UNIT - 5
8051 INTERFACING ANDAPPLICATIONS

Interfacing of 8051with: Analog Sensors, Keypad & LCD display, ADC, DAC, DC motor.

LCD INTERFACING:

Figure 5.1 16X2 LCD Module

- 16×2 LCD module is a very common type of LCD module.
- It consists of 16 rows and 2 columns of 5×7 or 5×8 LCD dot matrices.
- It is available in a 16 pin package with back light, contrast adjustment function and each dot matrix has 5×8 dot resolution.
- The pin numbers, their name and corresponding functions are shown in the table 5.1.

Table 5.1 LCD Pin Description

Pin No:	Name	Function
1	**VSS**	This pin must be connected to the ground
2	**VCC**	Positive supply voltage pin (5V DC)
3	**VEE**	Contrast adjustment
4	**RS**	Register selection
5	**R/W**	Read or write
6	**E**	Enable
7	**DB0**	Data
8	**DB1**	Data
9	**DB2**	Data
10	**DB3**	Data
11	**DB4**	Data
12	**DB5**	Data
13	**DB6**	Data
14	**DB7**	Data
15	**LED+**	Back light LED+
16	**LED-**	Back light LED

V_{CC},V_{SS} & V_{EE} Pin:

- V_{CC} and V_{SS} provide +5V and Ground

- V_{EE} pin is meant for adjusting the contrast of the LCD display and the contrast can be adjusted by varying the voltage at this pin.
- This is done by connecting one end of a POT to the Vcc (5V), other end to the Ground and connecting the center terminal (wiper) of of the POT to the VEE pin. (Refer Figure 5.2)

RS:

- LCD has two built in registers namely data register and command register.
- Data register is for placing the data to be displayed, and the command register is to place the commands.
- High logic at the RS pin will select the data register and Low logic at the RS pin will select the command register.
- If we make the RS pin high and the put a data in the 8 bit data line (DB0 to DB7), the LCD module will recognize it as a data to be displayed.
- If we make RS pin low and put a data on the data line, the module will recognize it as a command.

R/W:

- R/W pin is meant for selecting between read and write modes.
- High level at this pin enables read mode and low level at this pin enables write mode.

Enable (E):

- E pin is for enabling the module.
- The enable pin is used by the LCD to latch information presented to its data pins.
- When data is supplied to data pins, a high to low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins.
- This pulse must be a minimum of 450ns wide.

Data Pin:

- The 8-bit data pins, DB0 to DB7 are used to send information to the LCD or read the contents of the LCD's internal register.
- To display letters and numbers, send ASCII codes for the letters A-Z; a-z and numbers 0-9 to these pins while making RS=1.
- There are also instruction command codes that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor.
- Table 5.2 Lists the instructions command codes.

Table 5.2 LCD Command Codes

CODE(Hexa Decimal)	COMMAND
01	Clear display screen
02	Return Home
04	Decrement cursor (shift cursor to left)
05	Increment cursor (shift cursor to right)
06	shift display right
07	shift display left
08	Display off, cursor off
0A	Display off, cursor on

0C	Display on, cursor off
0E	Display on, cursor blinking
0F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to the beginning of 1st line
C0	Force cursor to the beginning of 2nd line
38	2 lines and 5 x 7 matrix

- We also use RS=0 to check the busy flag bit to see if the LCD is ready to receive information's.
- The busy flag id D7 and can be read when R/W=1 and RS=0, as follows: if R/W=1, RS=0.
- When D7=1 (busy flag =1), the LCD is busy taking care of internal operations and will not accept any new information.

LED+ & LED-:

- LED+ is the anode of the back light LED and this pin must be connected to Vcc through a suitable series current limiting resistor.
- LED- is the cathode of the back light LED and this pin must be connected to ground.

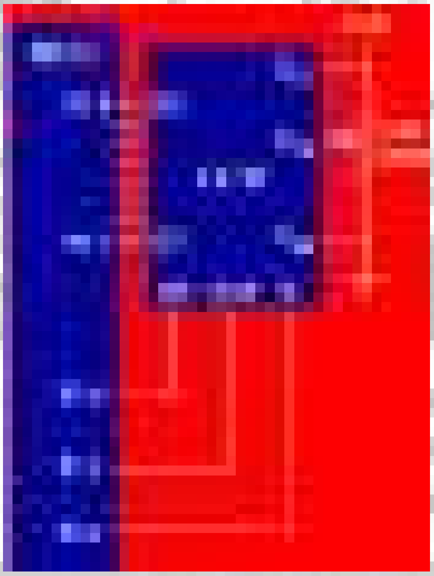


Figure 5.2 LCD Interfacing With 8051

LCD initialization

- The steps that has to be done for initializing the LCD display is given below and these steps are common for almost all applications.
  - Send 38H to the 8 bit data line for initialization
  - Send 0FH for making LCD ON, cursor ON and cursor blinking ON.

- Send 06H for incrementing cursor position.
- Send 80H for displaying the character from 1st row and 1st column in LCD
- Send 01H for clearing the display and return the cursor.

<u>Sending data to the LCD.</u>

- The steps for sending data to the LCD module is given below.
- It is the logic state of the pins (RS, R/W and E) that make the module to determine whether a given data input is a command or data to be displayed.
    - Make R/W low.
    - Make RS=0 if data byte is a command and make RS=1 if the data byte is a data to be displayed.
    - Place data byte on the data register.
    - Pulse E from high to low.
    - Repeat above steps for sending another data

<u>Program:</u>

```
                    ;calls a time delay before sending next data/command
                    ;P1.0-P1.7 are connected to LCD data pins D0-D7
                    ;P2.0 is connected to RS pin of LCD
                    ;P2.1 is connected to R/W pin of LCD
                    ;P2.2 is connected to E pin of LCD
        ORG 0
        MOV A,#38H          ;INIT. LCD 2 LINES, 5X7 MATRIX
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#0EH          ;display on, cursor on
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#01           ;clear LCD
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#06H          ;shift cursor right
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#84H          ;cursor at line 1, pos. 4
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#'N'          ;display letter N
        ACALL DATAWRT       ;call display subroutine
        ACALL DELAY          ;give LCD some time
        MOV A,#'O'           ;display letter O
        ACALL DATAWRT        ;call display subroutine
AGAIN:  SJMP AGAIN          ;stay here

COMNWRT:                     ;send command to LCD
```

```
            MOV P1,A            ;copy reg A to port 1
            CLR P2.0            ;RS=0 for command
            CLR P2.1            ;R/W=0 for write
            SETB P2.2            ;E=1 for high pulse
            ACALL DELAY
            CLR P2.2             ;E=0 for H-to-L pulse
            RET

DATAWRT:                         ;write data to LCD
            MOV P1,A            ;copy reg A to port 1
            CLR P2.0            ;RS=0 for command
            CLR P2.1             ;R/W=0 for write
            SETB P2.2           ;E=1 for high pulse
            ACALL DELAY
            CLR P2.2            ;E=0 for H-to-L pulse
            RET

DELAY:      MOV R3,#50          ;50 or higher for fast CPUs
HERE2:      MOV R4,#255         ;R4 = 255
HERE:       DJNZ R4,HERE        ;stay until R4 becomes 0
            DJNZ R3,HERE2
            RET
            END
```

Check Busy Flag:

- The above code showed how to send command to the LCD without checking the busy flag.
- Notice that we put a long delay between issuing data or commands to the LCD.
- However a much better way is to monitor the busy flag before issuing a command or data to the LCD. This is shown in below program

```
            ORG 0
            MOV A,#38H          ;INIT. LCD 2 LINES, 5X7 MATRIX
            ACALL COMNWRT       ;call command subroutine
            MOV A,#0EH          ;display on, cursor on
            ACALL COMNWRT       ;call command subroutine
            MOV A,#01           ;clear LCD
            ACALL COMNWRT       ;call command subroutine
            MOV A,#06H          ;shift cursor right
            ACALL COMNWRT       ;call command subroutine
            MOV A,#84H          ;cursor at line 1, pos. 4
            ACALL COMNWRT       ;call command subroutine
            MOV A,#'N'          ;display letter N
            ACALL DATAWRT       ;call display subroutine
            MOV A,#'O'            ;display letter O
            ACALL DATAWRT         ;call display subroutine
AGAIN:      SJMP AGAIN           ;stay here

COMNWRT:    ACALL READY           ;send command to LCD if LCD is ready
            MOV P1,A             ;copy reg A to port 1
            CLR P2.0             ;RS=0 for command
            CLR P2.1             ;R/W=0 for write
            SETB P2.2             ;E=1 for high pulse
            ACALL DELAY
            CLR P2.2              ;E=0 for H-to-L pulse
            RET

DATAWRT:    ACALL READY           ;write data to LCD if LCD is ready
            MOV P1,A             ;copy reg A to port 1
            CLR P2.0             ;RS=0 for command
            CLR P2.1              ;R/W=0 for write
            SETB P2.2            ;E=1 for high pulse
            ACALL DELAY
            CLR P2.2             ;E=0 for H-to-L pulse
            RET


READY:
            SETB P1.7             ;make P1.7 input port
            CLR P2.0             ;RS=0 access command reg
            SETB P2.1             ;R/W=1 read command reg
                                 ;read command reg and check busy flag
BACK:       SETB P2.2            ;E=1 for H-to-L pulse
            CLR P2.2             ;E=0 H-to-L pulse
            JB P1.7,BACK         ;stay until busy flag=0
            RET
            END
```

LCD Interfacing Using MOVC Instruction:

```
            ORG 0
            MOV DPTR,#MYCOM
      C1:   CLR A
            MOVC A,@A+DPTR
            ACALL COMNWRT              ;call command subroutine
            ACALL DELAY                 ;give LCD some time
            INC DPTR
            JZ SEND_DAT
            SJMP C1
SEND_DAT:
```

```
            MOV DPTR,#MYDATA
    D1:     CLR A
            MOVC A,@A+DPTR
            ACALL DATAWRT           ;call command subroutine
            ACALL DELAY             ;give LCD some time
            INC DPTR
            JZ AGAIN
            SJMP D1
AGAIN:      SJMP AGAIN

            ORG 300H
MYCOM:      DB 38H,0EH,01,06,84H,0  ; commands and null
MYDATA:     DB "HELLO",0
            END
```

KEYBOARD INTERFACING:

- At the lowest level, keyboards are organized in a matrix of rows and columns.
- The CPU accesses both rows and columns through ports; therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor.
- When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns

Scanning and identifying the key

- Figure 5.3 shows a 4 x4 matrix connected to two ports.
- The rows are connected to an output port and the columns are connected to an input port.
- If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (Vcc).
- If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground.
- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed, How it is done is explained next.

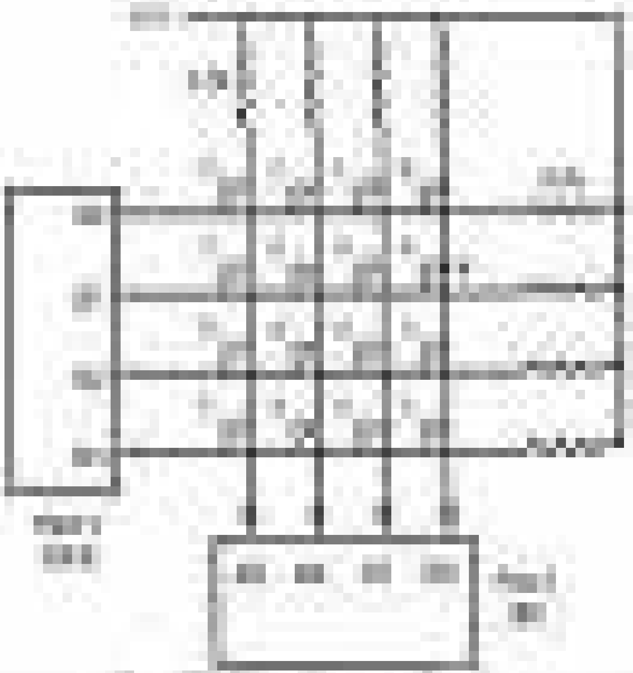


Figure 5.3 Matrix Keyboard Connection to Ports

Grounding rows and reading the columns

- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns.
- If the data read from the columns is D3 - D0 =1111, no key has been pressed and the process continues until a key press is detected.
- However, if one of the column bits has a zero, this means that a key press has occurred.
- For example, if D3 - D0 = 1101, this means that a key in the D1 column has been pressed.
- After a key press is detected, the microcontroller will go through the process of identifying the key.
- Starting with the top row, the microcontroller grounds it by providing a low to row D0 only; then it reads the columns.
- If the data read is all 1s, no key in that row is activated and the process is moved to the next row.
- It grounds the next row, reads the columns, and checks for any zero.
- This process continues until the row is identified.
- After identification of the row in which the key has been pressed, the next task is to find out which column the pressed key belongs to.
- This should be easy since the microcontroller knows at any time which row and column are being accessed.
- Given keyboard program is the 8051 Assembly language program for detection and identification of key activation.
- In this program, it is assumed that P1 and P2 are initialized as output and input, respectively.
- Program goes through the following four major stages:

- To make sure that the preceding key has been released, 0s are output to all rows at once, and the columns are read and checked repeatedly until all the columns are high. When all columns are found to be high, the program waits for a short amount of time before it goes to the next stage of waiting for a key to be pressed.
- To see if any key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it. Remember that the output latches connected to rows still have their initial zeros (provided in stage 1), making them grounded. After the key press detection, the microcontroller waits 20 ms for the bounce and then scans the columns again. This serves two functions: (a) it ensures that the first key press detection was not an erroneous one due to a spike noise, and (b) the 20-ms delay prevents the same key press from being interpreted as a multiple key press. If after the 20-ms delay the key is still pressed, it goes to the next stage to detect which row it belongs to; otherwise, it goes back into the loop to detect a real key press.
- To detect which row the key press belongs to, the microcontroller grounds one row at a time, reading the columns each time. If it finds that all columns are high, this means that the key press cannot belong to that row; therefore, it grounds the next row and continues until it finds the row the key press belongs to. Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or the ASCII value) for that row and goes to the next stage to identify the key.
- To identify the key press, the microcontroller rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low. Upon finding the zero, it pulls out the ASCII code for that key from the look-up table; otherwise, it increments the pointer to point to the next element of the look-up table. Figure 5.4 flowcharts this process.

➢ While the key press detection is standard for all keyboards, the process for determining which key is pressed varies.

➢ The look-up table method shown in keyboard Program can be modified to work with any matrix up t0 8 x 8.

➢ Figure 5.4 provides the flowchart for keyboard interfacing Program for scanning and identifying the pressed key.

