Chapter 1

# Introduction to Microcontrollers

(I. Scott Mackenzie)

## Introduction

- Computers have only been with us for a few decades but their impact (direct or indirect) on our lives is profound.
- Usually these are supposed to be just *data processors performing exhaustive numeric operations*. But there presence is unnoticed at most of the places; like
  - *At supermarkets* in Cash Registers, Weighing Scales, etc.
  - *At home* in Ovens, Washing Machines, Alarm Clocks, etc.
  - *At play* in Toys, VCRs, Stereo Equipment, etc.
  - *At office* in Typewriters, Photocopiers, Elevators, etc.
  - *In industry* in Industrial Automation, safety systems, etc.
  - *On roads* in Cars, Traffic Signals, etc.

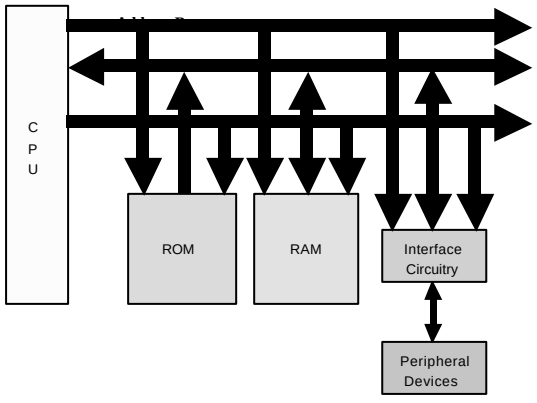
## Microcontrollers

- Computers can be divided into following two main types depending on their function
  - General purpose (Not Transparent)
  - Special purpose (Transparent)
- Microcontrollers are more suitable for special purpose devices.
- Microcontroller is a device similar to microprocessor but includes more circuitry in the same chip.

## Terminology

- A computer is defined by two main qualities
  - The ability to be programmed to operate on data without human intervention
  - The ability to store and retrieve data.
- Computers also have peripheral devices to communicate with outside world.

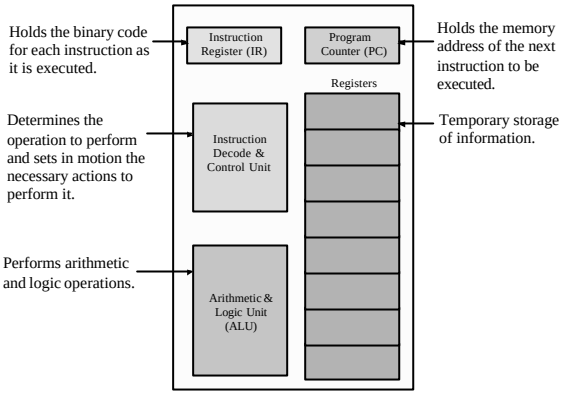
## Block Diagram of Microcomputer System




## Central Processing Unit (CPU)

- Brain of the computer system, administers all activity in the system and performs all operations on data.
- Continuously performs two operations: fetching and executing instructions.
- Understand and execute instructions based on a set of binary codes called the instruction set.

## Simplified Block Diagram of CPU




## Fetching & Executing An Instruction

- Fetching involves the following steps:
  a) Contents of PC are placed on address bus
  b) READ signal is activated
  c) Data (instruction opcode) are read from RAM and placed on data bus
  d) Opcode is latched into the CPU's internal instruction register
  e) PC is incremented to prepare for the next fetch from memory
- Execution involves decoding the opcode and generating control signals to gate internal registers in and out of the ALU and to signal the ALU to perform the specified operation.

## The Buses: Address, Data, & Control

- A **bus** is a collection of wires carrying information with a common purpose.
- For each *read* or *write* operation, the CPU specifies the location of the data or instruction by placing an address on the **address bus**, then activates a signal on the **control bus** indicating whether the operation is read or write.
- **Read operations** retrieve a byte of data from memory at the location specified and place it on the **data bus**. CPU reads the data and places it in one of its internal registers.
- **Write operations** put data from CPU on the **data bus** and store it in the location specified.

## The Buses (contd.)

- **Address bus** carries the address of a specified location. For ***n*** address lines, $2^n$ locations can be accessed. E.g., A 16-bit address bus can access $2^{16}$ = 65,536 locations or 64K locations ($2^{10}$ = 1024 = 1K, $2^6$ = 64).
- **Data bus** carries information between the CPU and memory or between the CPU and I/O devices. Computers spend up to two-thirds of their time simply moving data, so the number of lines of the data bus is important for overall performance. This limitation by width of data bus is a bottleneck even with a vast amount of memory on the system and a high speed CPU. 16-bit computer means…?
- **Control bus** carries control signals supplied by the CPU to synchronize the movement of information on the address and data bus.

## Input/Output Devices

- I/O devices or computer peripherals provide the path for communication between the computer system and the real world. Three main types:
  - Mass Storage Devices (Hard disk, magnetic tape, CD-ROM, etc.)
  - Human Interface (Keyboard, mouse, joystick, CRT, printer, speaker, etc.)
  - Control/Monitor devices (Phototransistors, sensors, thermistors, switches, motors, relays, etc.)
- **Control devices** are outputs, or actuators, that can affect the world around them when supplied with a voltage or current.
- **Monitoring devices** are inputs, or sensors, that are stimulated by temperature, pressure, light, motion, etc. and convert this to voltage or current read by the computer.

**The interface circuitry converts the voltage or current to binary data, or vice versa.**

## Computer Classification

- Computers can be classified by their size and power as microcomputers, minicomputers, or mainframe computers.
- *Microcomputers* contain single chip CPU (microprocessor).
- *Minicomputers* contain CPU consists of several chips.
- *Mainframes* contain CPU consists of several circuit boards of chips.
- *Microcomputers* are single-user, single-task systems while *minicomputers*, and *mainframe computers* are multi-user and multitasking systems.

## Microprocessors Vs. Microcontrollers

- Microprocessor is a single chip CPU, microcontroller contains, a CPU and much of the remaining circuitry of a complete microcomputer system in a single chip.
- Microcontroller includes RAM, ROM, serial and parallel interface, timer, interrupt schedule circuitry (in addition to CPU) in a single chip.
  - RAM is smaller than that of even an ordinary microcomputer, but enough for its applications.
  - Interrupt system is an important feature, as microcontrollers have to respond to control oriented devices in real time. E.g., opening of microwave oven's door cause an interrupt to stop the operation.

  (Most microprocessors can also implement powerful interrupt schemes, but external components are usually needed.)

## Microprocessors Vs. Microcontrollers (contd.)

- Microprocessors are most commonly used as the CPU in microcomputer systems. Microcontrollers are used in small, minimum component designs performing control-oriented activities.
- Microprocessor instruction sets are "processing intensive", implying powerful addressing modes with instructions catering to large volumes of data. Their instructions operate on nibbles, bytes, etc. Microcontrollers have instruction sets catering to the control of inputs and outputs. Their instructions operate also on a single bit. E.g., a motor may be turned ON and OFF by a 1-bit output port.

