CLR RI` instruction
  6. The RI flag bit is monitored with the use of instruction `JNB RI,xx` to see if an entire character has been received yet
  7. When RI is raised, SBUF has the byte, its contents are moved into a safe place
  8. To receive the next character, go to step 5

Write a program for the 8051 to receive bytes of data serially, and put them in P1, set the baud rate at 4800, 8-bit data, and 1 stop bit

Solution:

```
        MOV  TMOD,#20H  ;timer 1,mode 2(auto reload)
        MOV  TH1,#-6    ;4800 baud rate
        MOV  SCON,#50H  ;8-bit, 1 stop, REN enabled
        SETB TR1        ;start timer 1
HERE:   JNB  RI,HERE    ;wait for char to come in
        MOV  A,SBUF     ;saving incoming byte in A
        MOV  P1,A       ;send to port 1
        CLR  RI         ;get ready to receive next
                        ;byte
        SJMP HERE       ;keep getting data
```

- An *interrupt* is an external or internal event that interrupts the microcontroller to inform it that a device needs its service
- A single microcontroller can serve several devices by two ways
  - Interrupts
    - Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal
    - Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device
    - The program which is associated with the interrupt is called the *interrupt service routine* (ISR) or *interrupt handler*

- (cont')
  - Polling
    - The microcontroller continuously monitors the status of a given device
    - When the conditions met, it performs the service
    - After that, it moves on to monitor the next device until every one is serviced
- Polling can monitor the status of several devices and serve each of them as certain conditions are met
  - The polling method is not efficient, since it wastes much of the microcontroller's time by polling devices that do not need service
  - ex. `JNB TF,target`

- The advantage of interrupts is that the microcontroller can serve many devices (not all at the same time)
  - Each devices can get the attention of the microcontroller based on the assigned priority
  - For the polling method, it is not possible to assign priority since it checks all devices in a round-robin fashion

- For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler
  - When an interrupt is invoked, the micro-controller runs the interrupt service routine
  - For every interrupt, there is a fixed location in memory that holds the address of its ISR
  - The group of memory locations set aside to hold the addresses of ISRs is called interrupt vector table

- Upon activation of an interrupt, the microcontroller goes through the following steps
  1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack
  2. It also saves the current status of all the interrupts internally (i.e: not on the stack)
  3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR

4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it
   - It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted
   - First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC
   - Then it starts to execute from that address

- Six interrupts are allocated as follows
  - Reset – power-up reset
  - Two interrupts are set aside for the timers: one for timer 0 and one for timer 1
  - Two interrupts are set aside for hardware external interrupts
    - P3.2 and P3.3 are for the external hardware interrupts INT0 (or EX1), and INT1 (or EX2)
  - Serial communication has a single interrupt that belongs to both receive and transfer

Interrupt vector table

Interrupt	ROM Location (hex)	Pin
Reset	0000	9
External HW (INT0)	0003	P3.2 (12)
Timer 0 (TF0)	000B	
External HW (INT1)	0013	P3.3 (13)
Timer 1 (TF1)	001B	
Serial COM (RI and TI)	0023	

- Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated
- The interrupts must be enabled by software in order for the microcontroller to respond to them
  - There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the interrupts

IE (Interrupt Enable) Register

D7 D0

EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA (enable all) must be set to 1 in order for rest of the register to take effect

EA	IE.7	Disables all interrupts
--	IE.6	Not implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow or capture interrupt (8952)
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 1 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt 0

- To enable an interrupt, we take the following steps:
  1. Bit D7 of the IE register (EA) must be set to high to allow the rest of register to take effect
  2. The value of EA
     - If EA = 1, interrupts are enabled and will be responded to if their corresponding bits in IE are high
     - If EA = 0, no interrupt will be responded to, even if the associated bit in the IE register is high

- The 8051 has two external hardware interrupts
  - Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts
    - The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
  - There are two activation levels for the external hardware interrupts
    - Level trigged
    - Edge trigged