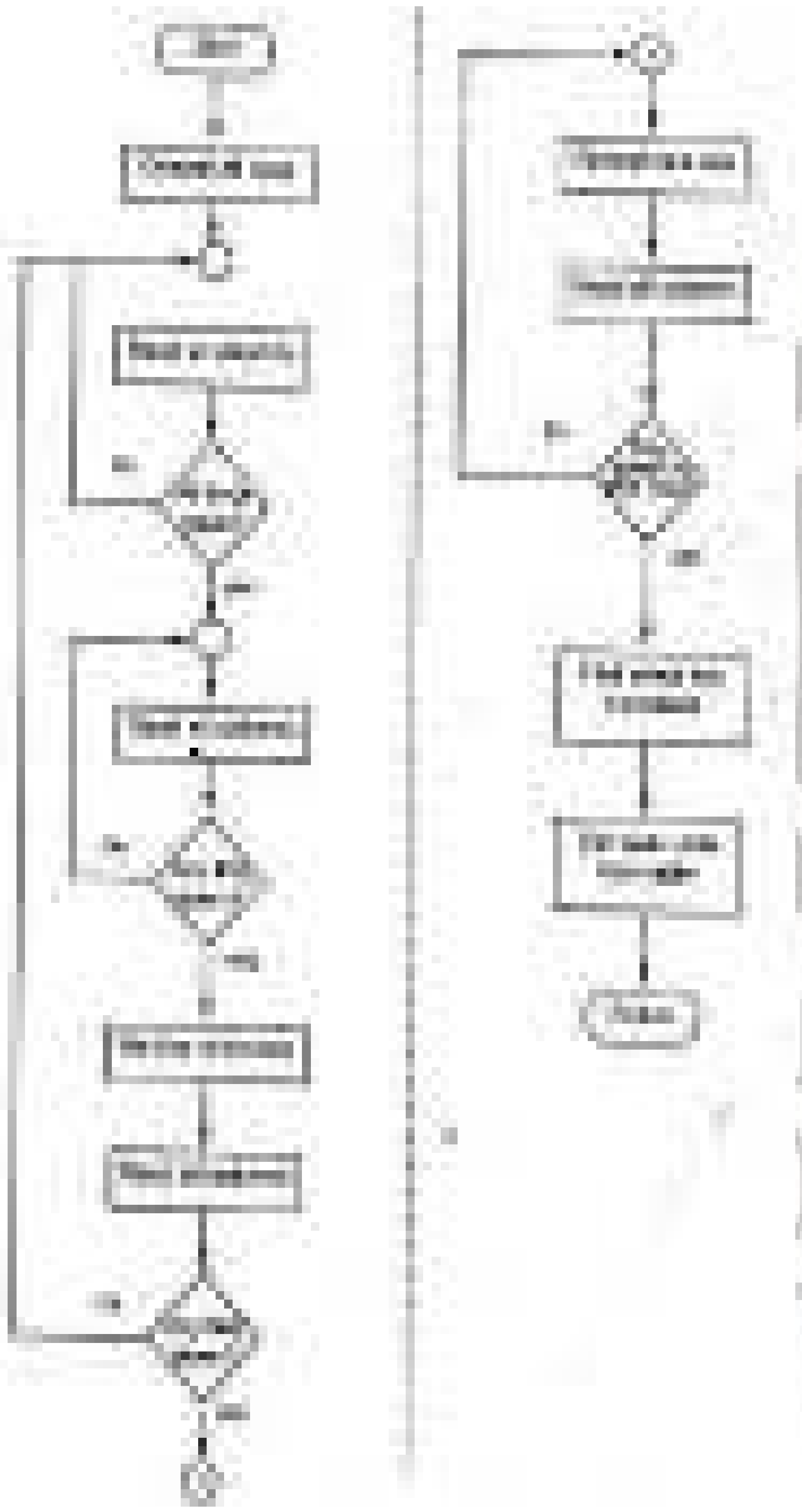


Figure 5.4 Flowchart for Programming Keyboard Interfacing

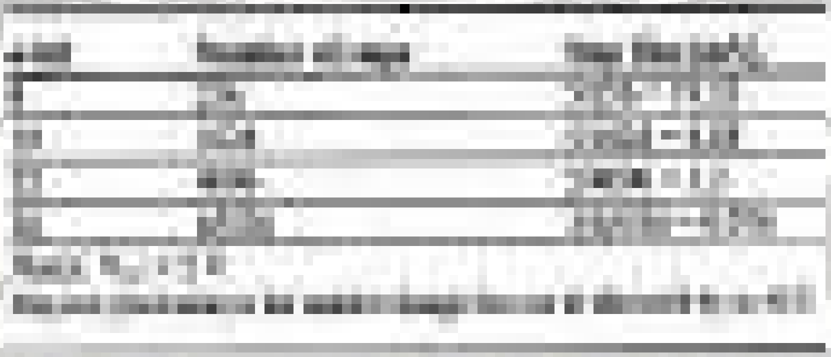
Program:

```
                    ;keyboard subroutine. This program sends the ASCII
                    ;code for pressed key to P0.1
                    ;P1.0-P1.3 connected to rows, P2.0-P2.3 to column
        MOV P2,#0FFH                ;make P2 an input port
K1:     MOV P1,#0                   ;ground all rows at once
        MOV A,P2                    ;read all col
                                    ;(ensure keys open)
        ANL A,00001111B             ;masked unused bits
        CJNE A,#00001111B,K1        ;till all keys release
K2:     ACALL DELAY                 ;call 20 msec delay
        MOV A,P2                    ;see if any key is pressed
        ANL A,00001111B             ;mask unused bits
        CJNE A,#00001111B,OVER  ;key pressed, find row
        SJMP K2                     ;check till key pressed
OVER: ACALL DELAY                   ;wait 20 msec debounce time
        MOV A,P2                    ;check key closure
        ANL A,00001111B             ;mask unused bits
        CJNE A,#00001111B,OVER1;key pressed, find row
        SJMP K2                     ;if none, keep polling
OVER1: MOV P1, #11111110B           ;ground row 0
        MOV A,P2                    ;read all columns
        ANL A,#00001111B            ;mask unused bits
        CJNE A,#00001111B,ROW_0 ;key row 0, find col.
        MOV P1,#11111101B           ;ground row 1
        MOV A,P2                    ;read all columns
        ANL A,#00001111B            ;mask unused bits
        CJNE A,#00001111B,ROW_1 ;key row 1, find col.
        MOV P1,#11111011B           ;ground row 2
        MOV A,P2                    ;read all columns
        ANL A,#00001111B            ;mask unused bits
        CJNE A,#00001111B,ROW_2 ;key row 2, find col.
        MOV P1,#11110111B           ;ground row 3
        MOV A,P2                    ;read all columns
        ANL A,#00001111B            ;mask unused bits
        CJNE A,#00001111B,ROW_3 ;key row 3, find col.
        LJMP K2                     ;if none, false input,
                                    ;repeat
ROW_0: MOV DPTR,#KCODE0             ;set DPTR=start of row 0
        SJMP FIND                   ;find col. Key belongs to
ROW_1: MOV DPTR,#KCODE1             ;set DPTR=start of row
        SJMP FIND                   ;find col. Key belongs to
ROW_2: MOV DPTR,#KCODE2             ;set DPTR=start of row 2
        SJMP FIND                   ;find col. Key belongs to
        ROW_3: MOV DPTR,#KCODE3 ;set DPTR=start of row 3
FIND:   RRC A                   ;see if any CY bit low
        JNC MATCH               ;if zero, get ASCII code
        INC DPTR                ;point to next col. addr
        SJMP FIND               ;keep searching
MATCH: CLR A                    ;set A=0 (match is found)
        MOVC A,@A+DPTR          ;get ASCII from table
        MOV P0,A                ;display pressed key
        LJMP K1
                                ;ASCII LOOK-UP TABLE FOR EACH ROW
        ORG 300H
KCODE0: DB '0','1','2','3' ;ROW 0
KCODE1: DB '4','5','6','7' ;ROW 1
KCODE2: DB '8','9','A','B' ;ROW 2
KCODE3: DB 'C','D','E','F' ;ROW 3
        END
```

ANALOG-TO-DIGITAL CONVERTER (ADC) INTERFACING:

- ADCs (analog-to-digital converters)are among the most widely used devices for data acquisition.
- A physical quantity, like temperature, pressure, humidity, and velocity, etc., is converted to electrical (voltage, current)signals using a device called a transducer or sensor
- We need an analog-to-digital converter to translate the analog signals to digital numbers, so microcontroller can read and process them.
- An ADC has n-bit resolution where n can be 8, 10, 12, 16 or even 24 bits.
- The higher-resolution ADC provides a smaller step size, where step size is the smallest change that can be discerned by an ADC. This is shown in table 5.3

Table 5.3 Resolution Vs Step Size for ADC



n-bit	Number of steps	Step Size (mV)
8	256	5/256 = 19.53
10	1024	5/1024 = 4.88
12	4096	5/4096 = 1.2
16	65536	5/65536 = 0.076

Notes: V_{CC} = 5 V

Step size (resolution) is the smallest change that can be discerned by an ADC.

- In addition to resolution, conversion time is another major factor in judging an ADC.
- Conversion time is defined as the time it takes the ADC to convert the analog input to a digital (binary) number.
- The ADC chips are either parallel or serial.
- In parallel ADC, we have 8 of more pins dedicated to bringing out the binary data, but in serial ADC we have only one pin for data out.

ADC804 chip:

- ADC804 IC is an 8-bit parallel analog-to-digital converter.
- It works with +5 volts and has a resolution of 8bits.
- In ADC804 conversion time varies depending on the clocking signals applied to the CLK R and CLK IN pins, but it cannot be faster than 110µs.
- Figure 5.5 Pin out of ADC0804 in free running mode.
- The following is the ADC0804 pin description.

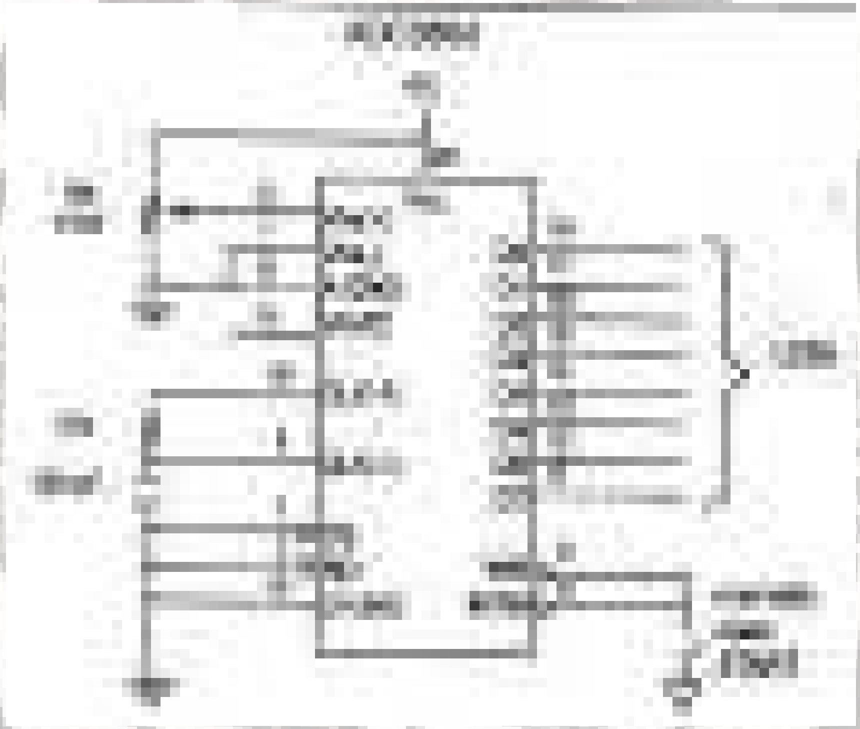


Figure 5.5 ADC0804 Chip (Testing ADC0804 in Free Running Mode)

- **CLK IN and CLK R:**
  - CLK IN is an input pin connected to an external clock source when an external clock is used for timing.
  - However, the 0804 has an internal clock generator.
  - To use the internal clock generator (also called self-clocking), CLK IN and CLK R pins are connected to a capacitor and a resistor and the clock frequency is determined by:

$$f = \frac{1}{1.1\,RC}$$

  - Typical values are R = 10K ohms and C =150pF.
  - By substituting, we get f = 606 kHz and the conversion time is 110µs.
- **Vref/2: (Pin 9)**
  - It is used for the reference voltage.
  - If this pin is open (not connected), the analog input voltage is in the range of 0 to 5 volts (the same as the Vcc pin).
  - If the analog input range needs to be 0 to 4 volts, Vref/2 is connected to 2 volts.
  - Table 5.4 shows the Vin range for various Vref/2 inputs.

Table 5.4 Vref/2 Relation to Vin Range (ADC0804)

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

[illegible]

[illegible]

- **D0-D7:**
  - D0-D7 are the digital data output pins.
  - These are tri-state buffered and the converted data is accessed only when CS =0 and RD is forced low.
  - To calculate the output voltage, use the following formula

$$D_{out} = \frac{V_{in}}{\text{step size}}$$

    - Dout = digital data output (in decimal),
    - Vin = analog voltage, and
    - Step size (resolution) is the smallest change, which is (2 * Vref/2)/256 for ADC 0804
- **Analog ground and digital ground:**
  - Analog ground is connected to the ground of the analog Vin and digital ground is connected to the ground of the Vcc pin.
  - The reason that to have ground pin is to isolate the analog Vin signal from transient voltages caused by digital switching of the output D0 – D7. This contributes to the accuracy of the digital data output.
- **Vin(+) & Vin(-):**
  - Differential analog inputs where Vin= Vin (+) – Vin (-).
  - Vin (-) is connected to ground and Vin(+) is used as the analog input to be converted.
- **RD:**
  - This is an input signal and is active low.
  - The ADC converts the analog input to its binary equivalent and holds it in an internal register.
  - RD is, used to get the converted data out of the ADC0804 chip.
  - Is “output enable” a high-to-low RD pulse is used to get the 8-bit converted data out of ADC804.
- **INTR:**
  - This is an output pin and is active low.
  - It is “end of conversion” When the conversion is finished, it goes low to signal the CPU that the converted data is ready to be picked up.

- **WR:**
  - This is an active low input
  - It is "start conversion" When WR makes a low-to-high transition, ADC804 starts converting the analog input value of Vin to an 8-bit digital number.
  - When the data conversion is complete, the INTR pin is forced low by the ADC0804.
- **CS:**
  - It is an active low input used to activate ADC804.
- **Steps to Be followed For Data Conversion:**
  - The following steps must be followed for data conversion by the ADC804 chip:
    - Make CS= 0 and send a L-to-H pulse to pin WR to start conversion.
    - Monitor the INTR pin, if high keep polling but if low, conversion is complete, go to next step.
    - Make CS= 0 and send a H-to-L pulse to pin RD to get the data out.
  - Figure 5.6 shows the timing diagram for ADC process.