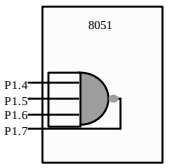
## Gains and Losses

- **Gains:** Reduced component count in a circuit, high degree of integration, shorter development time, lower manufacturing cost, lower power consumption, higher reliability, etc.
- **Losses:** Some situations (very few) require extremely fast response to events are poorly handled by the microcontrollers. E.g., Implementation of NAND operation using an 8051 microcontroller.

```
LOOP: MOV   C, P1.4     ;READ P1.4 BIT INTO CARRY FLAG
      ANL   C, P1.5     ;AND WITH P1.5
      ANL   C, P1.6     ;AND WITH P1.6
      CPL   C           ;CONVERT TO NAND BY INVERTING
      MOV   P1.7, C     ;SEND TO P1.7 OUTPUT BIT
      SJMP  LOOP        ;REPEAT
```

## Gains and Losses (contd.)

- The propagation delay can be measured by a voltmeter or by an oscilloscope. It is 3 microsecond (assuming 8051 operation using 12 MHz crystal frequency) while equivalent TTL has delay of 10 nanosecond.




Chapter 2

# Hardware Summary

(I. Scott Mackenzie)

## The 8051

- A Microcontroller derivative family based on the 8051 core.
- A Microcontroller because a one-chip system can be made with the one chip containing:
  - Program & Data Memory
  - I/O Ports
  - Serial Communication
  - Counters/Timers
  - Interrupt Control logic
  - A-to-D and D-to-A converters
  - & so on ...

# MCS-51 Family Overview

- Term 8051 refers to MCS-51 family of microcontroller ICs by Intel Corp. (From 8031-8752)
- Features are summarized below:
  - 8 Bit data path and ALU.
  - Easy interfacing.
  - 12 to 30 MHz versions available.
    (1 µsec to 400 ns for single cycle instructions).
  - Full instruction set including:
    Multiply and Divide.
    Bit set, reset, and test (Boolean instructions).
  - Variety of addressing modes.

# Hardware Features of the 8051

–0K (8031), ROM 4K (8051), EPROM 4K (8751)

–RAM 128 bytes (8XX1), 256 bytes (8XX2)
(where X= 0 or 7 & X=3 or 5)

–Four 8-bit I/O Ports (P0-P3)

–Two 16-bit Timers/Counters (T0 &T1)

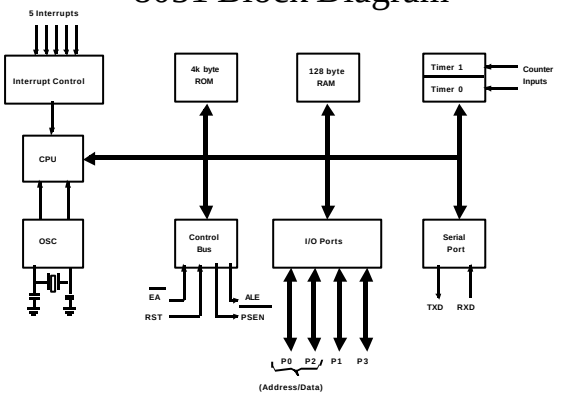
–Serial I/O Port

–Boolean Processor (Operates on Single Bits)

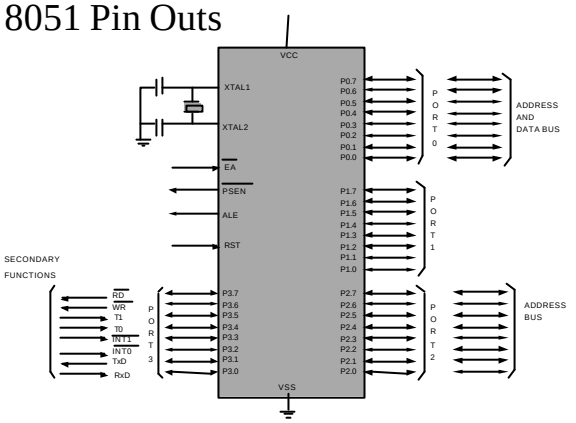
–210 bit-addressable locations

–Oscillator & Clock Circuit

# 8051 Block Diagram




# 8051 Pin Outs

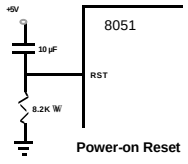


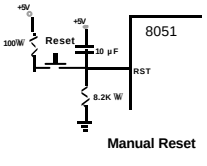

# 8051 has 4 Bus Control Signals

- $\overline{\text{PSEN}}$ (pin 29): (Program Store Enable) enables external program (code) memory. Usually connected to EPROM's output enable ($\overline{\text{OE}}$). It pulses low during fetch stage of an instruction. It remains high while executing a program from internal ROM.
- ALE (pin 30): (Address Latch Enable) used for demultiplexing the address and data bus when port 0 is used as the data bus and low-byte of address bus.
- $\overline{\text{EA}}$ (pin 31): (External Access) high to execute programs from internal ROM and low to execute from external memory only.
- RST (pin 9): (RESET) master reset of 8051.

# Reset

- External reset is asynchronous to the internal clock.
- RST pin must be high for at least two machine cycles while the oscillator is running.
- Internal RAM is not affected by reset.
- Reset sets PC to 0000H.
- Typical circuits:



## 8051 Oscillator & Power Pins

- Pins 18 and 19 are the oscillator pins to connect the crystal of nominal frequency 12 MHz.
- Pin 40 is for +5V and pin 20 is for GND.

## I/O Ports

- Four 8-bit I/O ports.
- Most have alternate functions.
- Bi-directional.

## Port 0 (pin 32-39)

- Dual purpose I/O port.
- In min. component design, it is used as a general purpose I/O port.
- In larger designs with external memory, it becomes a multiplexed data bus:
  - Low byte of address bus, strobed by ALE.
  - 8-bit instruction bus, strobed by PSEN.
  - 8-bit data bus, strobed by WR and RD.

## Port 1 (pin 1-8)

- As an I/O port:

  Standard bi-directional port for interfacing to external devices as required for I/O.
- Alternate functions:

  Only on some derivatives.

## Port 2 (pin 21-28)

- Dual purpose I/O port.
- As an I/O port:

  Standard bi-directional general purpose I/O port.
- Alternate functions:

  High byte of address bus for external program and data memory accesses.

