# UNIT V 8051 Microcontroller

## Lecture 1

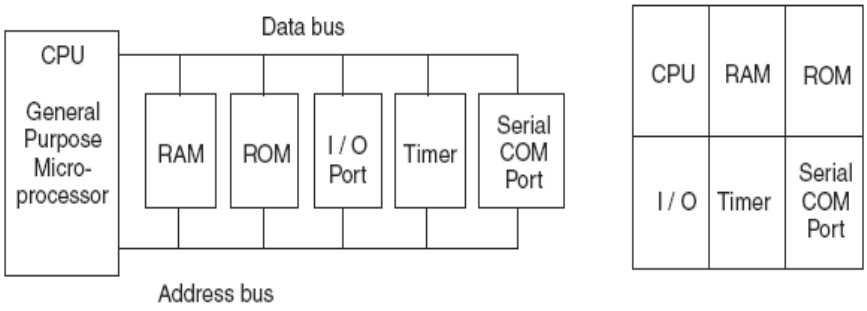
### Introduction to 8051

- **A Harvard architecture (separate instruction/data memories)**
- **single chip microcontroller (µC)**
- **developed by Intel in 1980 for use in embedded systems.**
- **today largely superseded by a vast range of faster and/or functionally enhanced 8051-compatible devices manufactured by more than 20 independent manufacturers**

The microcontroller incorporates all the features that are found in microprocessor. The microcontroller has built in ROM, RAM, Input Output ports, Serial Port, timers, interrupts and clock circuit. A microcontroller is an entire computer manufactured on a single chip. Microcontrollers are usually dedicated devices embedded within an application. For example, microcontrollers are used as engine controllers in automobiles and as exposure and focus controllers in cameras. In order to serve these applications, they have a high concentration of on-chip facilities
such as serial ports, parallel input output ports, timers, counters, interrupt control, analog-to-digital converters, random access memory, read only memory, etc. The I/O, memory, and on-chip peripherals of a microcontroller are selected depending on the specifics of the target application. Since microcontrollers are powerful digital processors, the degree of control and programmability they provide significantly enhances the effectiveness of the application. The 8051 is the first microcontroller of the MCS-51 family introduced by Intel Corporation at the end
of the 1970s. The 8051 family with its many enhanced members enjoys the largest market share, estimated to be about 40%, among the various microcontroller architectures. The microcontroller has on chip peripheral devices.

• Microcontroller (MC) may be called computer on chip since it has basic features of microprocessor with internal ROM, RAM, Parallel and serial ports within single chip. Or we can say microprocessor with memory and ports is called as microcontroller. This is widely used in washing machines, vcd player, microwave oven, robotics or in industries.
• Microcontroller can be classified on the basis of their bits processed like 8bit MC, 16bit MC.
• 8 bit microcontroller, means it can read, write and process 8 bit data. Ex. 8051 microcontroller. Basically 8 bit specifies the size of data bus. 8 bit microcontroller means 8 bit data can travel on the data bus or we can read, write process 8 bit data.

**DIFFERENCE BETWEEN MICROCONTROLLER AND MICROPROCESSOR**




(a) General-purpose Microprocessor System (b) Microcontroller

| | Microprocessors | Microcontrollers |
|---|---|---|
| 1 | It is only a general purpose computer CPU | It is a micro computer itself |
| 2 | Memory, I/O ports, timers, interrupts are not available inside the chip | All are integrated inside the microcontroller chip |
| 3 | This must have many additional digital components to perform its operation | Can function as a micro computer without any additional components. |
| 4 | Systems become bulkier and expensive. | Make the system simple, economic and compact |
| 5 | Not capable for handling Boolean functions | Handling Boolean functions |
| 6 | Higher accessing time required | Low accessing time |
| 7 | Very few pins are programmable | Most of the pins are programmable |
| 8 | Very few number of bit handling instructions | Many bit handling instructions |
| 9 | Widely Used in modern PC and laptops | widely in small control systems |
| E.g. | INTEL 8086,INTEL Pentium series | INTEL8051,89960,PIC16F877 |

**The features of Intel 8051 family include:**

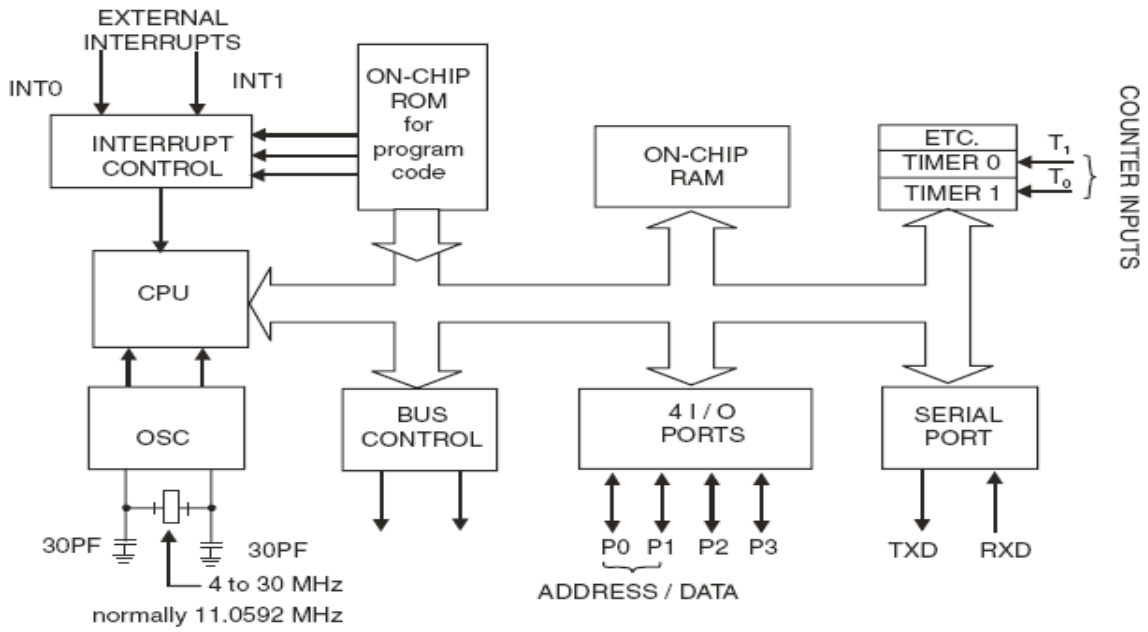
1. 8-bit CPU optimized for control applications.
2. Extensive Boolean processing (single-bit logic) capabilities.
3. 128 bytes of on-chip RAM (256 bytes for 8052).
4. 4K bytes of on-chip ROM (8K bytes for 8052).
5. 32 bidirectional and individually addressable I/O lines.
6. Two 16-bit timer/counters (three for 8052).
7. Full duplex UART.
8. Two-level priority interrupts.
9. 5 interrupt sources including 2 external interrupts and 3 internal interrupts (UART and 2 timer/counters); 6 interrupts sources for 8052.
10. 64K program memory address space.
11. 64K data memory address space.
12. On-chip clock oscillator can operate up to 12 MHz.
13. Maximum system memory up to 128KB plus internal data memory

## UNIT V 8051 Microcontroller

## Lecture 2

**Architecture of 8051 microcontroller (RGPV DEC 2014)**

It is 8-bit microcontroller, means MC 8051 can Read, Write and Process 8 bit data. This is mostly used microcontroller in the robotics, home appliances like mp3 player, washing machines, electronic iron and industries. Mostly used blocks in the architecture of 8051 are as follows




**Block Diagram of 8051**

1) A microcontroller basically contains one or more following components:

- Central processing unit(CPU)
- Random Access Memory)(RAM)
- Read Only Memory(ROM)
- Input/output ports
- Timers and Counters
- Interrupt Controls
- Analog to digital converters
- Digital analog converters
- Serial interfacing ports
- Oscillatory circuits

2) A microcontroller internally consists of all features required for a computing system and functions as a computer without adding any external digital parts in it.