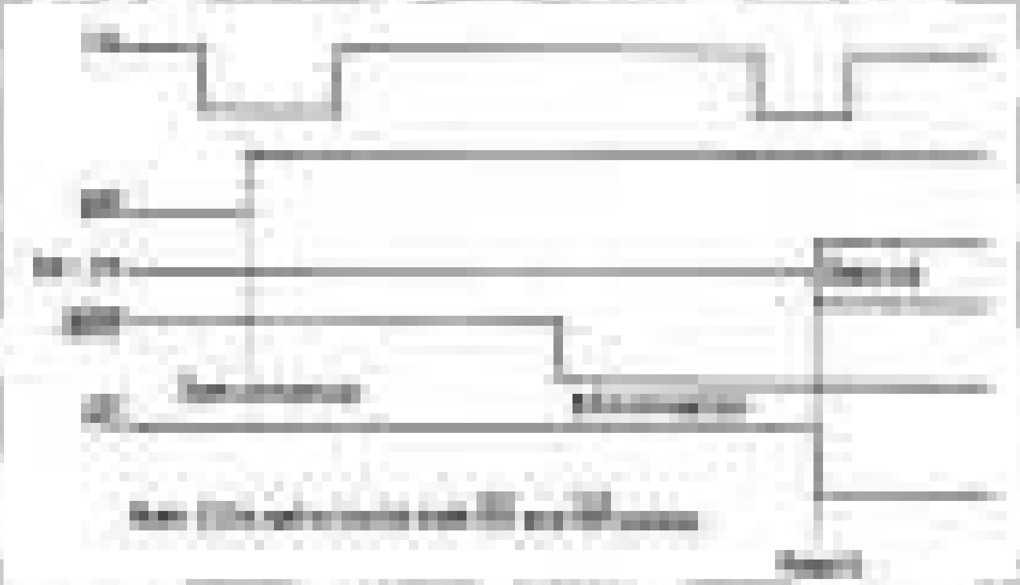


Figure 5.6 Read and Write Timing for ADC08804

<u>Clock source for ADC0804:</u>

- The speed at which an analog input is converted to the digital output depends on the speed of the CLK input.
- According to the ADC0804 datasheets, the typical operating frequency is approximately 640kHz at 5 volts.
- Figures 5.7 and 5.8 show two ways of providing clock to the ADC0804.
- In Figure 5.8, notice that the clock in for the ADC0804 is coming from the crystal of the microcontroller.
- Since this frequency is too high, we use D flip-flops (74LS74) to divide the frequency.
- A single D flip-flop divides the frequency by 2 if we connect its $\bar{Q}$ to the D input.
- For a higher-frequency crystal, you can use 4 flip-flops

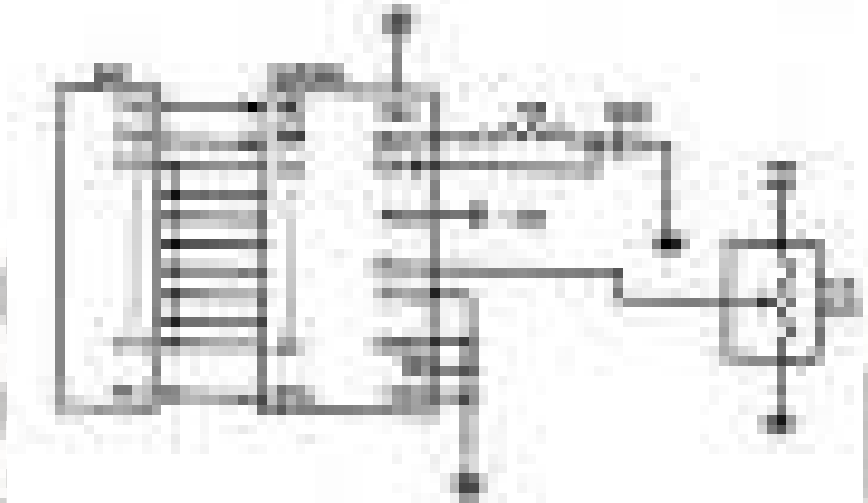


Figure 5.7 8051 Connection to ADC0804 with Self-Clocking

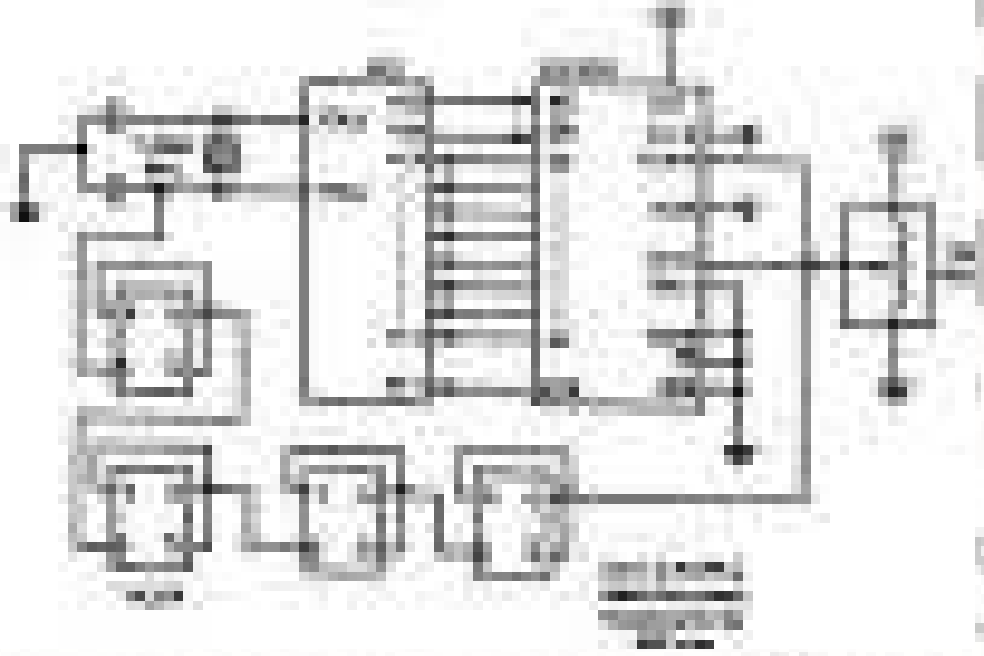


Figure 5.8 8051 Connection to ADC0804 with Clock from XTAL2 of the 8051

Example:
Write a program to monitor the INTR pin and bring an analog input into register A. Then call a hex-to ACSII conversion and data display subroutines. Do this continuously.

```
            ;p2.6=WR (start conversion needs to L-to-H pulse)
            ;p2.7 When low, end-of-conversion)
            ;p2.5=RD (a H-to-L will read the data from ADC chip)
            ;p1.0 – P1.7= D0 - D7 of the ADC804
            ;
            MOV P1,#0FFH                ;make P1 = input
BACK:       CLR P2.6                    ;WR = 0
```

```
            SETB P2.6                ;WR = 1 L-to-H to start conversion
HERE:       JB P2.7,HERE              ;wait for end of conversion
            CLR P2.5                 ;conversion finished, enable RD
            MOV A,P1                  ;read the data
            ACALL CONVERSION          ;hex-to-ASCII conversion
            ACALL DATA_DISPLAY       ;display the data
            SETB P2.5                ;make RD=1 for next round
            SJMP BACK
```

ADC0808:

- While the ADC0804 has only one analog input, this chip has 8 of them.
- The ADC0808/0809 chip allows us to monitor up to 8 different analog inputs using only a single chip.
- Notice that the ADC0808/0809 has an 8-bit data output just like the ADC804.
- The 8 analog input channels are multiplexed and selected according to Table 5.5 using three address pins, A, B, and C.

Table 5.5 Channel Selection in ADC0808

[illegible]

- In the ADC0808/0809, Vref (+) and Vref.(-) set the reference voltage.
- If Vref(-) = Gnd and Vref (+) = 5 V, the step size is 5 V/256 = 19.53 mV.
- Therefore, to get a l0 mV step size we need to set Vref (+) = 2.56 V and Vref.(-) = Gnd.
- From Figure 5.9, notice the ALE pin.
- We use A, B, and C addresses to select.IN0 - IN7, and activate ALE to latch in the address.
- SC is for start conversion.
- SC is the same as the WR pin in other ADC chips.
- EOC is for end-of-conversion, and OE is for output enable (READ).
- The EOC and OE are the same as the INTR and RD pins respectively.
- Table 5.6 shows the step size relation to the Vref voltage.
- Notice that there is no Vref/2 in the ADC0808/0809 chip.

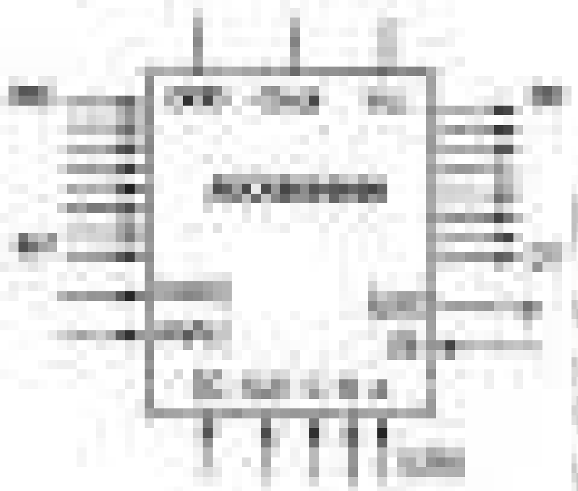


Figure 5.9 ADC0808/0809

Table 5.6 ADC0808/0809 Analog Channel Selection

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

<u>Steps to program the ADC0808/0809</u>

- The following are steps ro get data from an ADC0808/0809.
  - Select an analog channel by providing bits to A, B, and C addresses according to Table 5.6.
  - Activate the ALE (address latch enable) pin. It needs an L-to-H pulse to latch in the address.
  - Activate SC (start conversion) by an L-to-H pulse to initiate conversion.
  - Monitor EOC (end of conversion) to see whether conversion is finished. H-to- L output indicates that the data is converted and is ready to be picked up. If we do not use EOC, we can read the converted digital data after a brief time delay. The delay size depends on the speed of the external clock we connect to the CLK pin. Notice that the EOC is the same as the INTR pin in other ADC chips.
  - Activate OE (output enable) to read data out of the ADC chip. An L-to H pulse to the OE pin will bring digital data out of the chip. Also notice that the OE is "the same as the RD pin in other ADC chips.
- The speed of conversion depends on the frequency of the clock connected to the CLK pin, it cannot be faster than 100 microseconds

SENSOR INTERFACING:

<u>LM35 Temperature sensors:</u>

- The LM35 series sensors are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the celsius (centigrade) temperature.
- The LM35 requires no external calibration since it is internally calibrated.
- It outputs 10mV for each degree of centigrade temperature.
- Table 5.7 is the selection guide for the LM35

Table 5.7 LM35 Temperature Sensor Series Selection Guide

- The sensors of the LM34 series are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Fahrenheit temperature.
- It also internally calibrated.
- It outputs 10mV for each degree Fahrenheit temperature.

<u>Signal Conditioning and Interfacing the LM35 to the 8051</u>

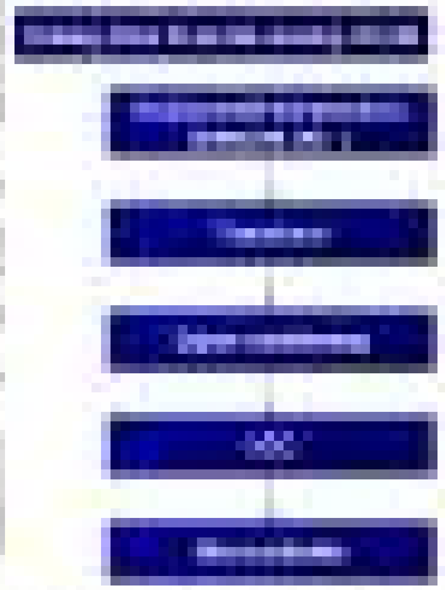


Figure 5.10 Getting Data from Analog World

- The above figure 5.10 shows the steps involved in acquiring data from analog world.
- Signal conditioning is widely used in the world of data acquisition.
- The most common transducers produce an output in the form of voltage, current, charge, capacitance, and resistance.
- However, we need to convert these signals to voltage in order to send input to an A-to-D converter.

- This conversion (modification) is commonly called signal conditioning.
- Signal conditioning can be a current-to-voltage conversion or a signal amplification.
- For example, the thermistor changes resistance with temperature.
- The change of resistance must be translated into voltages in order to be of any use to an ADC.
- Look at the case of connecting an LM35 to an ADC0848.
- Since the ADC0848 has 8-bit resolution with a maximum of 256 (2^8) steps and the LM35 (or LM34) produces l0 mV for every degree of temperature change, we can condition V_{in} of the ADC0848 to produce a V_{out}, of 2560 mV (2.56 V) for full-scale output.
- Therefore, in order to produce the full-scale Vout of 2.56 V for the ADC0848, we need to set V_{ref} = 2.56.
- This makes V_{out}, of the ADC0848 correspond directly to the temperature as monitored by the LM35. Refer the table 5.8

Table 5.8 Temperature vs. Vout for ADC0848

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

- Figure 5.11 shows the connection of a temperature sensor to the ADC0848.
- The LM336-2.5 zener diode to fix the voltage across the 10K pot at 2.5V.
- The use of the LM336-2.5 should overcome any fluctuations in the power supply.

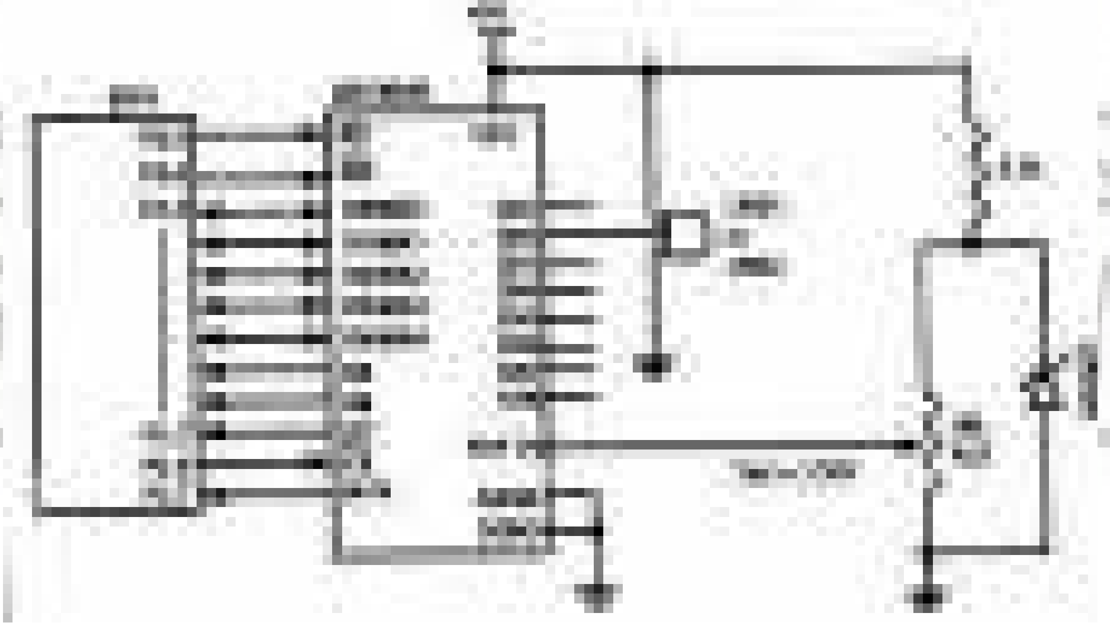


Figure 5.11 8051 Connection to ADC0848 and Temperature sensor

Program:

```
            RD BIT P2.5                 ;RD
            WR BIT P2.6                 ;WR
            INTR BIT P2.7               ; END OF CONVERSION
            MYDATA EQU P1               ; P1.0-P1.7 = D0-D7 OF THE ADC0848
            MOV P1,#0FFH                ;make P1 = input
            SETB INTR
BACK:       CLR WR                       ;WR = 0
            SETB WR                     ;WR = 1 L-to-H to start conversion
HERE:       JB INTR,HERE                 ;wait for end of conversion
            CLR RD                      ;conversion finished, enable RD
            MOV A,MYDATA                            ;read the data
            ACALL CONVERSION             ;hex-to-ASCII conversion
            ACALL DATA_DISPLAY          ;display the data
            SETB RD                     ;make RD=1 for next round
            SJMP BACK

CONVERSION:
            MOV B,#10
            DIV AB
            MOV R7,B
            MOV B,#10
            DIV AB
            MOV R6,B
            MOV R5,A
            RET

DATA_DISPLAY:
            MOV P0,R7
            ACALL DELAY
            MOV P0,R6
            ACALL DELAY
            MOV P0,R5
            ACALL DELAY
            RET
```

DIGITAL-TO-ANALOG (DAC) CONVERTER:

- The DAC is a device widely used to convert digital pulses to analog signals.
- In this section we will discuss the basics of interfacing a DAC to 8051.
- The two method of creating a DAC is binary weighted and R/2R ladder.
- The Binary Weighted DAC, which contains one resistor or current source for each bit of the DAC connected to a summing point.
- These precise voltages or currents sum to the correct output value.