## Port 3 (pin 10-17)

- Dual purpose I/O port.
- As an I/O port:

  Standard bi-directional general purpose I/O port.
- Alternate functions:

  Serial I/O - TXD, RXD

  Timer clocks - T0, T1

  Interrupts - INT0, INT1

  Data memory - RD, WR

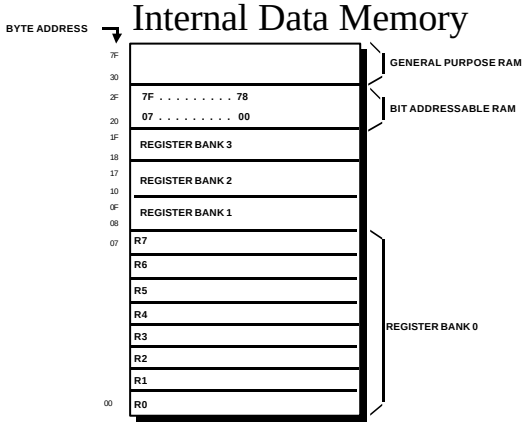
## Addressing Space

- 64K x 8 ROM - External Program Memory. (Enabled via $\overline{\text{PSEN}}$)
- 64K x 8 RAM - External Data Memory. (Enabled via $\overline{\text{RD}}$ and $\overline{\text{WR}}$)
- 256 x 8 RAM - Internal Data Memory.
- 128 x 8 Special Function Registers (SFRs).
- Bit addressing of 16 RAM locations and 16 SFRs.

## Internal Data Memory

- Four register banks (Register Bank 0-3):
  **00 to 1F** hexadecimal.
- Bit addressable RAM (128 bits):
  **20 to 2F** hexadecimal.
- General purpose RAM (directly addressable range):
  **30 to 7F** hexadecimal.
- Special function registers (indirectly addressable range):
  **80 to FF** hexadecimal.

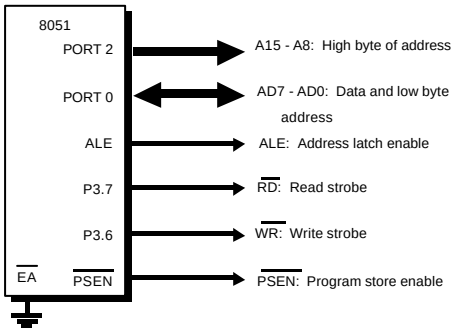
## Internal Data Memory




- Any location on general purpose RAM can be accessed freely using direct or indirect addressing modes.
  E.g., MOV A, 5FH ;contents of 5FH location will be loaded in A
  E.g., MOV R0, #5FH ; value 5FH will be loaded in register R0
  MOV A, @R0 ; data will be loaded in A which is pointed
  ; at by R0

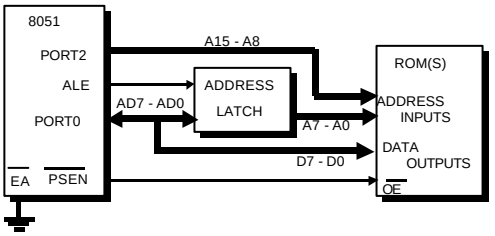
- Powerful feature that bits can be set, cleared, ANDed, ORed, etc. with a single instruction
  E.g., SETB 67H ; to set bit 67H
  Most microprocessors will do like
  MOV A, 2CH ; read entire byte
  ORL A, #10000000B ; set MSB
  MOV 2CH, A ; write back entire byte

## External Bus Expansion



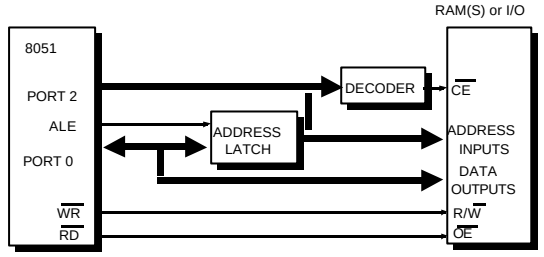

## External Program Memory

- 64K byte address space.
- Enabled by PSEN signal.



# External Data Memory

- 64K byte address space.
- The only access to this memory is with the MOVX instruction, using either 16-bit data pointer DPTR, R0, or R1 as the address register.




# Special Function Register Space

- 128 byte address space, directly addressable as 80 to FF hex.
- 16 addresses are bit addressable:

  Set, Clear, AND, OR, MOV (those ending with 0 or 8).
- This space contains:
  - Special purpose CPU registers.
  - I/O control registers.
  - I/O ports.

# Special Function Register Map

| Bit Addressable | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| F8 | | | | | | | | |
| F0 | B | | | | | | | |
| E8 | | | | | | | | |
| E0 | ACC | | | | | | | |
| D8 | | | | | | | | |
| D0 | PSW | | | | | | | |
| C8 | | | | | | | | |
| C0 | | | | | | | | |
| B8 | IP | | | | | | | |
| B0 | P3 | | | | | | | |
| A8 | IE | | | | | | | |
| A0 | P2 | | | | | | | |
| 98 | SCON | SBUF | | | | | | |
| 90 | P1 | | | | | | | |
| 88 | TCON | TMOD | TL0 | TL1 | TH0 | TH1 | | |
| 80 | P0 | SP | DPL | DPH | | | | PCON |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

BYTE ADDRESS

# Special Function Registers

CPU registers:

- ACC : Accumulator.
- B : B register.
- PSW : Program Status Word.
- SP : Stack Pointer.
- DPTR : Data Pointer (DPH, DPL).

Interrupt control:

- IE : Interrupt Enable.
- IP : Interrupt Priority.

I/O Ports:

- P0 : Port 0.
- P1 : Port 1.
- P2 : Port 2.
- P3 : Port 3.

# Special Function Registers (cont'd)

Timers:

- TMOD : Timer mode.
- TCON : Timer control.
- TH0 : Timer 0 high byte.
- TL0 : Timer 0 low byte.
- TH1 : Timer 1 high byte.
- TL1 : Timer 1 low byte.

Serial I/O:

- SCON : Serial port control.
- SBUF : Serial data registers.

Other:

- PCON : Power control

# PSW : Program Status Word

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

- CY : Carry Flag.
- AC : Auxiliary Carry Flag.
- F0 : Flag 0 (available for user).
- RS1: Register Select 1.
- RS0: Register Select 0.
- OV : Arithmetic Overflow Flag.
- P : Accumulator Parity Flag.

• Flags are 1-bit registers provided to store the results of certain program instructions. In order to conveniently address the flags, they are grouped inside the PSW register.

## PSW (Program Status Word)

- CY: (Carry Flag) is dual purpose: **(1)** As traditional CY for arithmetic operations e.g., If A contains FFH then the instruction **ADD A, #1**
  leaves A equal to 00H and sets the CY in PSW. (A=00H & CY=1)
  **(2)** As Boolean accumulator e.g., ANL C, 25H ; ANDs bit 25H with the carry flag and places the result back in the CY.
- AC: (Auxiliary Carry Flag) used in addition of BCD numbers, is set if a carry was generated out of bit 3 into bit 4. If the values are added are BCD, then the add instruction must be followed by DAA (decimal adjust accumulator) to bring results greater than 9 back into range.
- F0: (Flag 0) is a general-purpose flag bit available for user applications.