3) Most of the pins in the microcontroller chip can be made programmable by the user.

4) A microcontroller has many bit handling instructions that can be easily understood by the programmer.

5) A microcontroller is capable of handling Boolean functions.

6) Higher speed and performance.

7) On-chip ROM structure in a microcontroller provides better firmware security.

8 ) Easy to design with low cost and small size.

- **CPU**

CPU is the brain of a microcontroller .CPU is responsible for fetching the instruction, decodes it, then finally executed. CPU connects every part of a microcontroller into a single system. The primary function of CPU is fetching and decoding instructions. Instruction fetched from program memory must be decoded by the CPU.

- **Memory**

The function of memory in a microcontroller is same as microprocessor. It is used to store data and program. A microcontroller usually has a certain amount of RAM and ROM (EEPROM, EPROM, etc) or flash memories for storing program source codes.

- **Parallel input/output ports**

Parallel input/output ports are mainly used to drive/interface various devices such as LCD'S, LED'S, printers, memories, etc to a microcontroller.

- **Serial ports**

Serial ports provide various serial interfaces between microcontroller and other peripherals like parallel ports.

- **Timers/counters**

This is the one of the useful function of a microcontroller. A microcontroller may have more than one timer and counters. The timers and counters provide all timing and counting functions inside the microcontroller. The major operations of this section are perform clock functions, modulations, pulse

generations, frequency measuring, making oscillations, etc. This also can be used for counting external pulses.

- **Analog to Digital Converter (ADC)**:-ADC converters are used for converting the analog signal to digital form. The input signal in this converter should be in analog form (e.g. sensor output) and the output from this unit is in digital form. The digital output can be use for various digital applications (e.g. measurement devices).

- **Digital to Analog Converter (DAC)**:-DAC perform reversal operation of ADC conversion.DAC convert the digital signal into analog format. It usually used for controlling analog devices like DC motors, various drives, etc.

- **Interrupt control**:-The interrupt control used for providing interrupt (delay) for a working program .The interrupt may be external (activated by using interrupt pin) or internal (by using interrupt instruction during programming).

- **Special functioning block**:-Some microcontrollers used only for some special applications (e.g. space systems and robotics) these controllers containing additional ports to perform such special operations. This considered as special functioning block.

**Advantages of Microcontrollers:-**The main advantages of microcontrollers are given.

a) Microcontrollers act as a microcomputer without any digital parts.

b) As the higher integration inside microcontroller reduce cost and size of the system.

c) Usage of microcontroller is simple, easy for troubleshoot and system maintaining.

d) Most of the pins are programmable by the user for performing different functions.

e) Easily interface additional RAM, ROM,I/O ports.

f) Low time required for performing operations.

**Disadvantages of Microcontrollers**

a) Microcontrollers have got more complex architecture than that of microprocessors.

b) Only perform limited number of executions simultaneously.

c) Mostly used in micro-equipments.

d) Cannot interface high power devices directly.

**Applications:-**Microcontrollers are widely used in modern electronics equipments. Some basic applications of microcontroller is given below.

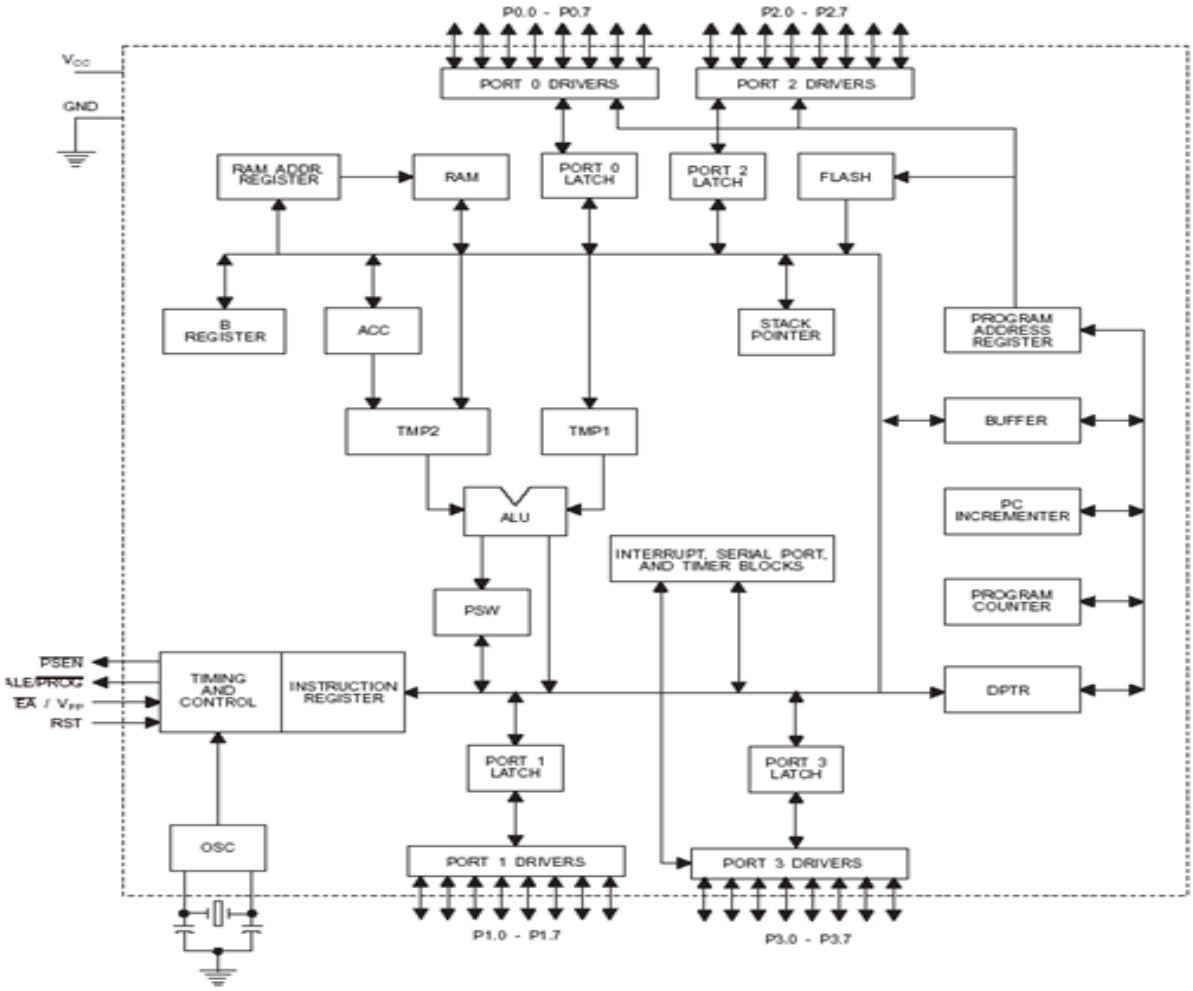
a) Used in biomedical instruments.

b) Widely used in communication systems.

c) Used as peripheral controller in PC.

d) Used in robotics.

e) Used in automobile fields




**Architecture of 8051**

| S.NO | RGPV QUESTIONS | Year | Marks |
|---|---|---|---|
| Q.1 | What is microcontroller?Explain the architecture of 8051 microcontroller | DEC 2014 | 7 |

| S.NO | RGPV QUESTIONS | Year | Marks |
|---|---|---|---|
| Q.1 | Draw & discuss the internal architecture of 8051. | JUNE 2013 | 10 |