- This is one of the fastest conversion methods but suffers from poor accuracy because of the high precision required for each individual voltage or current.
- Such high-precision resistors and current-sources are expensive, so this type of converter is usually limited to 8-bit resolution or less.
- The R-2R ladder DAC, which is a binary weighted DAC that uses a repeating cascaded structure of resistor values R and 2R.
- This improves the precision due to the relative ease of producing equal valued matched resistors (or current sources).
- However, wide converters perform slowly due to increasingly large RC-constants for each added R-2R link.
- The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs.
- The common ones are 8, 10, and 12 bits.
- The number of data bit inputs decides the resolution of the DAC since the number of analog output levels is equal to 2^n, where n is the number of data bit inputs.
- Therefore, an 8-input DAC such as the DAC0808 provides 256 discrete voltage (or current) levels of output.
- Similarly, the 12-bit DAC provides 4096 discrete voltage levels.
- There also 16-bit DACs, but they are more expensive.

DAC0808:

- The digital inputs are converter to current (I_{out}), and by connecting a resistor to the I_{out} pin, we can convert the result to voltage.
- The total current provided by the I_{out} pin is a function of the binary numbers at the D0-D7 inputs of the DAC0808 and the reference current (I_{ref}), and is as follows

[illegible]

- Usually reference current is 2mA.
- Ideally we connect the output pin to a resistor, convert this current to voltage, and monitor the output on the scope.
- But this can cause inaccuracy; hence an opamp is used to convert the output current to voltage.
- The 8051 connection to DAC0808is as shown in the below figure 5.12.
- Now assuming that I_{ref} = 2mA, if all the inputs to the DAC are high, the maximum output current is 1.99mA.

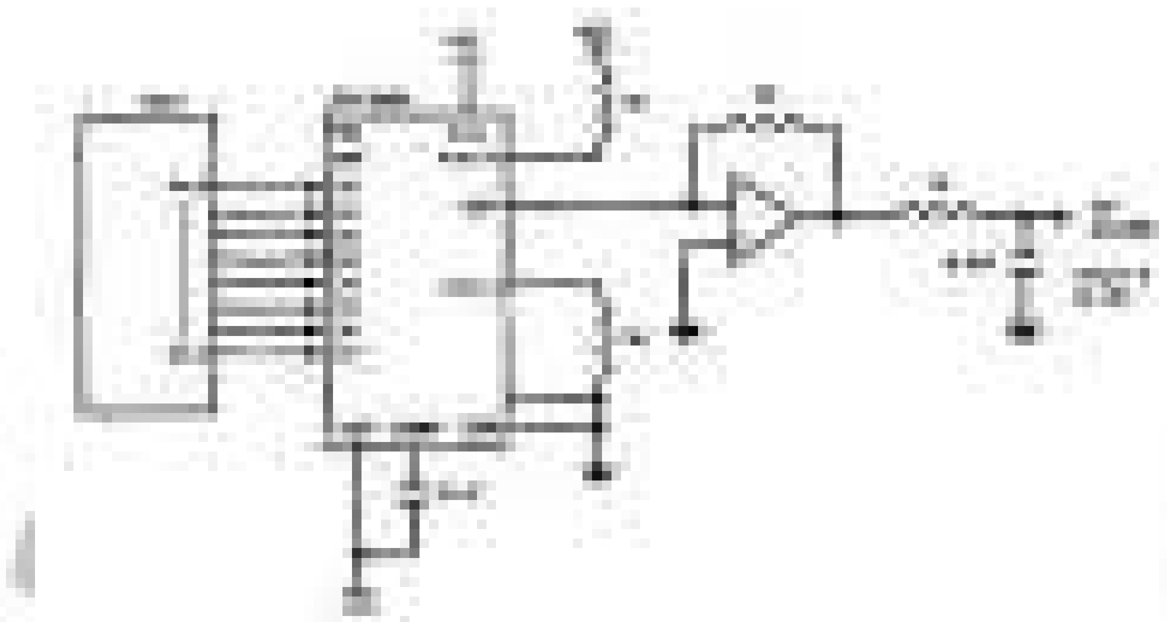


Figure 5.12 8051 Connection to DAC808

Example 1:

Assuming that R=5K and Iref=2mA, calculate Vout for the following binary inputs:

(a) 10011001B

(b) 11001000B

Solution:

(a) Iout = 2mA(153/256) = 1.195mA and Vout = 1.195mA * 5K =5.975V

(b) Iout = 2mA(200/256) = 1.562mA and Vout = 1.562mA * 5K =7.8125V

Converting I_{out} to voltage in DAC0808:

- Ideally we connect the output pin I_{out}, to a resistor, convert this current to voltage, and monitor the output on the scope.
- In real life, however, this can cause inaccuracy since the input resistance of the load where it is connected will also affect the output voltage.
- For this reason, the I_{ref} current output is isolated by connecting it to an op-amp such as the 741 with R_f = 5K ohms for the feedback resistor.
- Assuming that R= 5K ohms, by changing the binary input, the output voltage changes as shown in Example 2.

Example 2:

Inorder to generate a stair-step ramp, set up the circuit in figure 5.12 and connect the output to an oscilloscope. Then write a program to send data to the DAC to generate a stair-step ramp.

Solution:

```
            CLR A
AGAIN:      MOV P1,A          ; SEND DATA TO DAC
            INC A             ; COUNT FROM 0 TO FFH
            ACALL DELAY       ; LET DAC RECOVER
            SJMP AGAIN
```

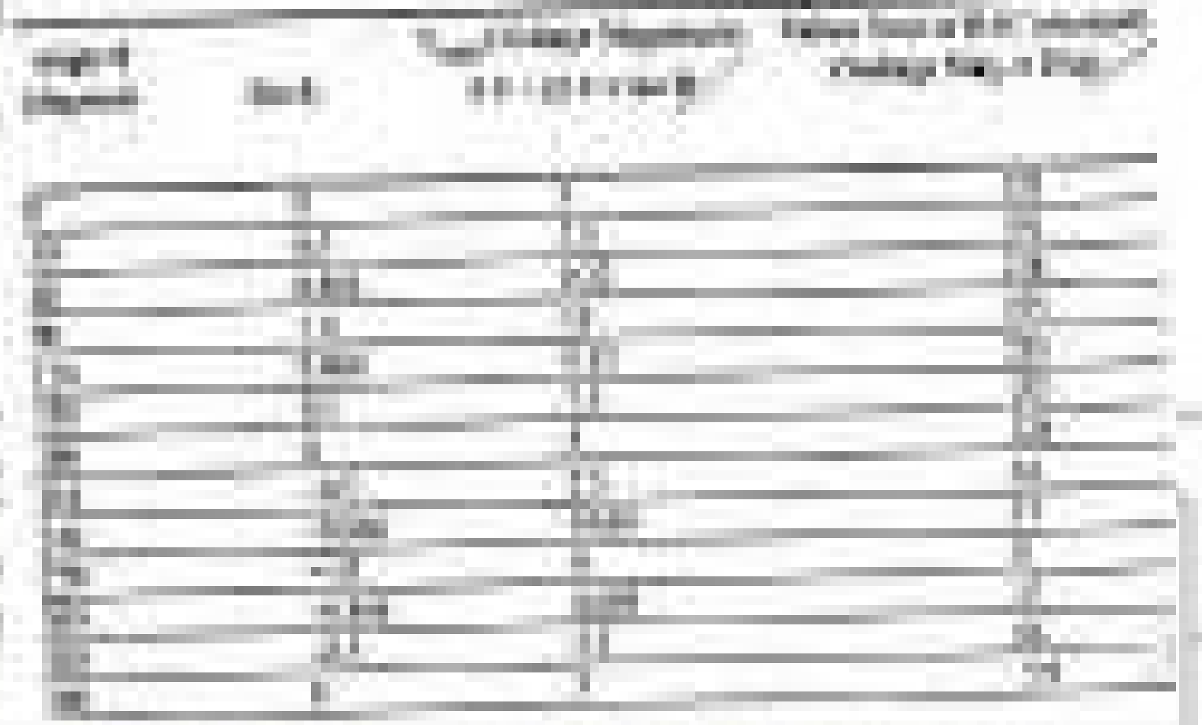
Generating a sine wave

- To generate a sine wave, we first need a table whose values represent the magnitude of the sine of angles between 0 and 360 degrees.
- The values for the sine function vary from -1.0 to +1.0 for 0- to 360-degree angles.
- Therefore, the table values are integer numbers representing the voltage magnitude for the sine of theta.
- This method ensures that only integer numbers are output to the DAC by the 805l microcontroller.
- Table 5.9 shows the angles, the sine values, the voltage magnitudes, and the integer values representing the voltage magnitude for each angle (with 30-degree increments).
- To generate Table 5.9,we assumed the full-scale voltage of 10 V for DAC output (as designed in Example 4 Figure).
- Full-scale output of the DAC is achieved when all the data inputs of the DAc are high.
- Therefore, to achieve the full-scale 10 V output, we use the following equation

$$\mathbf{V_{out} = 5V(1+\sin\theta)}$$

- Vout of DAC for various angles is calculated and shown in Table 5.9. See Example 3 for verification of the calculations

Table 5.9 Angle Vs Voltage Magnitude for Sine Wave


[illegible]

Example 3:

Verify the values given for the following angles: (a) 30º (b) 60º

Solution:

(a) Vout = 5V+(5V * sin30) =7.5V
DAC input values = 7.5V * 25.6 = 192 (Decimal)

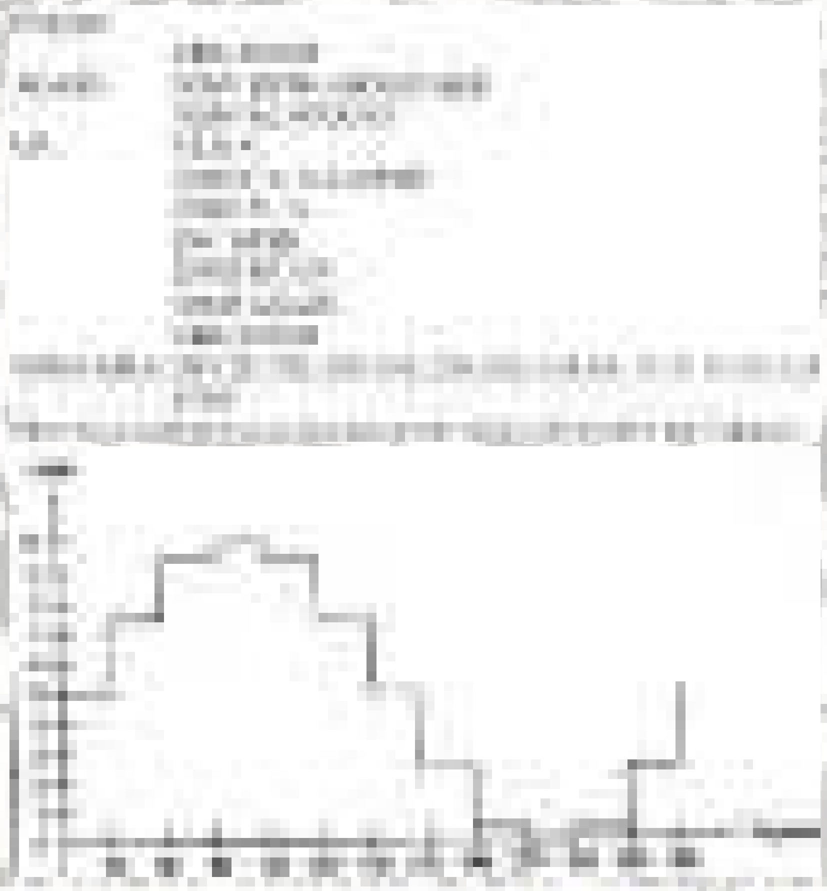
(b) Vout = 5V+(5V * sin60) =9.33V
DAC input values = 9.33V * 25.6 = 238 (Decimal)

- To find the values sent to the DAC for various angles, we simply multiply Vout voltage by 25.6 because there are 256 steps and full scale Vout is 10 volts.
  256 steps/10V = 25.6 steps per volt
- The following examples 9, 10 and 11 will show the generation of waveforms using DAC0808.

Example 4:
Write an ALP to generate a sine waveform.

$$V_{out} = 5V(1+\sin\theta)$$

Solution:
Calculate the decimal values for every 10 degree of the sine wave. These values can be maintained in a table and simply the values can be sent to port P1. The sine wave can be observed on the CRO.



```
[illegible]
```

Example 5:
Write an ALP to generate a triangular waveform.

[illegible]

DC MOTOR INTERFACING:

- DC motor is a device that translates electrical pulses into mechanical movement.
- The DC motor has + and – leads
- Connecting them to a DC voltage source moves the motor in one direction and by reversing the polarity, the DC motor will move in opposite direction.

Unidirectional Control:

- The following figure 5.13 shows the DC motor rotation for clockwise (CW) and counterclockwise (CCW) rotations.

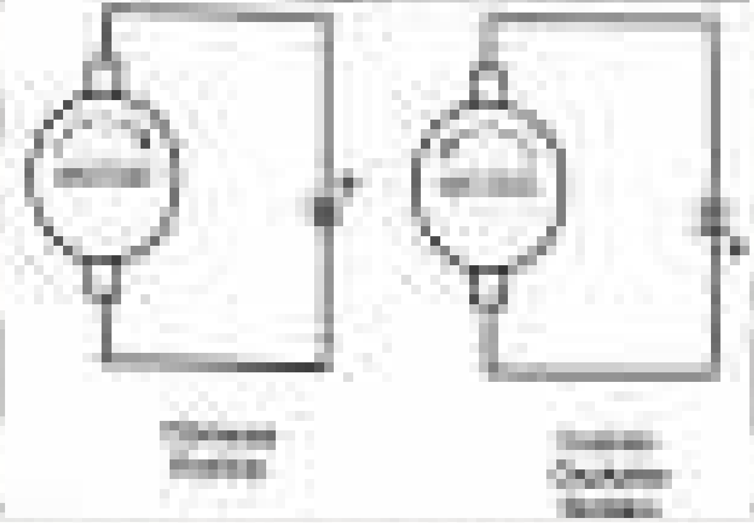


Figure 5.13 DC Motor Rotation (Permanent Magnet Field)

Bidirectional Control:

- With the help of relays or some specially designed chips we can change the direction of the DC motor rotation.
- Figure 5.14 through 5.17 shows the basic concepts of H-Bridge control of DC motors.