## PSW (Contd.)

- OV: (Overflow flag) is set after an addition or subtraction operation if there was an arithmetic overflow. Results greater than +127 or less than –128 will set OV bit.
- P: (Parity Bit) automatically set or cleared each machine cycle to establish even parity with the accumulator. Parity bit is most commonly used in conjunction with serial port routines to include a parity bit before or after the transmission.
- RS1 & RS0 are used to select different register banks.

| RS1 | RSO | Register Bank | Address |
|---|---|---|---|
| 0 | 0 | 0 | 00h - 07h |
| 0 | 1 | 1 | 08h - 0Fh |
| 1 | 0 | 2 | 10h - 17h |
| 1 | 1 | 3 | 18h - 1Fh |

## SFRs (Special Function registers)

- **B Register:** (at F0H) also bit addressable and used along with the accumulator for multiply & divide operations.
  E.g., **MUL A B** instruction multiplies the 8-bit unsigned values in A & B and leaves the 16-bit result in A (low-byte) & B (high-byte)
  E.g., **DIV A B** instruction divides A by B leaving the integer result in A and remainder in B.
- **SP:** (Stack Pointer) (at 81H) is an 8-bit register contains the address of the data item currently on the top of stack. Its operations include "Pushing" & "Popping" data from the stack.
- **DPTR:** (Data Pointer) is 16-bit register at 82H (DPL, low-byte) and 83H (DPH, high-byte) used to access external code or data memory. It can be specified by its 16-bit name, DPTR, or by each individual byte name, DPH and DPL.

Chapter 3

# Instruction Set Summary

(I. Scott Mackenzie)

## 8051 Addressing Modes

There are basically **5 ways of specifying source/destination** operand addresses:

**1. Particular On-chip Resources:**

This includes the Accumulator (A), the Stack Pointer (SP), the Data Pointer (DP), the Program Counter (PC), and the Carry (C). Other On-chip Registers are Memory-mapped while these have special Op-codes.

**2. Immediate operands:**

The # sign is the designator. These are 8-bits except for DPTR contents (16-bits).

**3. Register operands:**

Designated as Rn, where n is 0..7. One of the four Register Banks is used (selected by RS0 and RS1 in PSW).

**4. Direct Operands:**

From 00 to FF Hex, specifies one of the internal data addresses.

**5. Indirect Address:**

Designated as @Ri, where i is 0 or 1, uses the contents of R0 or R1 in the selected Register Bank to specify the address. Other form is @A, using Accumulator contents.

## 8051 Addressing Modes

Addressing modes are an integral part of each computers instruction set. They allow different ways of specifying source/destination operand addresses depending on the programming situation. There are **8 modes of addressing**:

1. **Immediate**
2. **Register**
3. **Direct**
4. **Indirect**
5. **Relative**
6. **Absolute**
7. **Long**
8. **Indexed**

# Instruction Set : Arithmetic

| Mnemonics | Operands | Bytes/Cycles |
|---|---|---|
| ADD A, Rn | 1/1 | |
| ADDC | A, direct | 2/1 |
| SUBB | A, @Ri | 1/1 |
| | A, #data | 2/1 |
| INC | A | 1/1 |
| DEC Rn | 1/1 | |
| | direct | 2/1 |
| | @Ri | 1/1 |
| INC | DPTR | 1/2 |
| MUL AB | 1/4 | |
| DIV | AB | 1/4 |
| DA | A | 1/1 |

In Rn, n is 0..7. One of the four Register Banks is used (selected by RS0 and RS1 in PSW)

```
MOV PSW, #00011000B    ; Select Register Bank 3
ADD A, R7              ; Add the contents of Register 7 to the Acc.
```

In Ri, i is 0 or 1.

# Instruction Set : Logical

| Mnemonic | Operands | Bytes/Cycles |
|---|---|---|
| ANL A, Rn | 1/1 | |
| ORL A, direct | 2/1 | |
| XRL A, @Ri | 1/1 | |
| | A, #data | 2/1 |
| | direct, A | 2/1 |
| | direct, #data | 3/2 |
| | C, bit | 2/2 |
| CLR A | 1/1 | |
| CPL C | 1/1 | |
| | bit | 2/1 |

# Instruction Set : Logical (cont'd)

| Mnemonic | Operands | Bytes/Cycles |
|---|---|---|
| RL | A | 1/1 |
| RLC A | 1/1 | |
| RR | A | 1/1 |
| RRC | A | 1/1 |
| SWAP | A | 1/1 |
| SETB | C | 1/1 |
| CLR bit | 2/1 | |
| CPL | | |

# Instruction Set : Data Transfer

| Mnemonic | Operands | Bytes/Cycles |
|---|---|---|
| MOV | A, Rn | 1/1 |
| | A, direct | 2/1 |
| | A, @Ri | 1/1 |
| | A, #data | 2/1 |
| | Rn, A | 1/1 |
| | Rn , direct | 2/2 |
| | Rn, #data | 2/1 |
| | direct, A | 2/1 |
| | direct, Rn | 2/2 |
| | direct, direct | 3/2 |
| | direct, @Ri | 2/2 |
| | direct, #data | 3/2 |

# Instruction Set : Data Transfer (cont'd)

| Mnemonic | Operands | Bytes/Cycles |
|---|---|---|
| MOV | @Ri, A | 1/1 |
| | @Ri, direct | 2/2 |
| | @Ri, #data | 2/1 |
| | DPTR, #data16 | 3/2 |
| | C, bit | 2/1 |
| | bit, C | 2/2 |
| MOVX | A,@DPTR | 1/2 |
| | @DPTR,A | 1/2 |
| | A,@Ri | 1/2 |
| | @Ri,A | 1/2 |

# Instruction Set: Data Transfer (cont'd)

| Mnemonic | Operands | Bytes/Cycles |
|---|---|---|
| MOVC | A, @A+DPTR | 1/2 |
| | A, @A+PC | 1/2 |
| PUSH | direct | 2/2 |
| POP direct | 2/2 | |
| XCH A, Rn | 1/1 | |
| | A, direct | 2/1 |
| | A, @Ri | 1/1 |
| XCHD | A, @Ri | 1/1 |

## Instruction Set : Branching

| Mnemonic | Operands | Bytes/Cycles |
|---|---|---|
| LCALL | addr16 | 3/2 |
| ACALL | addr11 | 2/2 |
| RET - | 1/2 | |
| RETI | - | 1/2 |
| LJMP | addr16 | 3/2 |
| AJMP | addr11 | 2/2 |
| SJMP | rel | 2/2 |
| JMP @A+DPTR | 1/2 | |
| JZ | rel | 2/2 |
| JNZ | rel | 2/2 |