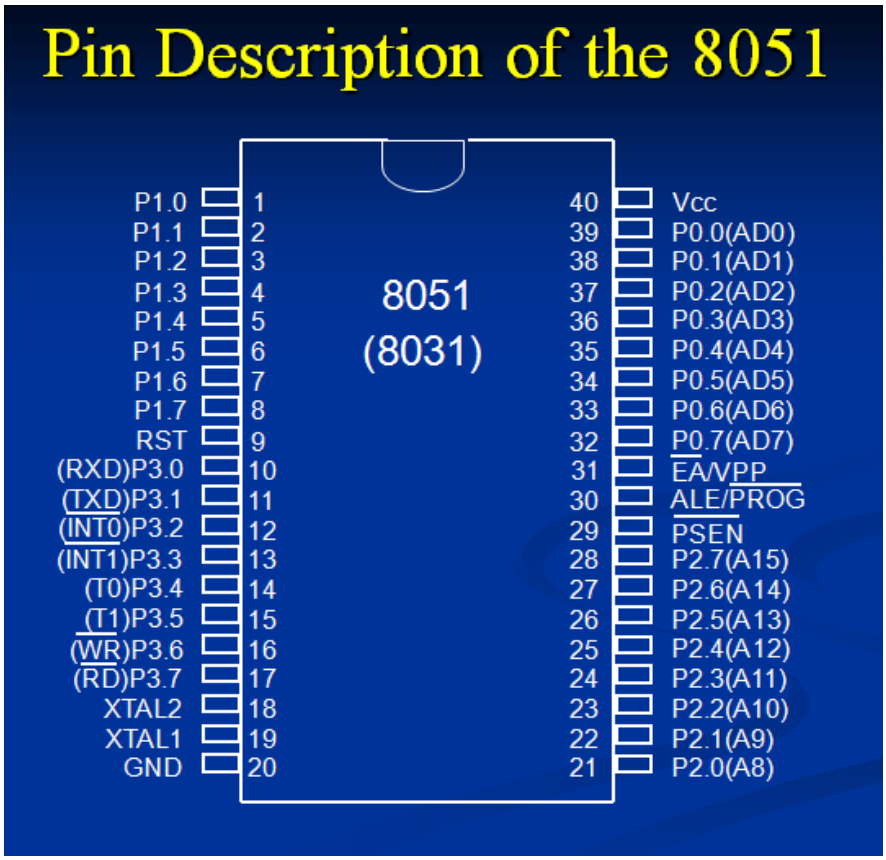
| S.NO | RGPV QUESTIONS | Year | Marks |
|---|---|---|---|
| Q.1 | What is microcontroller?Explain the architecture of 8051 microcontroller | JUNE 2011 | 10 |

| S.NO | RGPV QUESTIONS | Year | Marks |
|---|---|---|---|
| Q.1 | Discuss the register set & Flags of 8051 microcontroller. | DEC 2014 | 7 |

## UNIT V 8051 Microcontroller

### Lecture 3

**Pin Description of 8051**




- Vcc（pin 40）：
  - Vcc provides supply voltage to the chip.
  - The voltage source is +5V.
- GND（pin 20）：ground
- XTAL1 and XTAL2（pins 19,18）：
  - These 2 pins provide external clock.
  - Way 1：using a quartz crystal oscillator
  - Way 2：using a TTL oscillator
  - Example 4-1 shows the relationship between XTAL and the machine cycle.
- RST（pin 9）：reset
  - It is an input pin and is active high（normally low）.
    - The high pulse must be high at least 2 machine cycles.

- It is a power-on reset.
  - Upon applying a high pulse to RST, the microcontroller will reset and all values in registers will be lost.
  - Reset values of some 8051 registers
- Way 1：Power-on reset circuit
- Way 2：Power-on reset with debounce

- /EA（pin 31）：external access
  - There is no on-chip ROM in 8031 and 8032 .
  - The /EA pin is connected to GND to indicate the code is stored externally.
  - /PSEN ＆ ALE are used for external ROM.
  - For 8051, /EA pin is connected to Vcc.
  - “/” means active low.
- /PSEN（pin 29）：program store enable
  - This is an output pin and is connected to the OE pin of the ROM.
- ALE（pin 30）：address latch enable
  - It is an output pin and is active high.
  - 8051 port 0 provides both address and data.
  - The ALE pin is used for de-multiplexing the address and data by connecting to the G pin of the 74LS373 latch.
- I/O port pins
  - The four ports P0, P1, P2, and P3.
  - Each port uses 8 pins.
  - All I/O pins are bi-directional.
- Using a quartz crystal oscillator XTAL1 pin
- We can observe the frequency on the XTAL2 pin.

- The 8051 has four I/O ports
  - Port 0 （pins 32-39）：P0（P0.0～P0.7）
  - Port 1（pins 1-8）　　：P1（P1.0～P1.7）
  - Port 2（pins 21-28）：P2（P2.0～P2.7）
  - Port 3（pins 10-17）：P3（P3.0～P3.7）
  - Each port has 8 pins.
    - Named P0.X （X=0,1,...,7）, P1.X, P2.X, P3.X
    - Ex：P0.0 is the bit 0（LSB）of P0
    - Ex：P0.7 is the bit 7（MSB）of P0
    - These 8 bits form a byte.
- Each port can be used as input or output (bi-direction).

# UNIT V 8051 Microcontroller

## Lecture 4

**Memory organization**

The 8051 has separate address spaces for program storage and data storage. Depending on the type of instruction, the same address can refer to two logically and physically different memory locations.

**Program Storage**

- After reset, the MCS-51 starts fetching instructions from 0000H.
  - This can be either on-chip or external depending on the value of the EA input pin.
    - If EA* is low, then the program memory is external.
    - If EA* is high, then addresses from 0000 to 0FFF will refer to on-chip memory and addresses 1000 up to FFFF refer to external memory.

**Access to External Memory**

- Port 0 acts as a multiplexed address/data bus. Sending the low byte of the program counter (PCL) as an address.
- Port 2 sends the program counter high byte (PCH) directly to the external memory.
- The signal ALE operates as in the 8051 to allow an external latch to store the PCL byte while the multiplexed bus is made ready to receive the code byte from the external memory.
- Port 0 then switches function and becomes the data bus receiving the byte from memory.

**Data Storage**

- The 8051 has 256 bytes of RAM on-chip.
  - The lower 128 bytes are intended for internal data storage.
  - The upper 128 bytes are the Special Function Registers (SFR).
- The lower 128 bytes are not to be used as standard RAM.
  - Internally 8051's registers default to stack area, and other features. [00-7FH]
- The lowest 32 bytes of the on-chip RAM form 4 banks of 8 registers each.
- Only one of these banks can be active at any time.
- Bank is chosen by setting 2 bits in PSW
  - Default bank (at power up) is bank 0 (locations 00 – 07).
- The 8 registers in any active bank are referred to as R0 through R7
- Given that each register has a specific address; it can be accessed directly using that address even if its bank is not the active one.
- The next 16 bytes – locations 20H to 2FH – form a block that can be addressed as either bytes or individual bits.
  - The bytes have addresses 20H to 2FH.
  - The bits have addresses 00H to 7FH.
  - Specific instructions are used for accessing the bits
- Locations 30H to 7FH are general purpose RAM.
-

## UNIT V 8051 Microcontroller

## Lecture 5

**Timers/Counters**

The 8051 has 2 timers/counters:

1. timer/counter 0
2. timer/counter 1

They can be used as

1. The **timer** to generate time delay.
   - The clock source is the **internal** crystal frequency of the 8051.
2. An event **counter**.
   - **External pulse input** from input pin to count the number of events on registers.