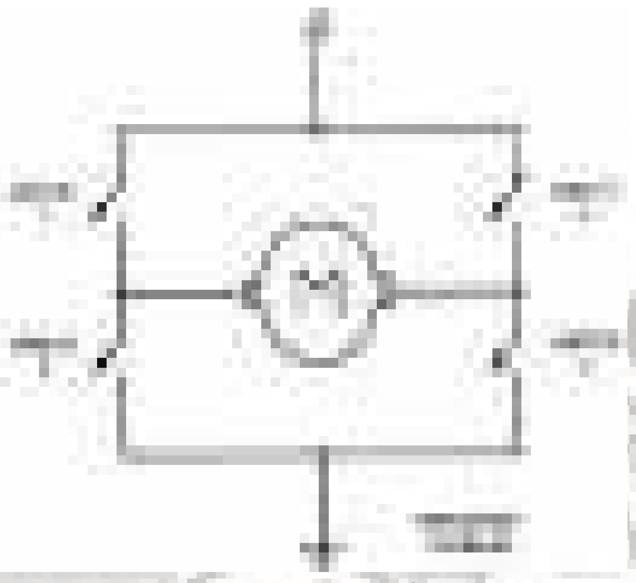


Figure 5.14 H-Bridge Motor Configuration

- Figure 5.2 shows the connection of an H-Bridge using simple switches.
- All the switches are open, which does not allow the motor to turn.

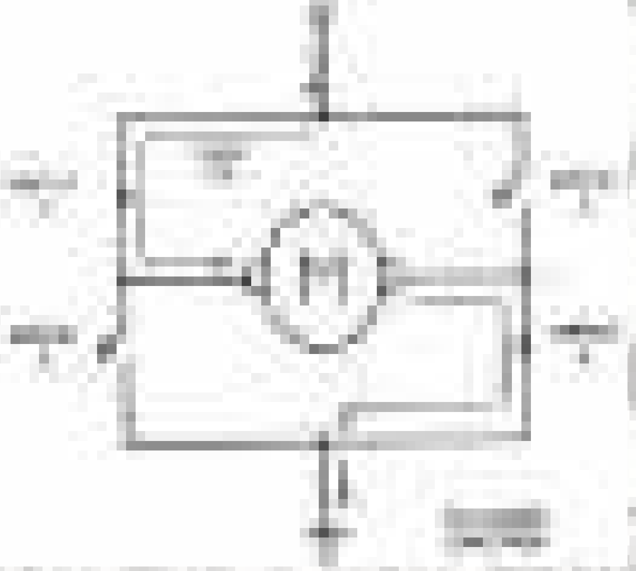


Figure 5.15 H-Bridge Motor Clockwise Configuration

- Figure 5.3 shows the switch configuration for turning the motor in one direction.
- When switches 1 and 4 are closed, current is allowed to pass through the motor.

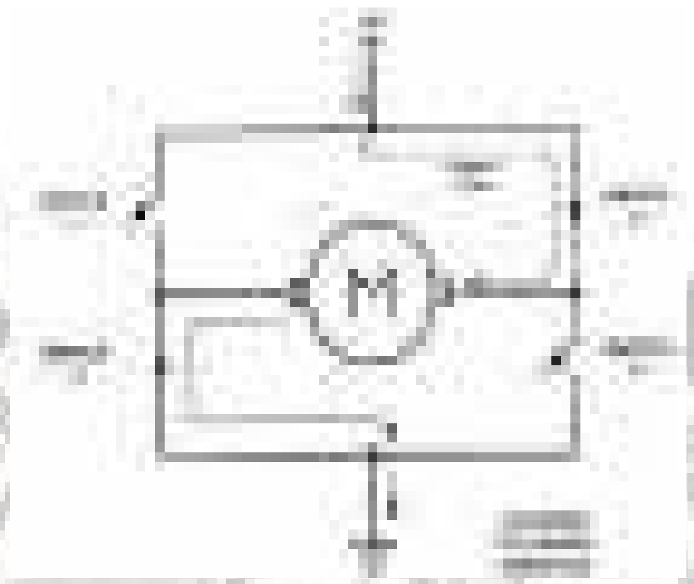


Figure 5.16 H-Bridge Motor Counterclockwise Configuration

- Figure 5.3 shows the switch configuration for turning the motor in the opposite direction from the configuration of Figure 5.3
- When switches 2 and 3 are closed, current is allowed to pass through the motor.

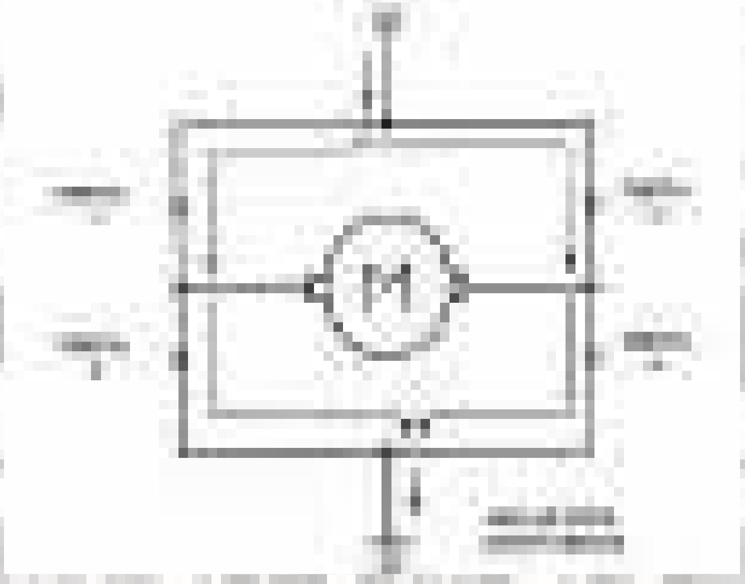


Figure 5.17 H-Bridge in an invalid configuration.

- Figure 5.4 shows an invalid configuration.
- Current flows directly to ground, creating a short circuit.
- The same effect occurs when switches 1 and 3 are closed or switches 2 and 4 are closed.
- Table 5.10 shows some of the logic configurations for the H-Bridge design.

Table 5.10 H-Bridge Logic Configurations

Motor Operation	**SW1**	**SW2**	**SW3**	**SW4**
OFF	Open	Open	Open	Open
Clockwise	Closed	Open	Open	Closed
Counter Clockwise	Open	Closed	Closed	Open
Invalid	Closed	Closed	Closed	Closed

- H-Bridge control can be created using relays, transistors, or a single IC Solution such as the L293.
- When using relays and transistors, must ensure that invalid configuration do not occur.
- **Example:**

  A switch is connected to pin P2.7. Write a program to monitor the status of SW and perform the following:

  (a) If SW=0, the DC motor moves clockwise

  (b) If SW=1, the DC motor moves counterclockwise

  Solution:

```
            ORG 0H
MAIN:       CLR P1.0                ; Switch 1
            CLR P1.1                ; Switch 2
            CLR P1.2                ; Switch 3
            CLR P1.3                ; Switch 4
            SETB P2.7
MONITOR:    JNB P2.7, CLOCKWISE
            SETB P1.0               ; Switch 1
            CLR P1.1                ; Switch 2
            CLR P1.2                ; Switch 3
            SETB P1.3               ; Switch 4
            SJMP MONITOR
CLOCKWISE:  CLR P1.0                ; Switch 1
            SETB P1.1               ; Switch 2
            SETB P1.2               ; Switch 3
            CLR P1.3                ; Switch 4
            SJMP MONITOR
            END
```

Motor Control Using L293

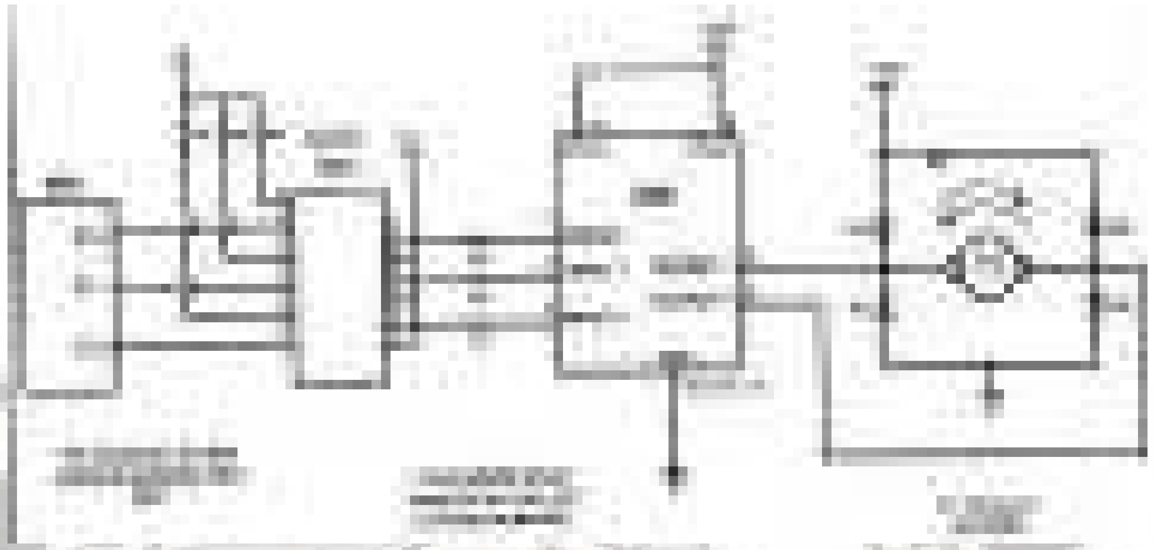


Figure 5.18 Bidirectional Motor Control Using L293 Chip

- Figure 5.18 shows the connection of L293 to an 8051.
- **Example:**

  A switch is connected to pin P2.7. Write a program to monitor the status of SW and perform the following:

  (a) If SW=0, the DC motor moves clockwise

  (b) If SW=1, the DC motor moves counterclockwise

  Solution:

```
ORG 0H
        MAIN:       CLR P1.0
                    CLR P1.1
                    CLR P1.2
                    SETB P2.7
        MONITOR:    SETB P1.0                ; Enable the Chip
                    JNB P2.7, CLOCKWISE
                    CLR P1.1                 ; Turn Motor counterclockwise
                    SETB P1.2
                    SJMP MONITOR
        CLOCKWISE:  SETB P1.1
                    CLR P1.2                 ; Turn Motor clockwise
                    SJMP MONITOR
                    END
```

PWM:

- The speed of the motor depends on three factors
  - Load
  - Voltage
  - Current
- For a given fixed load we can maintain a steady speed by using a method called Pulse Width Modulation(PWM)

- By changing (modulating) the width of the pulse applied to the DC motor we can increase or decrease the amount of power provided to the motor, thereby increasing or decreasing the motor speed.
- Notice that although the voltage has a fixed amplitude, it has a variable duty cycle
- That means the wider the pulse, the higher the speed.
- PWM is do widely used in DC motor control that some microcontrollers come with the PWM circuitry embedded in the chip.

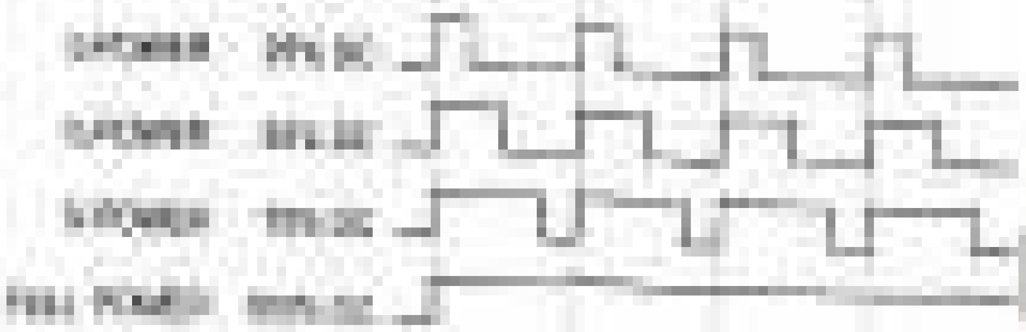


Figure 5.19 Pulse Width Modulation Comparison

Optoisolator:

- An **optoisolator** (also known as optical coupler, optocoupler and **opto-isolator**) is a semiconductor device that uses a short optical transmission path to transfer an electrical signal between circuits or elements of a circuit, while keeping them electrically isolated from each other.
- Advantage: Their high electrical isolation between the input and output terminals allowing relatively small digital signals to **control** much large AC voltages, currents and power.

Reference:

- Muhammed Ali Mazidi, Janice Gillispie Mazidi and Rolin D.McKinlay, “The 8051 Microcontroller and Embedded Systems: Using Assembly and C”

INTERFACING WITH 8051

- Interfacing of 8051 with
  - Relay
  - PWM Generator
  - DC Motor
  - Stepper Motor

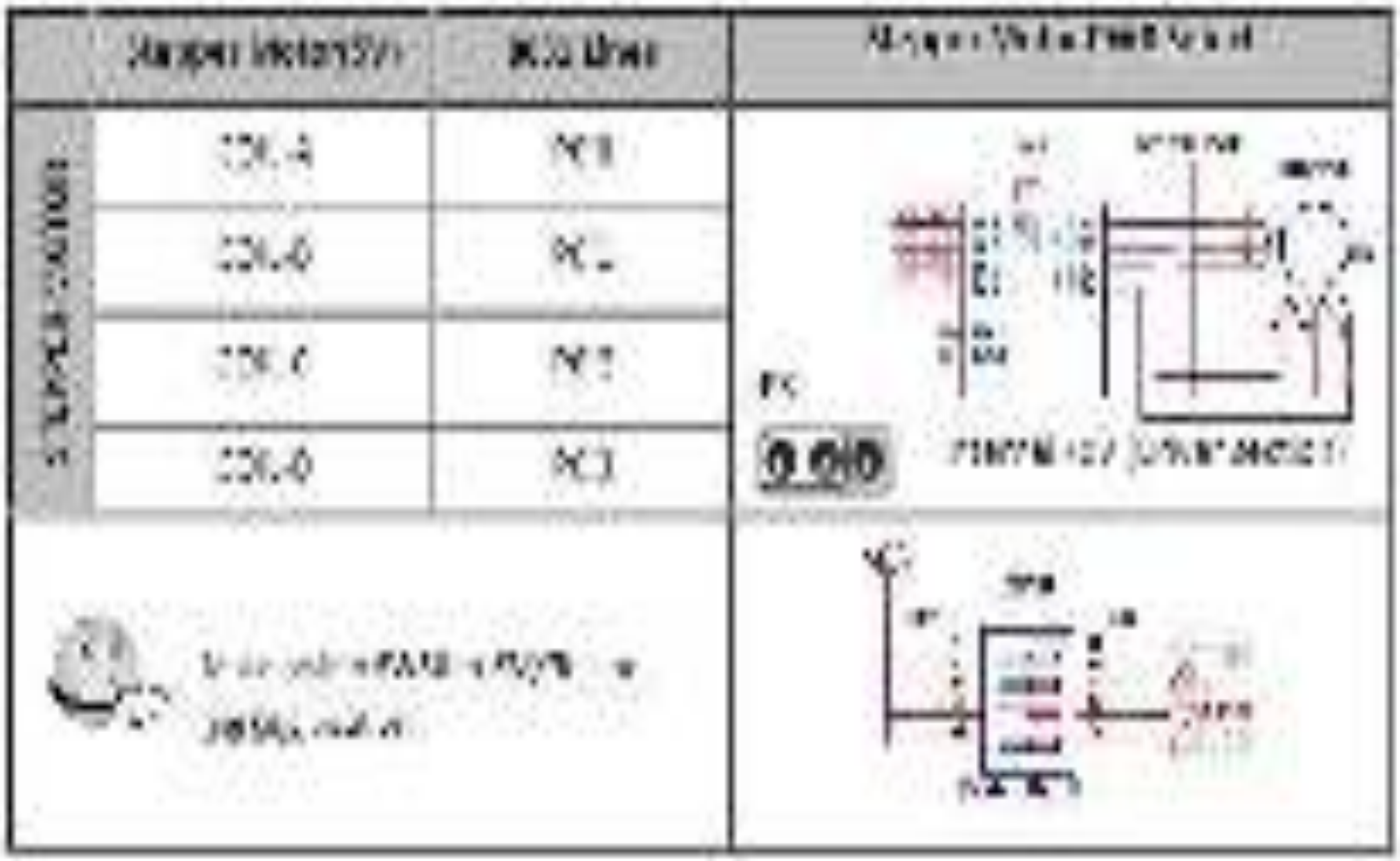
STEPPER MOTOR

- A *stepper motor* is an electromechanical device which converts electrical pulses into discrete mechanical movements or steps.
- This motor divides a full rotation of 360 degrees into a number of equal steps.