## Instruction Set : Branching (cont'd)

| Mnemonic | Operands | Bytes/Cycles |
|---|---|---|
| CJNE | A, direct, rel | 3/2 |
| | A, #data, rel | 3/2 |
| | Rn, #data, rel | 3/2 |
| | @Ri,#data,rel | 3/2 |
| DJNZ | Rn, rel | 2/2 |
| | direct, rel | 3/2 |
| NOP | - | 1/1 |
| JC | rel | 2/2 |
| JNC rel | 2/2 | |
| JB | bit, rel | 3/2 |
| JNB | bit, rel | 3/2 |
| JBC | bit, rel | 3/2 |

Chapter 4

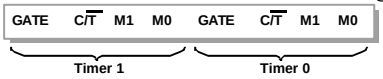
# Timer Operation

(I. Scott MacKenzie)

## Counter / Timers (T0 & T1)

- Timer is a series of divide-by-two flip-flops that receive an input signal as a clocking source.
- Clock is applied to the first flip-flop, which gives output divided by 2.
- That output of first flip-flop clocks the second flip-flop,which also divides it by 2 and so on.
- The output of the last stage clocks a timer overflow flip-flop, or flag, which is tested by the software.
- It is like a counter. A 16-bit timer would count from 0000H to FFFFH. The overflow flag is set on the FFFFH-to-0000H count.
- There are two timers in 8051 i.e., T0 and T1.
- Four modes of timer operations.

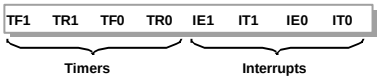
## TMOD : Counter/Timer MODe Register




• TMOD is not bit addressable. It is loaded, generally, by the software at the beginning of a program to initialize the timer mode.

- GATE : Permits INTx pin to enable/disable the counter.
- C/$\overline{T}$ : Set for counter operation, reset for timer operation.
- M1, M0 :
  - 00 : Mode 0 - 13-bit timer mode (Emulates 8048 counter/timer).
  - 01 : Mode 1 - 16-bit timer mode.
  - 10 : Mode 2 - 8-bit auto-reload mode.
  - 11 : Mode 3 - Split timer mode (Timer 0 = two 8-bit timers).

## TCON : Counter/Timer CONtrol Register



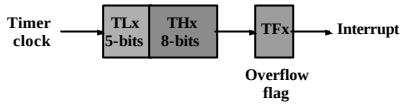

- TF1, TF0 : Overflow flags for Timer 1 and Timer 0.
- TR1, TR0 : Run control bits for Timer 1 and Timer 0.
  Set to run, reset to hold.

- IE1, IE0 : Edge flag for external interrupts 1 and 0. *
  Set by interrupt edge, cleared when interrupt is processed.
- IT1, IT0 : Type bit for external interrupts. *
  Set for falling edge interrupts, reset for 0 level interrupts.

* = not related to counter/timer operation but used to detect and initiate external interrupts.

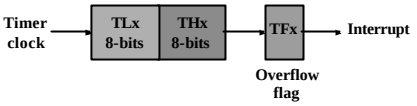
## Timer Modes

- **Timer Mode 0 (13-bit Timer):**
  - Timer high-byte (THx) is cascaded with the 5 least-significant bits of the timer low-byte (TLx) to form a 13-bit timer, where x = 0 or 1.
  - Upper 3-bits of TLx are not used.
  - Overflow occurs on the 1FFFH-to-0000H and sets the timer overflow flag.
  - MSB is THx bit 7, and LSB is TLx bit 0.
  - MOV TMOD, #00H ; setting both timers to mode 0



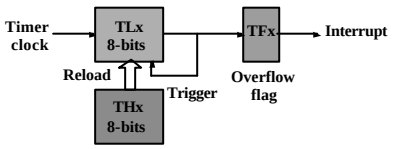

## Timer Modes (cont'd)

- **Timer Mode 1 (16-bit Timer):**
  - Same as mode 0 except that it is 16-bit. Timer high-byte (THx) is cascaded the timer low-byte (TLx) to form a 16-bit timer, where x = 0 or 1.
  - Clock is applied to the combined high and low-byte timer registers.
  - Overflow occurs on the FFFFH-to-0000H and sets the timer overflow flag.
  - MSB is THx bit 7, and LSB is TLx bit 0.
  - LSB toggles at ***clock frequency/2*** and MSB at ***clock frequency/$2^{16}$***



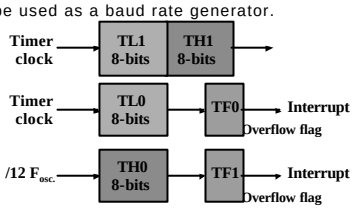

## Timer Modes (cont'd)

- **Timer Mode 2 (Auto-Reload):**
  - Timer low-byte (TLx) operates as an 8-bit timer while the timer high-byte (THx) holds a reload value.
  - When the count overflows from FFH-to-00H, not only the timer flag set, but also the value in THx is loaded into TLx, and counting continues from this value up to next FFH-to-00H, so on.




## Timer Modes (cont'd)

- **Timer Mode 3:**
  - Timer 0 Splits into two 8-bit counter/timers. TL0 and TH0 act as two separate timers with overflows setting the TF0 and TF1 respectively.
  - Timer 1 (when timer 0 is in mode 3 ):
    - Counter stopped if in mode 3
    - Can be used in mode 0, 1, or 2
    - Has gate (INT1) and external input (T1), but no flag or interrupt.
    - May be used as a baud rate generator.




## Clocking Sources

1. Interval Timing
2. Event Counting

## Clocking Sources (contd.)

1. Interval Timing:
   - If C/$\overline{T}$ = 0 (in TMOD), timer operation is selected and timer is clocked from on-chip oscillator. A divide-by-12 clock frequency is applied.
   - Timer registers (TLx/THx) increment at a rate of $1/12^{th}$ the frequency of on-chip oscillator.
   - 12 MHz crystal would yield a clock rate of 1 MHz.
   - Timer overflows occur after a fixed number of clocks, depending on the initial value loaded into the timer registers.

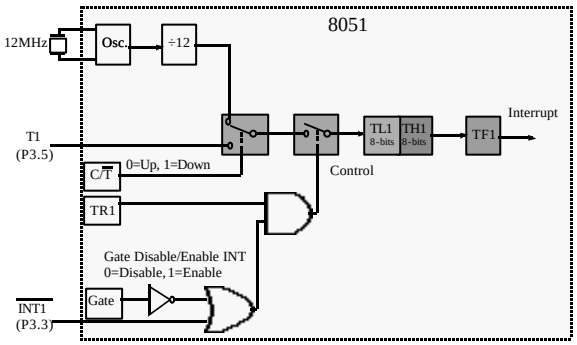
## Clocking Sources (contd.)