These clock pulses could represent the number of people passing through an entrance, or the number of wheel rotations, or any other event that can be converted to pulses

1. Two internal Timers/Counters
2. 16-bit timer/counter
3. Timer uses system clock as source of input pulses
4. Counter uses external input pulses from port 3 (T0,T1)
5. If associated interrupt is enabled, when count overflow an interrupt is generated

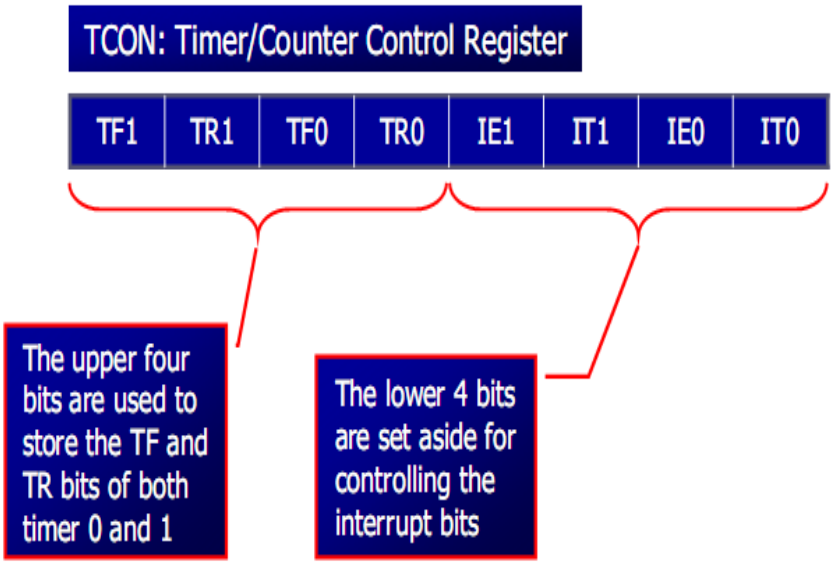
Registers

- TH0, TL0 : timer/counter register of timer 0
- TH1, TL1 : timer/counter register of timer 1
- TMOD : Mode Select register
- TCON : Control Register

Operation Modes
Mode 0
13-bit counter, an interrupt is generated when counter overflows
It takes 8192 input pulses to generate the next interrupt
Mode 1
16-bit counter, similar to mode 0, but take 65536 input pulses
Mode 2
8-bit reload
TLi operates as timer/counter
THi store a number and reload to TLi when overflows
Mode 3
Timer 1 is inactive, hold count value
TL0 and TH0 operate as two separate 8-bit timer/counter

TL0 control by timer 0 control bits
TH0 operate as timer driven by system clock, prescaled by 12 and cause timer 1 interrupt overflows

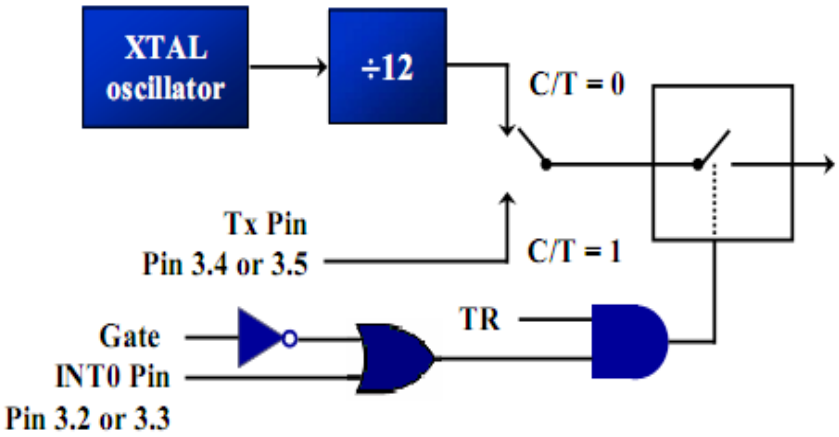
TCON register




Timer control and Flag bits

1. **TR** (Timer run control bit)
   - TR0 for Timer/counter 0; TR1 for Timer/counter 1.
   - TR is set by programmer to turn timer/counter on/off.
     - CLR TR0 : off (stop)
     - SETB TR0 : on (start)
2. **TF** (timer flag bit)
   - TF0 for timer/counter 0; TF1 for timer/counter 1.
   - TF is like a carry. Originally TF=0. When TH-TL rolls over from FFFFh to 0000, the 8051 sets TF to 1. TF should be cleared by software in polling method.
     - TF=0 : not reach
     - TF=1: reach

TMOD Register


Gate

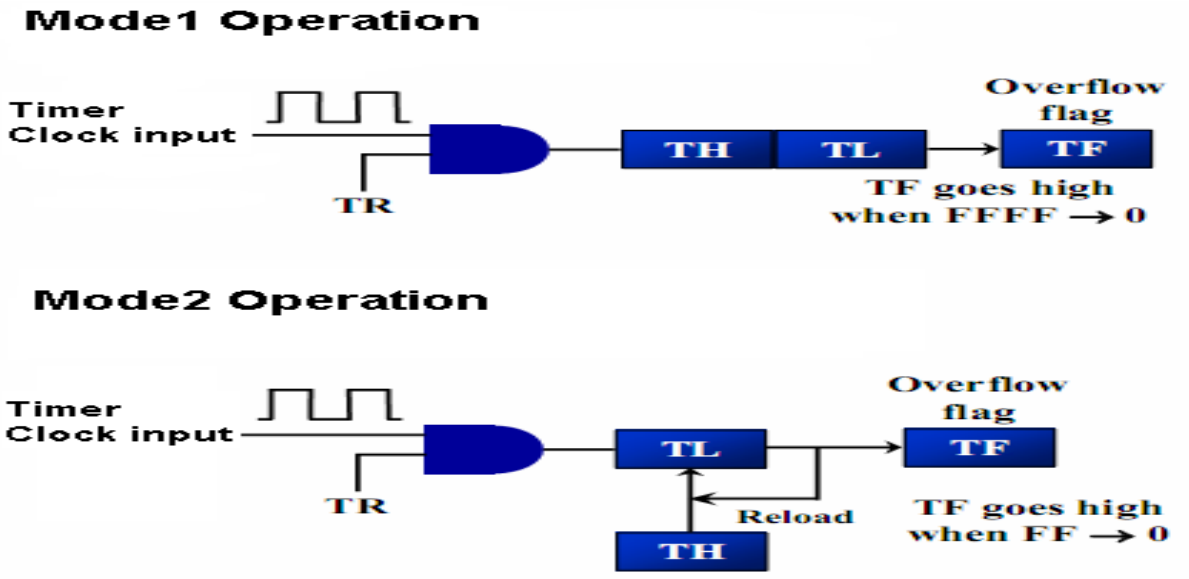
- Every timer has a mean of starting and stopping.
  - GATE=0
    - **Internal** control
    - The start and stop of the timer are controlled by way of **software.**
    - Set/clear the TR for start/stop timer.

  SETB TR0
  CLR TR0
  - GATE=1
    - **External** control
    - The hardware way of starting and stopping the timer by **software** and **an external source**.
    - Timer/counter is enabled only while the INT pin is high and the TR control pin is set (TR).