Interfacing Stepper Motor with 8051

- We now want to control a stepper motor in 8051 . It works by turning ON & OFF a four I/O port lines generating at a particular frequency.
- The 8051 has four numbers of I/O port lines, connected with I/O Port lines (P0.0 – P0.3) to rotate the stepper motor.
- ULN2003 is a high voltage and high current Darlington array IC.
- ULN2003 is used as a driver for port I/O lines, drivers output connected to stepper motor, connector provided for external power supply if needed.

Pin Assignment with 8051



	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	
	[illegible]	[illegible]	
	[illegible]	[illegible]	
[illegible]			[illegible]

C Program to control stepper motor using 8051

```
#include <reg51.h>                              //Define 8051 registers
#include<stdio.h>
void DelayMs(unsigned int);                     //Delay function\

//----------------------------------

//      Main Program

//----------------------------------
void Clockwise (void)
{
        unsigned int i;
        for (i=0;i<30;i++)
        {
                P0 = 0x01;DelayMs(5);               //Delay 20msec
                P0 = 0x02;DelayMs(5);
                P0 = 0x04;DelayMs(5);
                P0 = 0x08;DelayMs(5);
        }
}
```

C Program to control stepper motor using 8051 Contd….

```
void AntiClockwise (void)
{
        unsigned int i;
        for (i=0;i<30;i++)
        {
                P0 = 0x08;DelayMs(5);             //Delay 20msec
                P0 = 0x04;DelayMs(5);
                P0 = 0x02;DelayMs(5);
                P0 = 0x01;DelayMs(5);
        }
}
void main (void)
{
        P0 = 0;                                   //Initialize Port0

        while(1)                                  //Loop Forever
        {
                Clockwise ();
                DelayMs (100);
                P0   =     0;
                AntiClockwise ();
                DelayMs (100);
                P0   =     0;
        }
}
```

RELAY

- A **relay** is an electrically operated switch. Relays are used where it is necessary to control a circuit by a low-power signal (with complete electrical isolation between control and controlled circuits), or where several circuits must be controlled by one signal.
- A relay opens and closes under control of another electrical circuit. It is therefore connected to output pins of the microcontroller and used to turn on/off high-power devices such as motors, transformers, heaters, bulbs, antenna systems etc Relay is connected to port 1.0

C Program to control relay using 8051

```
#include<stdio.h>
sbit relay_pin = P2^0;
void Delay_ms(int);
void main()
{
do
 {
                relay_pin = 1; //Relay ON
                Delay_ms(1000);
                relay_pin = 0; //Relay OFF
                Delay_ms(1000);
}while(1);
}
 void Delay_ms(int k)
 {              int j; int i;
                for(i=0;i<k;i++)
                {
                 for(j=0;j<100;j++)
                { }
                 }
}
```

Interfacing 8051 with PWM GENERATOR

- **Pulse-width modulation (PWM), or pulse-duration modulation (PDM),** is a modulation technique that conforms the width of the pulse,the pulse duration, based on modulator signal information. Although this modulation technique can be used to encode information for transmission, its main use is to allow the control of the power supplied to electrical devices, especially to inertial loads such as motors.

- The term *duty cycle* describes the proportion of 'on' time to the regular interval or 'period' of time; a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is expressed in percent, 100% being fully on.

Interfacing 8051 with PWM GENERATOR Contd…

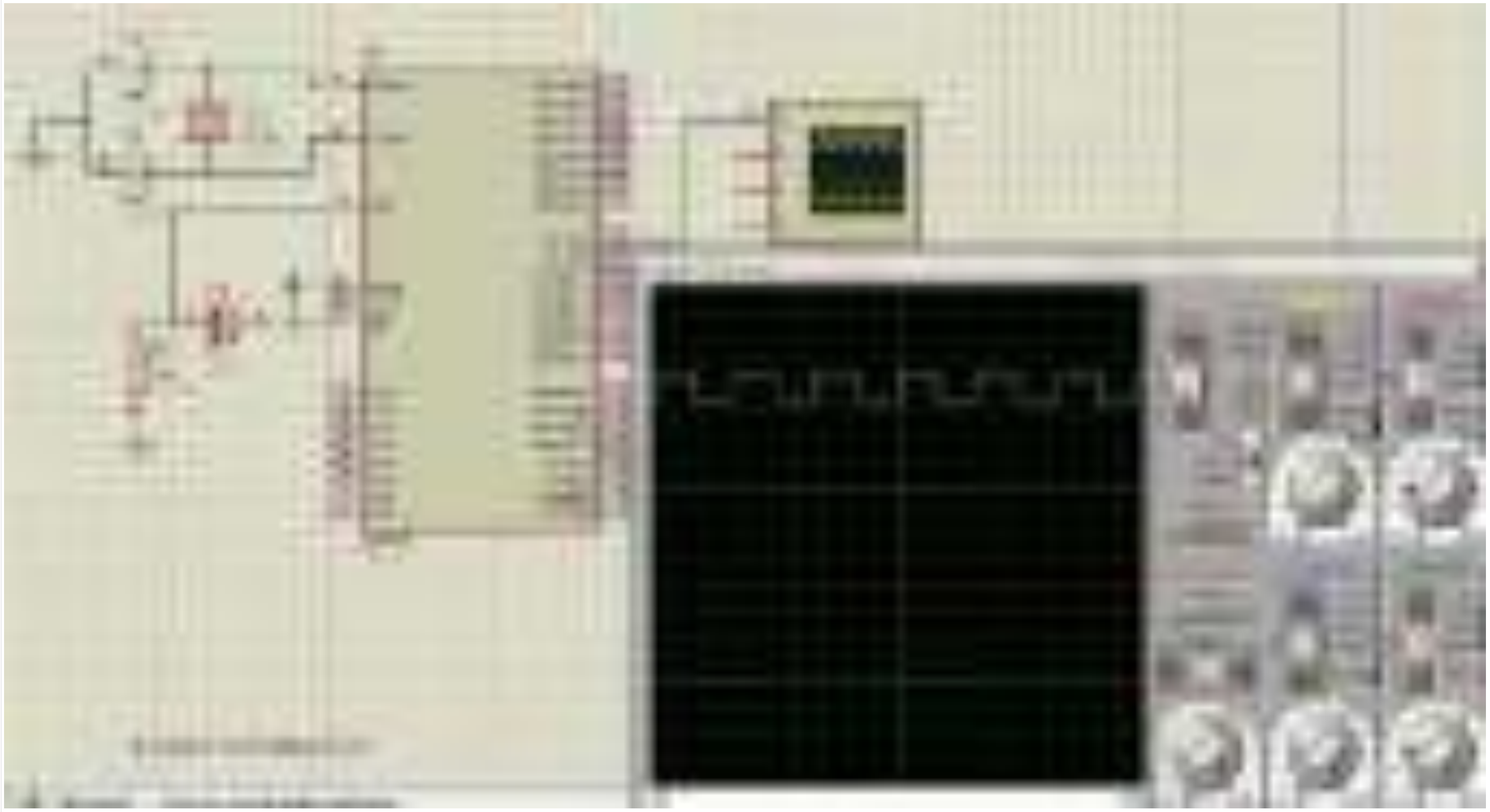
- By changing the width of the pulse applied to the DC motor we can increase or decrease the amount of power to the motor and increase or decrease the motor speed.
- The voltage have fixed amplitude but variable duty cycle.If the pulse is wider then the speed will be higher and if the pulse is smaller then the speed will be lower.

Interfacing 8051 with PWM GENERATOR Contd…

- If the PWM circuitry is embedded in the chip, we have to load the registers with high and low pulses then microcontroller will take remaining controls.

- If the PWM circuitry is not embedded in the chip, we have to create various duty cycle pulses using software.

Interfacing 8051 with PWM GENERATOR

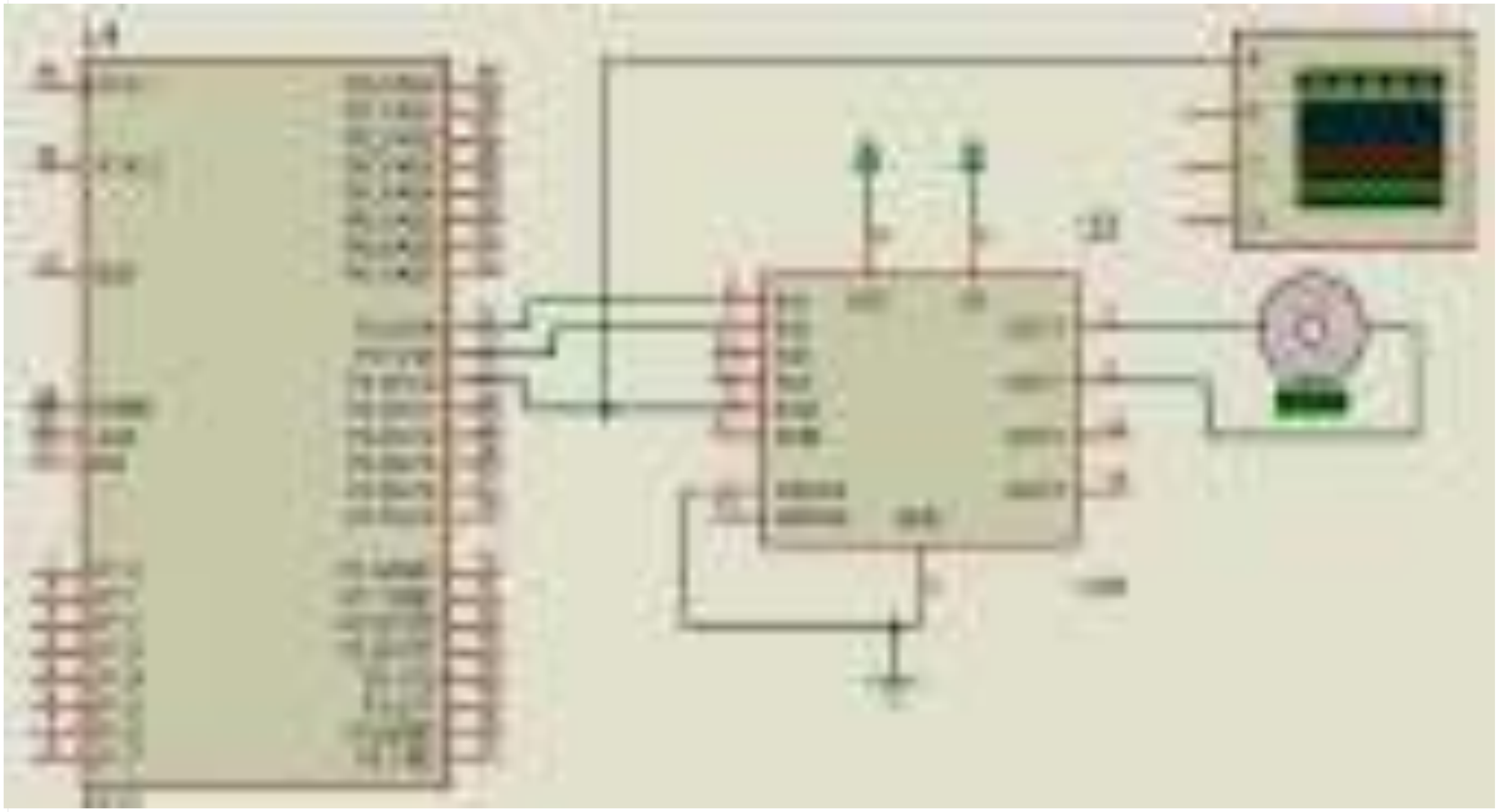


Interfacing 8051 with DC motor

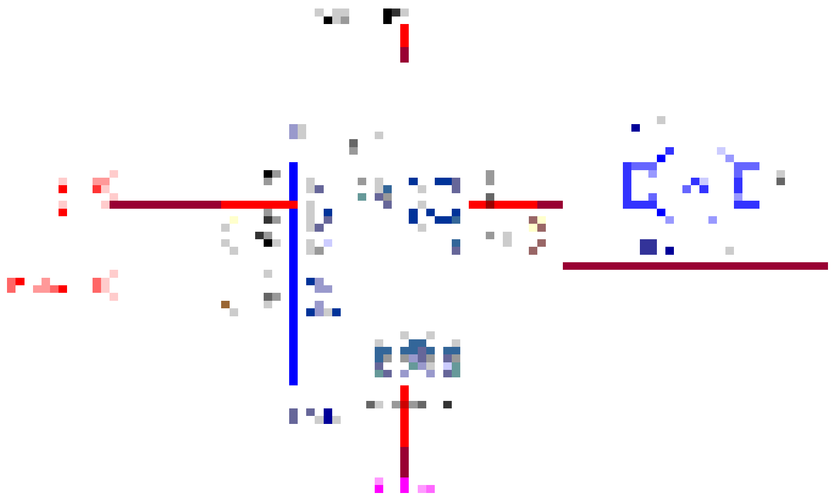
- **DC** Motors are small, inexpensive and powerful motors used widely in robotics for their small size and high energy out. A typical **DC motor** operates at speeds that are far too high speed to be useful, and torque that are too low. Gear's reduce the speed of **motor** and increases the torque.