2. Event Counting:
   - If C/$\overline{T}$ = 1 (in TMOD), counter operation is selected and timer is clocked from external source. Usually, external source supplies the timer with a pulse upon the occurrence of an event. Timer counts those events.
   - External clock source comes through P3.4 (for Timer 0) and P3.5 (for Timer 1).
   - Timer registers are incremented in response to a 1-to-0 transition at the external input.
   - Number of external events is determined in software by reading the timer registers TLx/THx.

## Start, Stop, and Control of Timers

- TRx bit in bit addressable register TCON is responsible for starting and stopping the counters
  - TRx = 0 stops/disables the timers (e.g., CLR TR1)
  - TRx = 1 starts/enables the timers (e.g., SETB TR0)
- System reset clears TRx, so timers are disabled by default.

## Timer 1 Operating in 16-bit (Mode 1)




## Initializing and Accessing Timer Registers

- Timers are usually initialized once at the beginning of the program to set the correct operating mode.
- Then, within the body of a program, the timers are started, stopped, flag bits are tested and cleared, timer registers read or updated, and so on, as required in the application.
- First register to be initialized is TMOD to set the mode of operation e.g.,

  MOV TMOD, #00010000B ; sets Timer 1 in mode 1, leave C/T = 0 and GATE = 0 for internal clocking, and clears the Timer 0 bits.

## Initializing and Accessing Timer Registers (Contd.)

- Secondly, registers to be initialized are TLx/THx. E.g., For 100 µs interval, the following instruction will do the job

  MOV TL1, #9CH ; $(-100)_{10}$ = FF9CH
  MOV TH1, #FFH ; load Timer 1 registers by FF9CH

## Initializing and Accessing Timer Registers (Contd.)

- The timer is then started by setting the run control bit i.e.,

  SETB TR1
- Overflow flag is automatically set 100 µs later. Following instruction will check that

  WAIT: JNB TF1, WAIT ; wait until overflow flag is set.
- When the timer overflows, it is necessary to stop the timer and clear the overflow flag in software by the following instructions:

  CLR TR1 ; stop Timer 1
  CLR TF1 ; clear overflow flag of Timer 1

## Short and Long Intervals

- Shortest possible interval is one machine cycle i.e., 1 ìs (using 12 MHz crystal frequency).
- An 8-bit counter can give maximum delay of 256 ìs because $2^8$=256.
- A 13-bit counter can give maximum delay of 8192 ìs because $2^{13}$=8192.
- A 16-bit counter can give maximum delay of 65536 ìs because $2^{16}$=65536.

  65536 ìs = 0.065536 sec = 0.066 sec (approx.)

## Short and Long Intervals

(Contd.)

- For more than 0.066 sec delay, there are two methods:
  1. Cascade Timer 0 and Timer 1 but in this way both timers will be tied up.
  2. Use one timer in 16-bit mode with a software loop counting overflows.

Chapter 5

# Serial Port Operation

(I. Scott MacKenzie)

## Introduction

- 8051 includes an on-chip serial port that can operate in four modes over a wide range of frequencies.
- Essential function of serial port is to perform parallel-to-serial conversion for output data, and serial-to-parallel conversion for input data.
- Transmission bit is P3.1 on pin 11 (TXD) and reception bit is P3.0 on pin 10 (RXD).
- Features full duplex (simultaneous reception and transmission).
- Receive buffering allowing one character to be received and held in a buffer while a second character is received. If the CPU reads the first character before the second is fully received, data are not lost.

- Two SFRs (SBUF & SCON) provide software access to serial port.
  - Writing to SBUF loads data to be transmitted and reading SBUF accesses received data.
  - SCON is a bit addressable register containing status bits and control bits. Control bits set the operating mode and status bits indicate the end of a character transmission or reception. The status bits are tested in software or programmed to cause an interrupt.
- Serial port frequency of operation (baud rate) can be fixed or variable.
  - Fixed is derived from on-chip oscillator and variable is supplied by Timer 1 which must be programmed accordingly.

## SCON : Serial Port CONtrol Register (098H)

| SMO | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |
|---|---|---|---|---|---|---|---|

- **SM0, SM1** : *Serial Port Mode bits*

| **Mode** | **Baud Rate** |
|---|---|
| 00 = Mode 0 : Shift register I/O | Fixed (oscillator frequency/12) |
| 01 = Mode 1 : 8-bit UART | Variable (set by timer) |
| 10 = Mode 2 : 9-bit UART | Fixed (osc frq/32 or osc frq/64 ) |
| 11 = Mode 3 : 9-bit UART | Variable (set by timer) |

- **SM2 :** *Serial Port Mode bit*

  Mode 0 : Not used.

  Mode 1 : 1 = Ignore bytes with no stop bit.

  Mode 2,3 : 0 = Set receive interrupt (RI) on all bytes.

  : 1 = Set RI on bytes where 9$^{th}$ bit is 1.

## SCON (contd.)

- **REN:** Receiver enable. Must be set to receive characters.
- **TB8:** Transmit bit 8. Ninth bit transmitted (in modes 2 and 3); set/cleared by software.
- **RB8:** Receive bit 8. Ninth bit received (in modes 2 and 3):

  Mode 0 : Not used.

  Mode 1 : Stop bit.

  Mode 2, 3 : Ninth data bit.
- **TI:** Transmit interrupt flag. Set at end of character transmission; cleared by software.
- **RI:** Receive interrupt flag. Set at end of character reception; cleared by software.

## Serial Interface

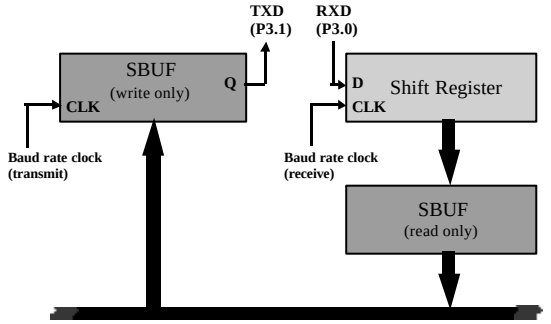
- Full duplex UART (Universal Asynchronous Receiver /Transmitter is a device that receives and transmits serial data with each data character preceded by a start bit "0" and followed by a stop bit "1"). Sometimes a parity bit is inserted between the last data bit and the stop bit.
- The essential operation of a UART is to perform parallel-to-serial conversion for output data, and serial-to-parallel conversion for input data.
- 10 or 11 bit frames.
- Interrupt driven.
- Registers:

  SCON - Serial port control register.

  SBUF - Read received data.

  - Write data to be transmitted.

## Serial Port Block Diagram




## Serial Interface Modes of Operation

**Mode 0:** 8-Bit Shift Register Mode. Terms RXD & TXD are misleading in this mode. RXD line is used for both input and output. TXD line serves as the clock.