Mode selection Bits in the TMOD

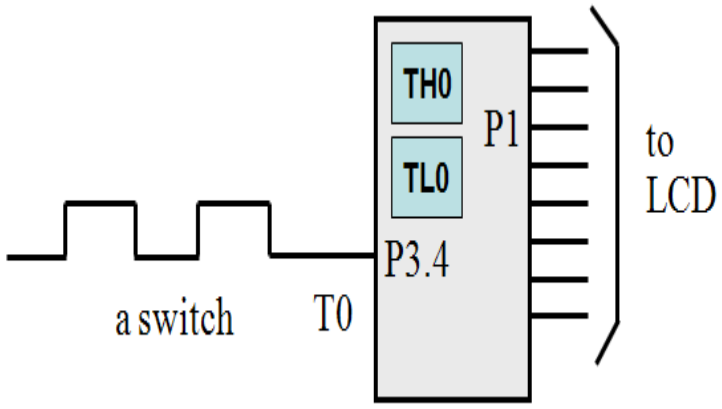
| M1 | M0 | Mode | Operation | |
|---|---|---|---|---|
| 0 | 0 | **0** | 1**3-bit timer** mode | 8-bit THx + 5-bit TLx (x= 0 or 1) |
| 0 | 1 | **1** | **16-bit timer** mode | 8-bit THx + 8-bit TLx |

| | | | |
|---|---|---|---|
| 1 | 0 | **2** | **8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded into TLx each time it overflows. |
| 1 | 1 | **3** | **Split timer** mode |




Counter

- Count the number of events
  - Show the number of events on registers
  - External input from T0 input pin (P3.4) for Counter 0
  - External input from T1 input pin (P3.5) for Counter 1
  - **External input** from Tx input pin.



## UNIT V 8051 Microcontroller

## Lecture 6

**Interrupts of 8051**

**Types of Interrupts**

Interrupts can be broadly classified as –

**Hardware Interrupts:-** These are interrupts caused by the connected devices. –

**Software Interrupts:-** These are interrupts deliberately introduced by software instructions to generate user defined exceptions-

**Trap** :-These are interrupts used by the processor alone to detect any exception such as divide by zero

Depending on the service the interrupts also can be classified as

Fixed interrupt

• Address of the ISR built into microprocessor, cannot be changed

• Either ISR stored at address or a jump to actual ISR stored if not enough bytes available

Vectored interrupt

• Peripheral must provide the address of the ISR

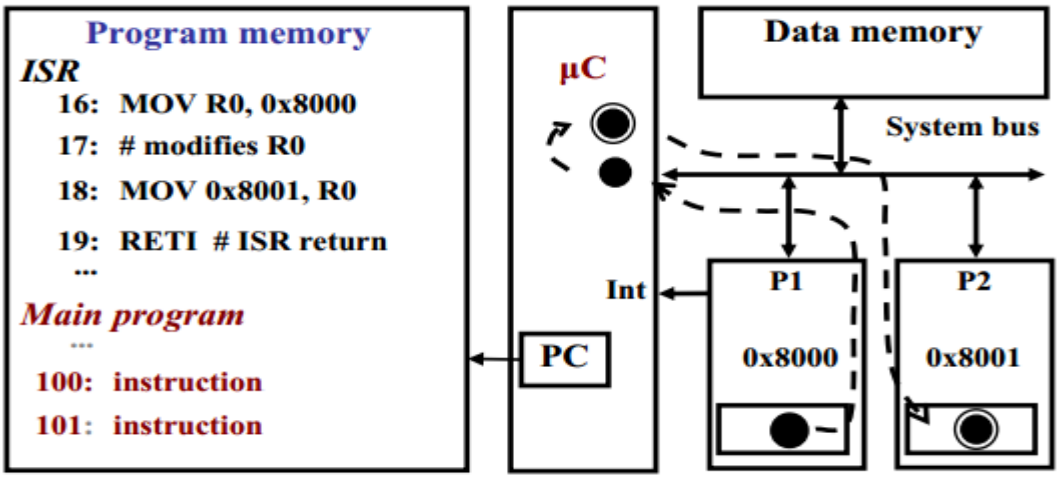
• Common when microprocessor has multiple peripherals connected by a system bus

Maskable vs. Non-maskable interrupts –

**Maskable**: programmer can set bit that causes processor to ignore interrupt • This is important when the processor is executing a time-critical code

**Non-maskable**: a separate interrupt pin that can't be masked • Typically reserved for drastic situations, like power failure requiring immediate backup of data to non-volatile memory

Example: Interrupt Driven Data Transfer (Fixed Interrupt)




The Interrupt Driven Data Transfer

The 8051 has 5 interrupt sources: 2 external interrupts, 2 timer interrupts, and the serial port interrupt.

These interrupts occur because of 1. timers overflowing 2. receiving character via the serial port 3. transmitting character via the serial port 4. Two “external events”

INT0 : External Interrupt 0 INT0 : External Interrupt 1 TF0: Timer 0 Interrupt TF1: Timer 1 Interrupt RI,TI: Serial Port Receive/Transmit Interrupt The service routine for each interrupt begins at a fixed location (fixed address interrupts). Only the Program Counter (PC) is automatically pushed onto the stack, not the Processor Status Word (which includes the contents of the accumulator and flag register) or any other register. Having only the PC automatically saved allows the programmer to decide how much time should be spent saving other registers. This enhances the interrupt response time, albeit at the expense of increasing the programmer’s burden of responsibility. As a result, many interrupt functions that are typical in control applications toggling a port pin for example, or reloading a timer, or unloading a serial buffer can often be completed in less time than it takes other architectures to complete.

| Interrupt Number | Interrupt Vector Address | Description |
|---|---|---|
| 0 | 0003h | EXTERNAL 0 |
| 1 | 000Bh | TIMER/COUNTER 0 |
| 2 | 0013h | EXTERNAL 1 |
| 3 | 001Bh | TIMER/COUNTER 1 |
| 4 | 0023h | SERIAL PORT |

# UNIT V 8051 Microcontroller

## Lecture 7

**Addressing modes of 8051 (RGPV DEC 2014)**

When instructions operate on data, the question arises: "Where are the data?" The answer to this question lies in the 8051's "addressing modes." There are several possible addressing modes and there are several possible answers to the question, such as "in byte 2 of the instruction," "in register R4," "in direct address 35H." or perhaps "in external data memory at the address contained in the data pointer."

Addressing modes are an integral part of each computers instruction set. They allow specifying the source or destination of data in different ways, depending on the programming situation. In this section, we'll examine all the 8051 addressing modes. There are eight modes available:

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

**Register Addressing**

The 8051 programmer has access to eight "working registers," numbered R0 through R7. Instructions using register addressing are encoded using the three least-significant bits of the instruction opcode to specify a register within this logical address space. Thus, a function code and operand address can be combined to form a short (1-byte) instruction.
The 8051 assembly language indicates register addressing with the symbol Rn where n is from 0 to 7. For example, to add the contents of Register 7 to the accumulator, the following instruction is used

ADD A, R7

and the opcode is 00101111B. The upper five bits, 00101, indicate the instruction, and the lower three bits, 111, the register.

There are four "banks" of working registers, but only one is active at a time. Physically, the register banks occupy the first 32 bytes of on-chip data RAM (addresses 00H-1FH) with PSW bits 4 and 3 determining the active bank. A hardware reset enables bank 0, but a different bank is selected by modifiying PSW bits 4 and 3 accordingly. For example, the instruction

MOV PSW, #00011000B

activates register bank 3 by setting the register bank, select bits bits (RS1 and RSO) in PSW bit position 4 and 3.

Some instructions are specific to a certain register. Such as the accumulator, data pointer, etc. so addressing bits are not needed. The opcode itself indicates the register. These "register-specific" instructions refer to the accumulator as "A," the data pointer as "DPTR" the program counter as "PC," the carry flag as "C," and the accumulator-B register regist pair as "AB."For example,

INC DPTR

is a 1-byte instruction that adds to the 16-bit data-pointer.