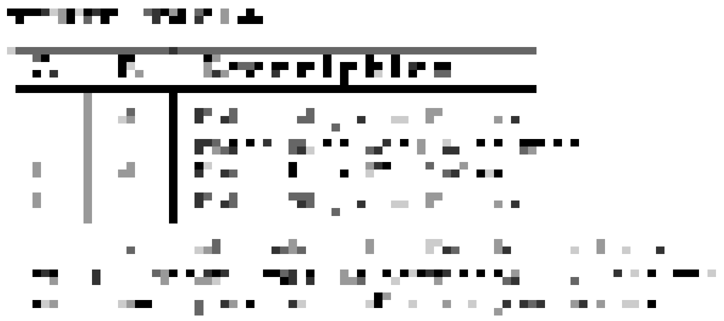
- The maximum current that can be sourced or sunk from a 8051 microcontroller is 15 mA at 5v. But a DC Motor need currents very much more than that and it need voltages 6v, 12v, 24v etc, depending upon the type of motor used. Another problem is that the back emf produced by the motor may affect the proper functioning of the microcontroller. Due to these reasons we can't connect a DC Motor directly to a microcontroller.

Interfacing 8051 with DC motor



DESCRIPTION OF DRIVER CIRCUIT





[illegible]

PROGRAM TO INTERFACE DC MOTOR WITH 8051

```
#include<reg52.h>
 #include<stdio.h>
void delay(void);
sbit motor_pin_1 = P2^0;
 sbit motor_pin_2 = P2^1;
void main()
 {
do {
		motor_pin_1 = 1;
		motor_pin_2 = 0; //Rotates Motor Anti Clockwise
		delay();
		 motor_pin_1 = 1;
		 motor_pin_2 = 1; //Stops Motor
		delay();
		motor_pin_1 = 0;
```

Contd…

```
            motor_pin_2 = 1; //Rotates Motor Clockwise
            delay();
            motor_pin_1 = 0;
            motor_pin_2 = 0; //Stops Motor
            delay();
    }
    while(1);
}
void delay()
{
int i,j; for(i=0;i<1000;i++)
{
for(j=0;j<1000;j++)
{ }
 }
}
```

Control Signals and Motor Status

P2.0/IN1	P2.1/IN2	Motor Status
LOW	LOW	Stops
LOW	HIGH	Clockwise
HIGH	LOW	Anti-Clockwise
HIGH	HIGH	Stops

Implementation of Traffic Light Controller using 8051

- Traffic lights or traffic signals are signaling devices positioned at road intersections, pedestrian crossings and other locations to control competing flows of traffic.

Implementation of traffic light controller using 8051

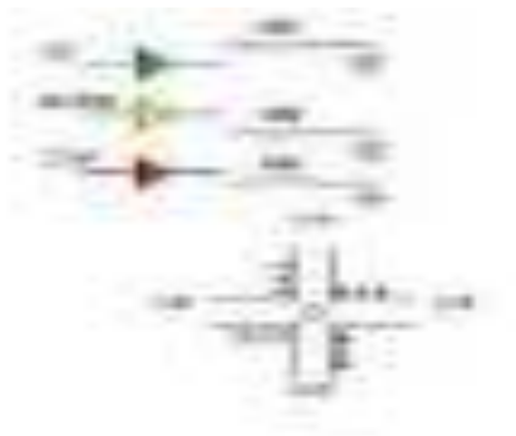
- Traffic lights alternate the right of way of road users by displaying lights of a standard color (red, yellow/amber, and green), using a universal color code.
  In the typical sequence of colored lights:

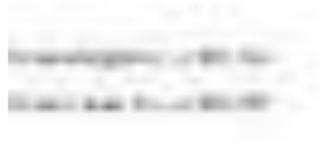
  Illumination of the green light allows traffic to proceed in the direction denoted,
- Illumination of the yellow/amber light denoting, if safe to do so, prepare to stop short of the intersection, and
- Illumination of the red signal prohibits any traffic from proceeding.

PIN ASSIGNMENT WITH 8051

LAN Direction	8051 Lines	LED’s
	P3.2	D8-Stop
NORTH	P3.3	D9-Listen
	P3.4	D10-Go
	P3.5	D11-Stop
WEST	P3.6	D12-Listen
	P3.7	D13-Go
	P1.0	D14-Stop
SOUTH	P1.1	D15-Listen
	P1.2	D16-Go
	P1.3	D17-Stop
EAST	P1.4	D18-Listen
	P1.5	D19-Go

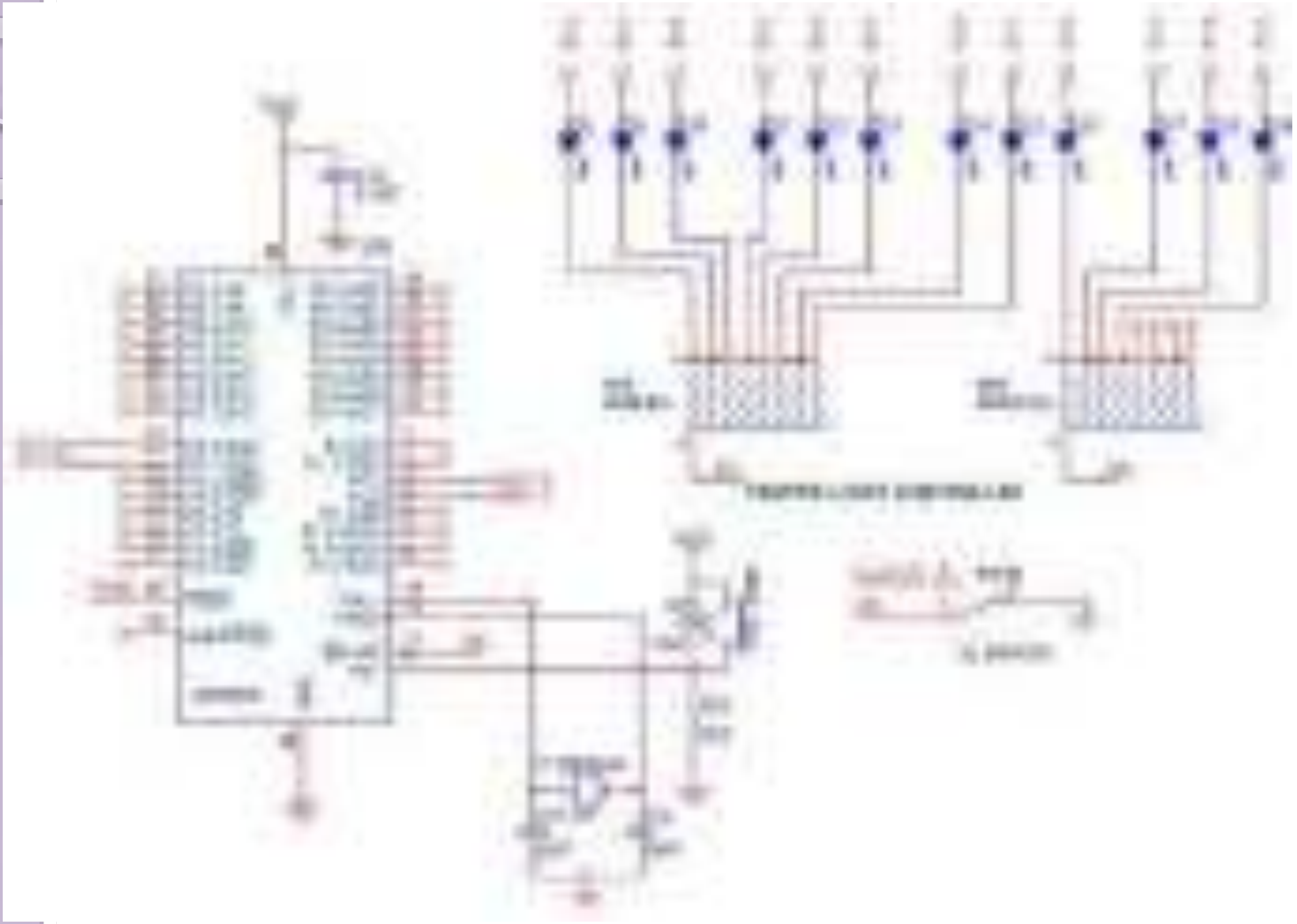
Traffic Light Controller




[illegible]

Note : Make **SW32** to “***Traffic”*** label marking position

CIRCUIT DIAGRAM TO INTERFACE TRAFFIC LIGHT



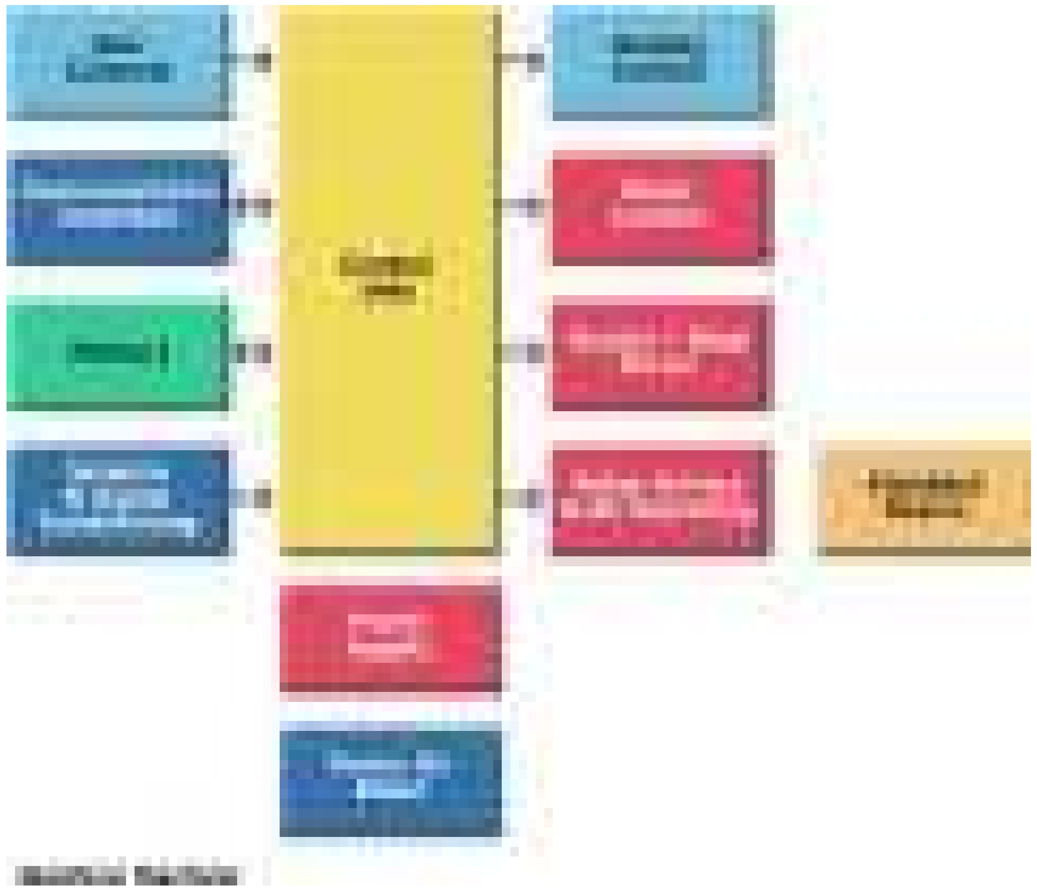
Program to control traffic signals

```
#include <reg51.h>
sbit RA =    P1^0;
sbit YA  =   P1^1;
sbit GA  =   P1^2;
sbit RB  =   P3^2;
sbit YB  =   P3^3;
sbit GB  =   P3^4;
sbit RC  =   P3^5;
sbit YC  =   P3^6;
sbit GC  =   P3^7;
sbit rD  =   P1^3;
sbit YD  =   P1^4;
sbit GD  =   P1^5;
```

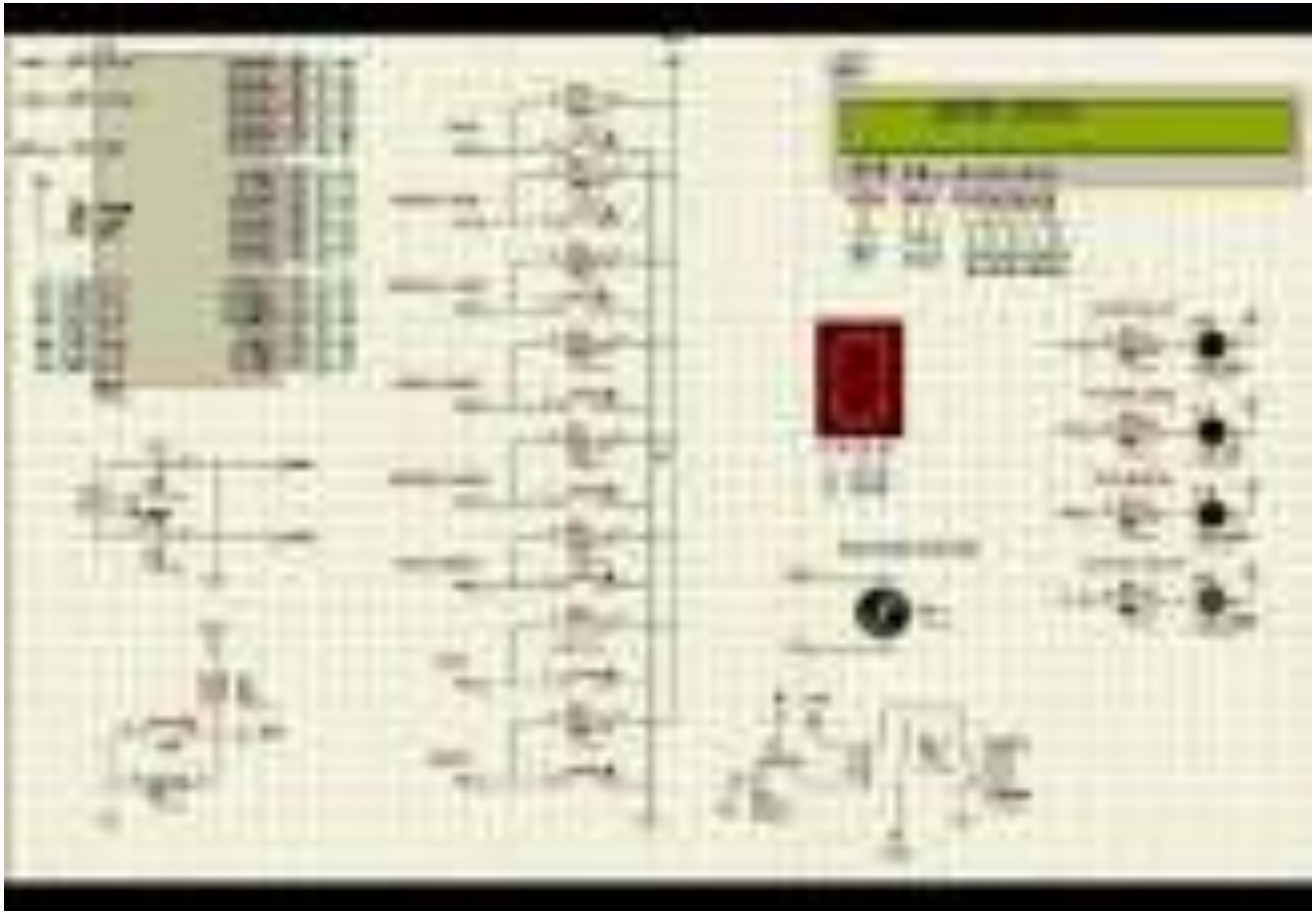
```
void Delay (void)
{
    unsigned int i,j;
    for (i=0;i<200;i++)
         for (j=0;j<500;j++);
}
void SuperDelay ()
{
    unsigned int i;
    for (i=0;i<25;i++)
         Delay();
}
```

```
void main ()
{
   P3  =   0;
   while (1)
   {
       RA  =   0; GA  =   1;  YA  =   0;
       RB  =   1;  GB  =   0;  YB  =   0;
       RC  =   1;  GC  =   0;  YC  =   0;
       rD  =   1;  GD  =   0;  YD  =   0;

       SuperDelay();

       GA  =   0;  YA  =   1;
       Delay();
       RA  =   1; GA  =   0;  YA  =   0;
       RB  =   0;  GB  =   1;  YB  =   0;
       RC  =   1;  GC  =   0;  YC  =   0;
       rD  =   1;  GD  =   0;  YD  =   0;
      SuperDelay ();
GB  =   0;  YB  =   1;
       Delay ();
       RA  =   1; GA  =   0;  YA  =   0;
       RB  =   1;  GB  =   0;  YB  =   0;
       RC  =   0;  GC  =   1;  YC  =   0;
       rD  =   1;  GD  =   0;  YD  =   0;
SuperDelay ();
       GC  =   0;  YC  =   1;
       Delay();
       RA  =   1; GA  =   0;  YA  =   0;
       RB  =   1;  GB  =   0;  YB  =   0;
       RC  =   1;  GC  =   0;  YC  =   0;
       rD  =   0;  GD  =   1;  YD  =   0;
SuperDelay ();
       GD  =   0;  YD  =   1;
       Delay();

   }
}
```