- Eight bits are transmitted and received with the LSB first. Baud Rate is 1/12 of on-chip oscillator frequency.
- Transmission is initiated by any instruction that writes data to SBUF. Data are shifted out on RXD line with clock pulses sent out by the TXD line. Each transmitted bit is valid on the RXD pin for one machine cycle. E.g., `MOV SBUF, A`
- Reception is initiated when the receiver enable bit (REN) is 1 and the receive interrupt bit (RI) is 0. REN is set at the beginning of the program, and then clear RI to begin a data input operation. The clocking of data into serial port occurs on the positive edge of TXD.

**Mode 1:** Serial port operates as an 8-bit UART with a variable baud rate.10-bits are transmitted on TXD or received on RXD. Start bit (always 0), 8 data bits (LSB first), and a stop bit (always 1). For a receive operation, the stop bit goes into RB8 in SCON. Baud Rate Clock is variable using Timer 1 overflow or external count input.

- Transmission is initiated by writing data to SBUF, but does not start until the next rollover of the divide-by-16 counter supplying the serial port baud rate. Shifted data are outputted on the TXD line beginning with the start bit. The transmit interrupt flag (TI) is set as soon as the stop bit appears on TXD.
- Reception is initiated by a 1-to-0 transition on RXD. The divide-by-16 counter is immediately reset to align the counts with the incoming bit stream.

**Mode 2:** Serial port operates as a 9-bit UART with a fixed baud rate. 11-bits are transmitted or received. Start bit (always 0), 8 data bits (LSB first), a programmable 9th bit, and a stop bit (always 1).

- On transmission, the 9$^{th}$ bit whatever has been put in TB8 in SCON (may be a parity bit).
- On reception, the 9$^{th}$ bit is placed in RB8 in SCON.
- Baud Rate is programmable to either 1/32 or 1/64 of the on-chip oscillator frequency.

**Mode 3:** Serial port operates as a 9-bit UART with a variable baud rate. 11-bits are transmitted or received. Baud Rate is programmable and provided by the Timer 1 overflow or external input.

**Summary:**

**Baud rate:** Fixed in mode 2, variable in modes 1 & 3

**Data Bits:** Eight in mode 1, nine in modes 2 & 3

## Initialization

- **Receiver Enable Bit (REN):** must be set by software to enable the reception of characters at the beginning of a program when the serial port, timers, etc. are initialized. The instructions are

  **SETB REN or MOV SCON, #xxx1xxxxB**

- **The 9th Bit:** transmitted must be loaded into TB8 by software and received is placed in RB8.
- **Adding a Parity Bit:** is a common use of 9th bit. E.g., if communication requires 8 data bits plus even parity

```
MOV C, P        ; Put even parity bit in C flag
MOV TB8, C      ; This becomes the 9th data bit in TB8
MOV SBUF, A ; Move 8 bits from ACC to SBUF
```

- E.g., if communication requires 8 data bits plus odd parity

```
MOV C, P        ; Put even parity bit in C flag
CPL C           ; Convert to odd parity
MOV TB8, C      ; This becomes the 9th data bit in TB8
MOV SBUF, A ; Move 8 bits from ACC to SBUF
```

- Parity can be used in mode 1 also if the 7 data bits are used. E.g., 7-bit ASCII code with even parity can be transmitted as follows:

```
CLR ACC.7       ; Ensure MSB is clear
MOV C, P        ; Put even parity bit in C flag
MOV ACC.7, C ; Copy even parity bit into MSB
MOV SBUF, A  ; Send character
```

## Interrupt Flags (RI &TI)

- RI & TI in SCON play an important role in serial communications. Both bits are set by hardware but must be cleared by software.
  - RI is set at the end of character reception and indicates "receive buffer full".
  - This condition is tested in software or programmed to cause an interrupt.
  - If software wishes to input a character from the device connected to the serial port, it must wait until RI is set, then clear RI and read the character from SBUF.

```
WAIT: JNB RI, WAIT      ; Check RI until set
      CLR RI            ; Clear the flag
      MOV A, SBUF       ; Read character
```

## Interrupt Flags (RI &TI) contd.

- TI is set at the end of character transmission and indicates "transmit buffer empty".
- If software wishes to send a character to the device connected to the serial port, it must wait until TI is set (means previous character was sent, wait until transmission is finished before sending the next character), then clear TI and send the character.

```
WAIT:   JNB TI, WAIT    ; Check TI until set
        CLR TI          ; Clear the flag
        MOV SBUF, A     ; Send character
```

## Multiprocessor Communication

- Serial Communication Modes 2 and 3 allow one "Master" 8051 to control several "Slaves" 8051.
- The serial port can be programmed to generate an interrupt (RI) if the 9th data bit = 1 by setting the **SM2** bit in SCON.
- The TXD outputs of the slaves are tied together and to the RXD input of the master. The RXD inputs of the slaves are tied together and to the TXD output of the master.
- Each slave is assigned an address. Address bytes transmitted by the master have the 9th bit = 1 & data bytes have it = 0.

## Multiprocessor Communication (Contd.)

- When the master transmits an address byte, all the slaves are interrupted. The slaves then check to see if they are being addressed or not.
- The addressed slave will clear its **SM2** bit and prepare to receive the data bytes that follows and the slaves that weren't addressed leave their **SM2** bits set and go about their business, ignoring the incoming data bytes. They will be interrupted again when the next address byte is transmitted by the master processor.

## Baud Rates

- Baud rate is also affected by a bit in the PCON register. PCON.7 is SMOD bit. If SMOD = 1, baud rate will be doubled in modes 1, 2 and 3.
- Mode 2 baud rate is the 1/64$^{th}$ the oscillator frequency (SMOD = 0) and can be doubled to 1/32$^{nd}$ the oscillator frequency (SMOD = 1).
- PCON is not bit-addressable, setting SMOD without altering the other bits requires a "read-modify-write" operation as follows:

```
MOV A, PCON    ; Get current value of PCON
SETB ACC.7     ; Set SMOD
MOV PCON, A    ; Write value back to PCON
```

## Using Timer 1 as Baud Rate Clock

- Usually the timer is used in auto-reload mode and TH1 is loaded with a proper reload value.
- Formula for the baud rate in modes 1 and 3 is

**Baud Rate = Timer 1 Overflow Rate / 32**

e.g., For 1200 baud

**1200 = Timer 1 Overflow Rate / 32**

**Timer 1 Overflow Rate = 38400 Hz**

- Timer must overflow at a rate of 38.4 kHz and the timer is clocked at a rate of 1000 kHz (1 MHz), overflow required every 1000/38.4 = 26.04 clocks, so

**MOV TH1, # -26**

- Due to rounding, there is a slight error in the resulting baud rate. Up to 5% is tolerable using asynchronous communications. Exact baud rates are possible using an 11.059 MHz crystal (Table 5-3).