**Direct Addressing**

Direct addressing can access any on-chip variable or hardware register. An additional byte is appended to the opcode specifying the location to be used. Depending on the high-order bit of the direct address, one of two on-chip memory spaces is selected. When bit 7 = 0, the direct address is between 0 and 127 (00H-7FH) and the 128 low-order on-chip RAM locations are referenced. All I/O ports and special function, control, or status registers, however, are assigned addresses between 128 and 255 80H—FFH). When the direct address byte is between these limits (bit 7 = 1), the corresponding special function register is accessed. For example, Ports 0 and 1 are assigned direct addresses 80H and 90H, respectively. It is usually not necessary to know the addresses of these registers; the assembler allows for and understands the mnemonic abbreviations "P0" for Port 0, "TMOD" for timer mode register, etc.). Some assemblers, such as Intel's ASM51, automatically include the definition of predefined symbols. Other assemblers may use a separate source file containing the definitions. As an example of direct addressing, the instruction

MOV P1 , A

transfers the content of the accumulator to Port 1. The direct address of Port 1 (9014) is determined by the assembler and inserted as byte 2 of the instruction. The source of the data, the accumulator, is specified implicitly in the opcode. The complete encoding of this instruction is

10001001 – 1st byte (opcode)
10010000 — 2nd byte (address of P1)

**Indirect Addressing**

How is a variable identified if its address is determined, computed, or modified while a program is running? This situation arises when manipulating sequential memory locations, indexed entries within tables in RAM, multiple-precision numbers, or character strings. Register or direct addressing cannot be used, since they require operand addresses to be known at assemble-time.

The 8051 solution is indirect addressing. RO and R1 may operate as "pointer" registers—their contents indicating an address in internal RAM where data are written or read. The least-significant bit of the instruction opcode determines which register (RO or RI) is used as the pointer.

In 8051 assembly language, indirect addressing is represented by a commercial "at" sign (@) preceding RO or R1. As an example, if R1 contains 40H and internal memory address 40H contains 55H, the instruction

MOV A, @R1

moves 55H into the accumulator.

Indirect addressing is essential when stepping through sequential memory locations. For example. the following instruction sequence clears internal RAM from address 60H to 7F1-1:

MOV R0, #60H

LOOP: MOV @R0, #0

INC R0

CJNE R0, #80H, LOOP

(continue)

The first instruction initializes R0 with the starting address of the block of memory; the second instruction uses indirect addressing to move 00H to the location pointed at by R0; the third instruction increments the pointer (R0) to the next address; and the last instruction tests the pointer to see if the end of the block has been reached. The test uses 80H, rather than 7FH, because the increment occurs after the indirect move. This ensures the final location (7FH) is written to before terminating.

**Immediate Addressing**

When a source operand is a constant rather than a variable (i.e., the instruction uses a value known at assemble-time), then the constant can be incorporated into the instruction as a byte of "immediate" data. An additional instruction byte contains the value.

In assembly language, immediate operands are preceded by a number sign (#). The operand may be a numeric constant, a symbolic variable, or an arithmetic expression using constants, symbols, and operators. The assembler computes the value and substitutes the immediate data into the instruction. For example, the instruction

MOV A, #12

loads the value 12 (0CH) into the accumulator. (It is assumed the constant "12" is in decimal notation, since it is not followed by "H.")
With one exception, all instructions using immediate addressing use an 8-bit data constant for the immediate data. When initializing the data pointer, a 16-bit constant is required. For example,

MOV DPTR, #8000H

is a 3-byte instruction that loads the 16-bit constant 8000H into the data pointer.

**Relative Addressing**

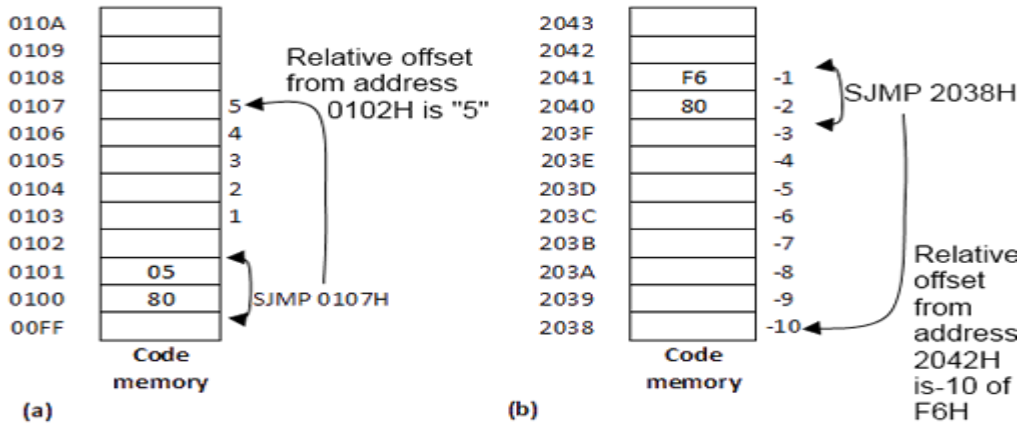
Relative addressing is used only with certain jump instructions. A relative address (or offset) is an 8-bit signed value, which is added to the program counter to form the address of the next instruction executed. Since an 8-bit signed offset is used, the range for jumping is —128 to +127 locations, The relative offset is appended to the instruction as an additional byte.

Prior to the addition, the program counter is incremented to the address following jump instruction; thus, the new address is relative to the next instruction, not the address of the jump instruction.

Normally, this detail is of no concern to the programmer, since jump destinations are usually specified as labels and the assembler determines the relative offset accordingly. For example, if the label THERE represents an instruction at location 1040H, and the instruction

SJMP THERE

is in memory at locations 1000H and 1001H, the assembler will assign a relative offset of 3EH as byte 2 of the instruction (1002H + 3EH = 1040H).



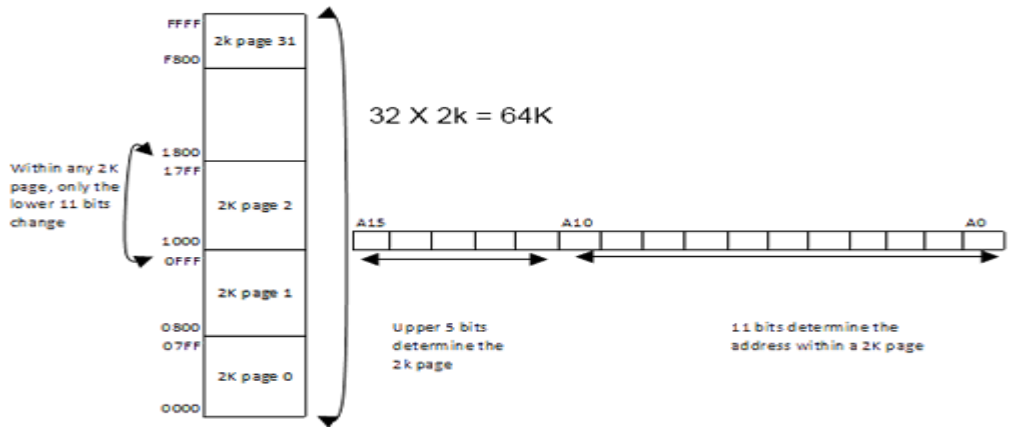

Relative addressing offers the advantage of providing position-independent code (since"absoulute" addresses are not used). but the disadvantage that the jump destinations are limited in range.

**Absolute Addressing**

absolute addressing is used only with the ACALL and AJMP instructions. These 2-byte instructions allow within the current 2K page of code memory by providing the 11 least significant bits of the destination address in the opcode(A10-A8) and byte 2 of the instruction (A7 - A0).

The upper five bits of the destination address are the current upper five bits in the program counter, so the instruction following the branch instruction and the destination for the branch instruction must be with in the same 2K page,since A15 - A11 do not change. For example, if the label THERE represents an instruction at address 0F46H, and the instruction

AJMP THERE




is the memory location 0900H and 0901H. the assembler will encode the instruction as

11100001 - 1st byte (A10 - A8 + opcode)
<u>01000110</u> - 2nd byte (A7 - A0)

The underlined bits are the low order 11 bits of the destination address, 0F46H = 00001<u>11101000110B</u>. The upper five bits in the program counter will not change when the instruction executes. Note that both the AJMP instruction and the destination are with in the 2K page bounded by 0800H and 0FFFH, and therefore have the upper five address bits in common.

Absolute addressing offers the advantage of short (2-byte) instructions, but has the disadvantage of limiting the range for the destination and providing position dependent code.

**Long Addressing**

Long addressing is used only with the LCALL and LJMP instructions. These 3-byte instructions The advantage include a full 16-bit destination address as bytes 2 and 3 of the instruction.

The advantage is that the full 64K code space may be used, but the disadvantage is that the instructions are three bytes long and are position-dependent. Position-dependence is a disadvantage because the

program cannot execute at different addresses. If, for example, a program begins at 2000H and an instruction such as LJMP 2040H appears, then the program cannot be moved to, say, 4000H. The LJMP instruction would still jump to 2040H, which is not the correct location after the program has been moved.

**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 instrution. Jump tables or lookup tables are easily created using indexed addressing

| S.NO | RGPV QUESTIONS | Year | Marks |
|---|---|---|---|
| Q.1 | What are the various addressing modes of 8051? | DEC 2014 | 7 |

# UNIT V 8051 Microcontroller

## Lecture 8

### 8051 Instruction set (RGPV DEC 2014)

### Introduction

The process of writing program for the microcontroller mainly consists of giving instructions (commands) in the specific order in which they should be executed in order to carry out a specific task. As electronics cannot “understand” what for example an instruction “if the push button is pressed- turn the light on” means, then a certain number of simpler and precisely defined orders that decoder can recognise must be used. All commands are known as INSTRUCTION SET. All microcontrollers compatibile with the 8051 have in total of 255 instructions, i.e. 255 different words available for program writing.

At first sight, it is imposing number of odd signs that must be known by heart. However, It is not so complicated as it looks like. Many instructions are considered to be “different”, even though they perform the same operation, so there are only 111 truly different commands. For example: ADD A,R0, ADD A,R1, ... ADD A,R7 are instructions that perform the same operation (additon of the accumulator and register). Since there are 8 such registers, each instruction is counted separately. Taking into account that all instructions perform only 53 operations (addition, subtraction, copy etc.) and most of them are rarely used in practice, there are actually 20-30 abbreviations to be learned, which is acceptable.

### 3.1 Types of instructions

Depending on operation they perform, all instructions are divided in several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions
- Logic Instructions
- Bit-oriented Instructions

The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.). Mnemonics are abbreviations of the name of operation being executed. For example:

- `INC R1` - Means: Increment register R1 (increment register R1);
- `LJMP LAB5` - Means: Long Jump LAB5 (long jump to the address marked as LAB5);
- `JNZ LOOP` - Means: Jump if Not Zero LOOP (if the number in the accumulator is not 0, jump to the address marked as LOOP);

The other part of instruction, called OPERAND is separated from mnemonic by at least one whitespace and defines data being processed by instructions. Some of the instructions have no operand, while some of them have one, two or three. If there is more than one operand in an instruction, they are separated by a comma. For example:

- `RET` - return from a subroutine;

- `JZ TEMP` - if the number in the accumulator is not 0, jump to the address marked as TEMP;
- `ADD A,R3` - add R3 and accumulator;
- `CJNE A,#20,LOOP` - compare accumulator with 20. If they are not equal, jump to the address marked as LOOP;

***Arithmetic instructions***

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:

`ADD A,R1` - The result of addition (A+R1) will be stored in the accumulator.

| ARITHMETIC INSTRUCTIONS | | | |
|---|---|---|---|
| **Mnemonic** | **Description** | **Byte** | **Cycle** |
| ADD A,Rn | Adds the register to the accumulator | 1 | 1 |
| ADD A,direct | Adds the direct byte to the accumulator | 2 | 2 |
| ADD A,@Ri | Adds the indirect RAM to the accumulator | 1 | 2 |
| ADD A,#data | Adds the immediate data to the accumulator | 2 | 2 |
| ADDC A,Rn | Adds the register to the accumulator with a carry flag | 1 | 1 |
| ADDC A,direct | Adds the direct byte to the accumulator with a carry flag | 2 | 2 |
| ADDC A,@Ri | Adds the indirect RAM to the accumulator with a carry flag | 1 | 2 |
| ADDC A,#data | Adds the immediate data to the accumulator with a carry flag | 2 | 2 |
| SUBB A,Rn | Subtracts the register from the accumulator with a borrow | 1 | 1 |
| SUBB A,direct | Subtracts the direct byte from the accumulator with a borrow | 2 | 2 |
| SUBB A,@Ri | Subtracts the indirect RAM from the accumulator with a borrow | 1 | 2 |
| SUBB A,#data | Subtracts the immediate data from the accumulator | 2 | 2 |

| | | | |
|---|---|---|---|
| | with a borrow | | |
| INC A | Increments the accumulator by 1 | 1 | 1 |
| INC Rn | Increments the register by 1 | 1 | 2 |
| INC Rx | Increments the direct byte by 1 | 2 | 3 |
| INC @Ri | Increments the indirect RAM by 1 | 1 | 3 |
| DEC A | Decrements the accumulator by 1 | 1 | 1 |
| DEC Rn | Decrements the register by 1 | 1 | 1 |
| DEC Rx | Decrements the direct byte by 1 | 1 | 2 |
| DEC @Ri | Decrements the indirect RAM by 1 | 2 | 3 |
| INC DPTR | Increments the Data Pointer by 1 | 1 | 3 |
| MUL AB | Multiplies A and B | 1 | 5 |
| DIV AB | Divides A by B | 1 | 5 |
| DA A | Decimal adjustment of the accumulator according to BCD code | 1 | 1 |

***Branch Instructions***

There are two kinds of branch instructions:

Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed.

Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

| BRANCH INSTRUCTIONS | | | |
|---|---|---|---|
| **Mnemonic** | **Description** | **Byte** | **Cycle** |

| | | | |
|---|---|---|---|
| ACALL addr11 | Absolute subroutine call | 2 | 6 |
| LCALL addr16 | Long subroutine call | 3 | 6 |
| RET | Returns from subroutine | 1 | 4 |
| RETI | Returns from interrupt subroutine | 1 | 4 |
| AJMP addr11 | Absolute jump | 2 | 3 |
| LJMP addr16 | Long jump | 3 | 4 |
| SJMP rel | Short jump (from −128 to +127 locations relative to the following instruction) | 2 | 3 |
| JC rel | Jump if carry flag is set. Short jump. | 2 | 3 |
| JNC rel | Jump if carry flag is not set. Short jump. | 2 | 3 |
| JB bit,rel | Jump if direct bit is set. Short jump. | 3 | 4 |
| JBC bit,rel | Jump if direct bit is set and clears bit. Short jump. | 3 | 4 |
| JMP @A+DPTR | Jump indirect relative to the DPTR | 1 | 2 |
| JZ rel | Jump if the accumulator is zero. Short jump. | 2 | 3 |
| JNZ rel | Jump if the accumulator is not zero. Short jump. | 2 | 3 |
| CJNE A,direct,rel | Compares direct byte to the accumulator and jumps if not equal. Short jump. | 3 | 4 |
| CJNE A,#data,rel | Compares immediate data to the accumulator and jumps if not equal. Short jump. | 3 | 4 |
| CJNE Rn,#data,rel | Compares immediate data to the register and jumps if not equal. Short jump. | 3 | 4 |
| CJNE @Ri,#data,rel | Compares immediate data to indirect register and jumps if not equal. Short jump. | 3 | 4 |
| DJNZ Rn,rel | Decrements register and jumps if not 0. Short jump. | 2 | 3 |

| | | | |
|---|---|---|---|
| DJNZ Rx,rel | Decrements direct byte and jump if not 0. Short jump. | 3 | 4 |
| NOP | No operation | 1 | 1 |