## Initializing the Serial Port

To initialize the serial port to operate as an 8-bit UART at 2400 baud.

```
ORG  0000H
MOV  SCON,#52H  ;serial port mode 1
MOV  TMOD,#20H  ;timer 1, mode 2
MOV  TH1, #-13  ;reload count for 2400 baud
SETB TR1        ;start timer 1
END
```

## Initializing the Serial Port

**SCON:**

| SMO | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

(52H)

• (SM0/SM1=0/1) sets serial port into 8-bit UART, (REN=1) enables the serial port to receive characters, (TI=1) allows transmission of the first character by indicating that the transmit buffer is empty.

**TMOD:**

| GATE | C/$\overline{T}$ | M1 | M0 | GATE | C/$\overline{T}$ | M1 | M0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

(20H)

• (M1/M0=1/0) puts Timer 1 into auto-reload mode.

**TCON:**

| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

(40H)

• (TR1=1) turns ON Timer 1.

**TH1:**

| Tb7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

(F3H)

• Loads the re-load value –13 or F3H in the TH1 register.

Chapter 6

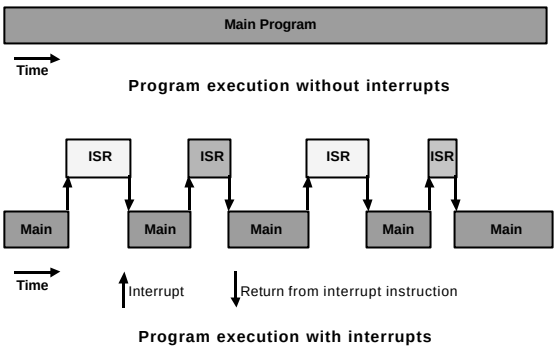
# Interrupts

(I. Scott Mackenzie)

# Interrupts

- An interrupt is the occurrence of an event that causes a temporary suspension of a program while the condition is serviced by another program.
- It is like a sub-routine. CPU cannot execute more than one instruction at a time; but it can temporarily suspend execution of one program, execute another, then return to the first program.
- Difference in interrupt and subroutine is that in an interrupt-driven system, the interruption occur asynchronously with the main program, and it is not known when the main program will be interrupted.
- Program that deals with the interrupt is called as **ISR (Interrupt Service Routine).**

# Program Execution




# Interrupt System

- Five interrupt sources in order of polling (priority) sequence are:

**External Interrupt 0**
**Timer 0**
**External Interrupt 1**
**Timer 1**
**Serial Port**

- The polling sequence is fixed but each interrupt type can be programmed to one of two priority levels.
- If two interrupts of same priority occur simultaneously then polling sequence will determine which is serviced first.
- External interrupts can be programmed for edge or level sensitivity.
- Each interrupt type has a separate vector address.
- All interrupts are disabled after a system reset and enabled individually by software.

# Processing Interrupts

When an interrupt occurs and is accepted by CPU, the following actions occur:

–Current instruction's complete execution

–PC is saved on the stack

–PC is loaded with the vector address of the ISR

–ISR executes and takes action in response to interrupt

–ISR finishes with a RETI instruction

–PC is loaded with its old value from the stack

–Execution of main program continues where it left off

# IE : Interrupt Enable Register (0A8H)

| EA | ---- | ---- | ES | ET1 | EX1 | ET0 | EX0 |
|---|---|---|---|---|---|---|---|

**(1 = Enabled, 0 = Disabled)**

- EA : Global interrupt enable/ disable.
- ES : Serial port interrupt enable/ disable.
- ET1 : Timer 1 interrupt enable/ disable.
- EX1: External interrupt 1 enable/ disable.
- ET0 : Timer 0 interrupt enable/ disable.
- EX0: External interrupt 0 enable/ disable.

e.g., Timer 1 interrupt can be enabled as follows:

```
SETB EA          ; Enable global interrupt bit
SETB ET1         ; Enable Timer 1 interrupt
```

Or **MOV IE, #10001000B**

# IP: Interrupt Priority Register (0B8H)

| ----- | ----- | ----- | PS | PT1 | PX1 | PT0 | PX0 |
|---|---|---|---|---|---|---|---|

(1 = High priority, 0 = Low priority)

- PS : Priority for Serial port interrupt.
- PT1: Priority for Timer 1 interrupt.
- PX1 : Priority for External interrupt 1.
- PT0 : Priority for Timer 0 interrupt.
- PX0 : Priority for External interrupt 0.

**IP** is cleared after a system reset to place all interrupts at the lower priority level by default.

## Interrupt Vector

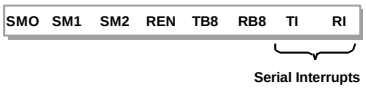
- When an interrupt is accepted, the value loaded into the PC is called the interrupt vector. It is the address of the start of the ISR for the interrupting source.
- When an interrupt is vectored, the flag that caused the interrupt is automatically cleared by hardware.
- Timer interrupts occur when the timer registers (TLx/THx) overflow and set the overflow flag (TFx).

## Interrupt Vector Addresses

| Interrupt | Flag | Vector Address | SFR & Bit Position |
|---|---|---|---|
| System Reset | RST | 0000H | |
| External 0 | IE0 | 0003H | TCON.1 |
| Timer 0 | TF0 | 000BH | TCON.5 |
| External 1 | IE1 | 0013H | TCON.3 |
| Timer 1 | TF1 | 001BH | TCON.7 |
| Serial Port | RI or TI | 0023H | SCON.0 or SCON.1 |

- Each one is 8 byte in size.

## SCON : Serial Port CONtrol Register (098H)




- **TI:** Transmit interrupt flag. Set at the end of character transmission; cleared by software.
- **RI:** Receive interrupt flag. Set at the end of character reception; cleared by software.

## External Interrupts

- External interrupt occurs as a result of a low-level or negative-edge on the $\overline{\text{INT0}}$ or $\overline{\text{INT1}}$ pin of 8051.
- Flags that generate these interrupts are bits IE0 and IE1 in TCON. These are automatically cleared when the CPU vectors to the interrupt.
- Low-level or negative-edge activated interrupts can be programmed through IT0 and IT1 bits in TCON, i.e., ITx = 0 means low-level and ITx = 1 means negative-edge triggered.

## TCON : Counter/Timer CONtrol Register (088H)


- TF1, TF0 : Overflow flags for Timer 1 and Timer 0.
- TR1, TR0 : Run control bits for Timer 1 and Timer 0.
  Set to run, reset to hold.

- IE1, IE0 : Edge flag for external interrupts 1 and 0. *
  Set by interrupt edge, cleared when interrupt is processed.
- IT1, IT0 : Type bit for external interrupts. *
  Set for falling edge interrupts, reset for 0 level interrupts.

* = not related to counter/timer operation but used to detect and initiate external interrupts.