***Data Transfer Instructions***

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix “X” (MOVX), the data is exchanged with external memory.

| DATA TRANSFER INSTRUCTIONS | | | |
|---|---|---|---|
| **Mnemonic** | **Description** | **Byte** | **Cycle** |
| MOV A,Rn | Moves the register to the accumulator | 1 | 1 |
| MOV A,direct | Moves the direct byte to the accumulator | 2 | 2 |
| MOV A,@Ri | Moves the indirect RAM to the accumulator | 1 | 2 |
| MOV A,#data | Moves the immediate data to the accumulator | 2 | 2 |
| MOV Rn,A | Moves the accumulator to the register | 1 | 2 |
| MOV Rn,direct | Moves the direct byte to the register | 2 | 4 |
| MOV Rn,#data | Moves the immediate data to the register | 2 | 2 |
| MOV direct,A | Moves the accumulator to the direct byte | 2 | 3 |
| MOV direct,Rn | Moves the register to the direct byte | 2 | 3 |
| MOV direct,direct | Moves the direct byte to the direct byte | 3 | 4 |
| MOV direct,@Ri | Moves the indirect RAM to the direct byte | 2 | 4 |
| MOV direct,#data | Moves the immediate data to the direct byte | 3 | 3 |
| MOV @Ri,A | Moves the accumulator to the indirect RAM | 1 | 3 |
| MOV @Ri,direct | Moves the direct byte to the indirect RAM | 2 | 5 |

| | | | |
|---|---|---|---|
| MOV @Ri,#data | Moves the immediate data to the indirect RAM | 2 | 3 |
| MOV DPTR,#data | Moves a 16-bit data to the data pointer | 3 | 3 |
| MOVC A,@A+DPTR | Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR) | 1 | 3 |
| MOVC A,@A+PC | Moves the code byte relative to the PC to the accumulator (address=A+PC) | 1 | 3 |
| MOVX A,@Ri | Moves the external RAM (8-bit address) to the accumulator | 1 | 3-10 |
| MOVX A,@DPTR | Moves the external RAM (16-bit address) to the accumulator | 1 | 3-10 |
| MOVX @Ri,A | Moves the accumulator to the external RAM (8-bit address) | 1 | 4-11 |
| MOVX @DPTR,A | Moves the accumulator to the external RAM (16-bit address) | 1 | 4-11 |
| PUSH direct | Pushes the direct byte onto the stack | 2 | 4 |
| POP direct | Pops the direct byte from the stack/td> | 2 | 3 |
| XCH A,Rn | Exchanges the register with the accumulator | 1 | 2 |
| XCH A,direct | Exchanges the direct byte with the accumulator | 2 | 3 |
| XCH A,@Ri | Exchanges the indirect RAM with the accumulator | 1 | 3 |
| XCHD A,@Ri | Exchanges the low-order nibble indirect RAM with the accumulator | 1 | 3 |

***Logic Instructions***

Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

| LOGIC INSTRUCTIONS | | | |
|---|---|---|---|
| **Mnemonic** | **Description** | **Byte** | **Cycle** |
| ANL A,Rn | AND register to accumulator | 1 | 1 |

| | | | |
|---|---|---|---|
| ANL A,direct | AND direct byte to accumulator | 2 | 2 |
| ANL A,@Ri | AND indirect RAM to accumulator | 1 | 2 |
| ANL A,#data | AND immediate data to accumulator | 2 | 2 |
| ANL direct,A | AND accumulator to direct byte | 2 | 3 |
| ANL direct,#data | AND immediae data to direct register | 3 | 4 |
| ORL A,Rn | OR register to accumulator | 1 | 1 |
| ORL A,direct | OR direct byte to accumulator | 2 | 2 |
| ORL A,@Ri | OR indirect RAM to accumulator | 1 | 2 |
| ORL direct,A | OR accumulator to direct byte | 2 | 3 |
| ORL direct,#data | OR immediate data to direct byte | 3 | 4 |
| XRL A,Rn | Exclusive OR register to accumulator | 1 | 1 |
| XRL A,direct | Exclusive OR direct byte to accumulator | 2 | 2 |
| XRL A,@Ri | Exclusive OR indirect RAM to accumulator | 1 | 2 |
| XRL A,#data | Exclusive OR immediate data to accumulator | 2 | 2 |
| XRL direct,A | Exclusive OR accumulator to direct byte | 2 | 3 |
| XORL direct,#data | Exclusive OR immediate data to direct byte | 3 | 4 |
| CLR A | Clears the accumulator | 1 | 1 |
| CPL A | Complements the accumulator (1=0, 0=1) | 1 | 1 |
| SWAP A | Swaps nibbles within the accumulator | 1 | 1 |
| RL A | Rotates bits in the accumulator left | 1 | 1 |
| RLC A | Rotates bits in the accumulator left through carry | 1 | 1 |

| | | | |
|---|---|---|---|
| RR A | Rotates bits in the accumulator right | 1 | 1 |
| RRC A | Rotates bits in the accumulator right through carry | 1 | 1 |

***Bit-oriented Instructions***

Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.

| BIT-ORIENTED INSTRUCTIONS | | | |
|---|---|---|---|
| **Mnemonic** | **Description** | **Byte** | **Cycle** |
| CLR C | Clears the carry flag | 1 | 1 |
| CLR bit | Clears the direct bit | 2 | 3 |
| SETB C | Sets the carry flag | 1 | 1 |
| SETB bit | Sets the direct bit | 2 | 3 |
| CPL C | Complements the carry flag | 1 | 1 |
| CPL bit | Complements the direct bit | 2 | 3 |
| ANL C,bit | AND direct bit to the carry flag | 2 | 2 |
| ANL C,/bit | AND complements of direct bit to the carry flag | 2 | 2 |
| ORL C,bit | OR direct bit to the carry flag | 2 | 2 |
| ORL C,/bit | OR complements of direct bit to the carry flag | 2 | 2 |
| MOV C,bit | Moves the direct bit to the carry flag | 2 | 2 |
| MOV bit,C | Moves the carry flag to the direct bit | 2 | 3 |

| S.NO | RGPV QUESTIONS | Year | Marks |
|---|---|---|---|
| Q.1 | Give an overview of 8051 instruction set. | DEC 2014 | 7 |

## UNIT V 8051 Microcontroller

## Lecture 9

**Assembly language Programs of 8051**

**Statement 1: -** exchange the content of FFh and FF00h

**Solution: -** here one is internal memory location and other is memory external location. so first the content of ext memory location FF00h is loaded in acc. then the content of int memory location FFh is saved first and then content of acc is transferred to FFh. now saved content of FFh is loaded in acc and then it is transferred to FF00h.

```
Mov dptr, #0FF00h     ; take the address in dptr
    Movx a, @dptr     ; get the content of 0050h in a
    Mov r0, 0FFh      ; save the content of 50h in r0
    Mov 0FFh, a       ; move a to 50h
    Mov a, r0         ; get content of 50h in a
    Movx @dptr, a     ; move it to 0050h
```