Washing machine control



Interfacing washing machine control with 8051



Control flow

Interfacing I2C - Inter integrated circuit serial bus

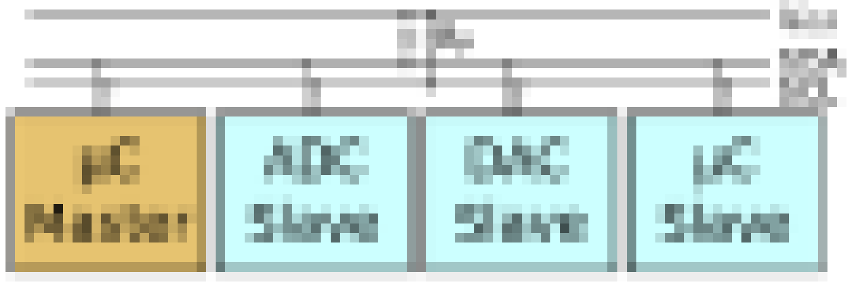
- At any time the electrical power is turned off, the Microcontroller will start from initial state after power is turned on, losing all intermediate states.

- To avoid this problem a Serial EEPROM is
- interfaced to the microcontroller via I2C serial BUS.

- This is a two wire bus using a protocol to transfer data between microcontroller and serial memory and save the status of process event sequentially

Data Transfer from master to slave

- A master device sends the sequence of signal with START, ADDRESS, and WRITE protocol, then waits for an acknowledge bit (A) from the slave
- The slave will generate acknowledge bit only if its internal address matches the value sent by the master.
- After that the master sends DATA and waits for acknowledge (A) from the slave. The master completes the byte transfer by generating a stop bit (P)



Data transfer sequence

- A Master to slave read or write sequence for I2C follows the following order:
- 1. Send the START bit (S).
- 2. Send the address (ADDR).
- 3. Send the Read(R)-1 / Write (W)-0 bit.
- 4. Wait for/Send an acknowledge bit (A).
- 5. Send/Receive the data byte (8 bits) (DATA).
- 6. Expect/Send acknowledge bit (A).
- 7. Send STOP bit (P).

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

IoT (Internet of Things) is an advanced automation and analytics system which exploits networking, sensing, big data, and artificial intelligence technology to deliver complete systems for a product or service. These systems allow greater transparency, control, and performance when applied to any industry or system.

IoT systems have applications across industries through their unique flexibility and ability to be suitable in any environment. They enhance data collection, automation, operations, and much more through smart devices and powerful enabling technology.

This tutorial aims to provide you with a thorough introduction to IoT. It introduces the key concepts of IoT, necessary in using and deploying IoT systems.

Audience

This tutorial targets IT professionals, students, and management professionals who want a solid grasp of essential IoT concepts. After completing this tutorial, you will achieve intermediate expertise in IoT and a high level of comfort with IoT concepts and systems.

Prerequisites

This tutorial assumes general knowledge of networking, sensing, databases, programming, and related technology. It also assumes familiarity with business concepts and marketing.

Copyright & Disclaimer

Table of Contents

1. IoT – Overview

IoT systems allow users to achieve deeper automation, analysis, and integration within a system. They improve the reach of these areas and their accuracy. IoT utilizes existing and emerging technology for sensing, networking, and robotics.

IoT exploits recent advances in software, falling hardware prices, and modern attitudes towards technology. Its new and advanced elements bring major changes in the delivery of products, goods, and services; and the social, economic, and political impact of those changes.

IoT – Key Features

The most important features of IoT include artificial intelligence, connectivity, sensors, active engagement, and small device use. A brief review of these features is given below:

- **AI –** IoT essentially makes virtually anything "smart", meaning it enhances every aspect of life with the power of data collection, artificial intelligence algorithms, and networks. This can mean something as simple as enhancing your refrigerator and cabinets to detect when milk and your favorite cereal run low, and to then place an order with your preferred grocer.
- **Connectivity –** New enabling technologies for networking, and specifically IoT networking, mean networks are no longer exclusively tied to major providers. Networks can exist on a much smaller and cheaper scale while still being practical. IoT creates these small networks between its system devices.
- **Sensors –** IoT loses its distinction without sensors. They act as defining instruments which transform IoT from a standard passive network of devices into an active system capable of real-world integration.
- **Active Engagement –** Much of today's interaction with connected technology happens through passive engagement. IoT introduces a new paradigm for active content, product, or service engagement.
- **Small Devices –** Devices, as predicted, have become smaller, cheaper, and more powerful over time. IoT exploits purpose-built small devices to deliver its precision, scalability, and versatility.

IoT – Advantages

The advantages of IoT span across every area of lifestyle and business. Here is a list of some of the advantages that IoT has to offer:

- **Improved Customer Engagement –** Current analytics suffer from blind-spots and significant flaws in accuracy; and as noted, engagement remains passive. IoT completely transforms this to achieve richer and more effective engagement with audiences.
- **Technology Optimization –** The same technologies and data which improve the customer experience also improve device use, and aid in more potent improvements to technology. IoT unlocks a world of critical functional and field data.

- **Reduced Waste –** IoT makes areas of improvement clear. Current analytics give us superficial insight, but IoT provides real-world information leading to more effective management of resources.
- **Enhanced Data Collection –** Modern data collection suffers from its limitations and its design for passive use. IoT breaks it out of those spaces, and places it exactly where humans really want to go to analyze our world. It allows an accurate picture of everything.

IoT – Disadvantages

Though IoT delivers an impressive set of benefits, it also presents a significant set of challenges. Here is a list of some its major issues:

- **Security –** IoT creates an ecosystem of constantly connected devices communicating over networks. The system offers little control despite any security measures. This leaves users exposed to various kinds of attackers.
- **Privacy –** The sophistication of IoT provides substantial personal data in extreme detail without the user's active participation.
- **Complexity –** Some find IoT systems complicated in terms of design, deployment, and maintenance given their use of multiple technologies and a large set of new enabling technologies.
- **Flexibility –** Many are concerned about the flexibility of an IoT system to integrate easily with another. They worry about finding themselves with several conflicting or locked systems.
- **Compliance –** IoT, like any other technology in the realm of business, must comply with regulations. Its complexity makes the issue of compliance seem incredibly challenging when many consider standard software compliance a battle.

2. IoT – Hardware

The hardware utilized in IoT systems includes devices for a remote dashboard, devices for control, servers, a routing or bridge device, and sensors. These devices manage key tasks and functions such as system activation, action specifications, security, communication, and detection to support-specific goals and actions.

IoT – Sensors

The most important hardware in IoT might be its sensors. These devices consist of energy modules, power management modules, RF modules, and sensing modules. RF modules manage communications through their signal processing, WiFi, ZigBee, Bluetooth, radio transceiver, duplexer, and BAW.

The sensing module manages sensing through assorted active and passive measurement devices. Here is a list of some of the measurement devices used in IoT:

Devices	
accelerometers	temperature sensors
magnetometers	proximity sensors
gyroscopes	image sensors
acoustic sensors	light sensors
pressure sensors	gas RFID sensors
humidity sensors	micro flow sensors

Wearable Electronics

Wearable electronic devices are small devices worn on the head, neck, arms, torso, and feet.

Smartwatches not only help us stay connected, but as a part of an IoT system, they allow access needed for improved productivity.

Current smart wearable devices include:

- **Head** – Helmets, glasses
- **Neck** – Jewelry, collars
- **Arm** – Watches, wristbands, rings
- **Torso** – Clothing, backpacks
- **Feet** – Socks, shoes

Smart glasses help us enjoy more of the media and services we value, and when part of an IoT system, they allow a new approach to productivity.

Standard Devices

The desktop, tablet, and cellphone remain integral parts of IoT as the command center and remotes.

- The **desktop** provides the user with the highest level of control over the system and its settings.
- The **tablet** provides access to the key features of the system in a way resembling the desktop, and also acts as a remote.
- The **cellphone** allows some essential settings modification and also provides remote functionality.

Other key connected devices include standard network devices like **routers** and **switches**.

3. IoT – Software

IoT software addresses its key areas of networking and action through platforms, embedded systems, partner systems, and middleware. These individual and master applications are responsible for data collection, device integration, real-time analytics, and application and process extension within the IoT network. They exploit integration with critical business systems (e.g., ordering systems, robotics, scheduling, and more) in the execution of related tasks.

Data Collection

This software manages sensing, measurements, light data filtering, light data security, and aggregation of data. It uses certain protocols to aid sensors in connecting with real-time, machine-to-machine networks. Then it collects data from multiple devices and distributes it in accordance with settings. It also works in reverse by distributing data over devices. The system eventually transmits all collected data to a central server.

Device Integration

Software supporting integration binds (dependent relationships) all system devices to create the body of the IoT system. It ensures the necessary cooperation and stable networking between devices. These applications are the defining software technology of the IoT network because without them, it is not an IoT system. They manage the various applications, protocols, and limitations of each device to allow communication.

Real-Time Analytics

These applications take data or input from various devices and convert it into viable actions or clear patterns for human analysis. They analyze information based on various settings and designs in order to perform automation-related tasks or provide the data required by industry.

Application and Process Extension

These applications extend the reach of existing systems and software to allow a wider, more effective system. They integrate predefined devices for specific purposes such as allowing certain mobile devices or engineering instruments access. It supports improved productivity and more accurate data collection.

4. IoT – Technology and Protocols

IoT primarily exploits standard protocols and networking technologies. However, the major enabling technologies and protocols of IoT are RFID, NFC, low-energy Bluetooth, low-energy wireless, low-energy radio protocols, LTE-A, and WiFi-Direct. These technologies support the specific networking functionality needed in an IoT system in contrast to a standard uniform network of common systems.

NFC and RFID

RFID (radio-frequency identification) and NFC (near-field communication) provide simple, low-energy, and versatile options for identity and access tokens, connection bootstrapping, and payments.

- RFID technology employs 2-way radio transmitter-receivers to identify and track tags associated with objects.

- NFC consists of communication protocols for electronic devices, typically a mobile device and a standard device.

Low-Energy Bluetooth

This technology supports the low-power, long-use need of IoT function while exploiting a standard technology with native support across systems.

Low-Energy Wireless

This technology replaces the most power hungry aspect of an IoT system. Though sensors and other elements can power down over long periods, communication links (i.e., wireless) must remain in listening mode. Low-energy wireless not only reduces consumption, but also extends the life of the device through less use.

Radio Protocols

ZigBee, Z-Wave, and Thread are radio protocols for creating low-rate private area networks. These technologies are low-power, but offer high throughput unlike many similar options. This increases the power of small local device networks without the typical costs.

LTE-A

LTE-A, or LTE Advanced, delivers an important upgrade to LTE technology by increasing not only its coverage, but also reducing its latency and raising its throughput. It gives IoT a tremendous power through expanding its range, with its most significant applications being vehicle, UAV, and similar communication.

WiFi-Direct

WiFi-Direct eliminates the need for an access point. It allows P2P (peer-to-peer) connections with the speed of WiFi, but with lower latency. WiFi-Direct eliminates an element of a network that often bogs it down, and it does not compromise on speed or throughput.

5. IoT – Common Uses

IoT has applications across all industries and markets. It spans user groups from those who want to reduce energy use in their home to large organizations who want to streamline their operations. It proves not just useful, but nearly critical in many industries as technology advances and we move towards the advanced automation imagined in the distant future.

Engineering, Industry, and Infrastructure

Applications of IoT in these areas include improving production, marketing, service delivery, and safety. IoT provides a strong means of monitoring various processes; and real transparency creates greater visibility for improvement opportunities.

The deep level of control afforded by IoT allows rapid and more action on those opportunities, which include events like obvious customer needs, nonconforming product, malfunctions in equipment, problems in the distribution network, and more.

Example

Joan runs a manufacturing facility that makes shields for manufacturing equipment. When regulations change for the composition and function of the shields, the new appropriate requirements are automatically programmed in production robotics, and engineers are alerted about their approval of the changes.

Government and Safety

IoT applied to government and safety allows improved law enforcement, defense, city planning, and economic management. The technology fills in the current gaps, corrects many current flaws, and expands the reach of these efforts. For example, IoT can help city planners have a clearer view of the impact of their design, and governments have a better idea of the local economy.

Example

Joan lives in a small city. She's heard about a recent spike in crime in her area, and worries about coming home late at night.

Local law enforcement has been alerted about the new "hot" zone through system flags, and they've increases their presence. Area monitoring devices have detected suspicious behavior, and law enforcement has investigated these leads to prevent crimes.

Home and Office

In our daily lives, IoT provides a personalized experience from the home to the office to the organizations we frequently do business with. This improves our overall satisfaction, enhances productivity, and improves our health and safety. For example, IoT can help us customize our office space to optimize our work.

Example

Joan works in advertising. She enters her office, and it recognizes her face. It adjusts the lighting and temperature to her preference. It turns on her devices and opens applications to her last working points.

Her office door detected and recognized a colleague visiting her office multiple times before she arrived. Joan's system opens this visitor's messages automatically.

Health and Medicine

IoT pushes us towards our imagined future of medicine which exploits a highly integrated network of sophisticated medical devices. Today, IoT can dramatically enhance medical research, devices, care, and emergency care. The integration of all elements provides more accuracy, more attention to detail, faster reactions to events, and constant improvement while reducing the typical overhead of medical research and organizations.

Example

Joan is a nurse in an emergency room. A call has come in for a man wounded in an altercation. The system recognized the patient and pulls his records. On the scene, paramedic equipment captures critical information automatically sent to the receiving parties at the hospital. The system analyzes the new data and current records to deliver a guiding solution. The status of the patient is updated every second in the system during his transport. The system prompts Joan to approve system actions for medicine distribution and medical equipment preparation.

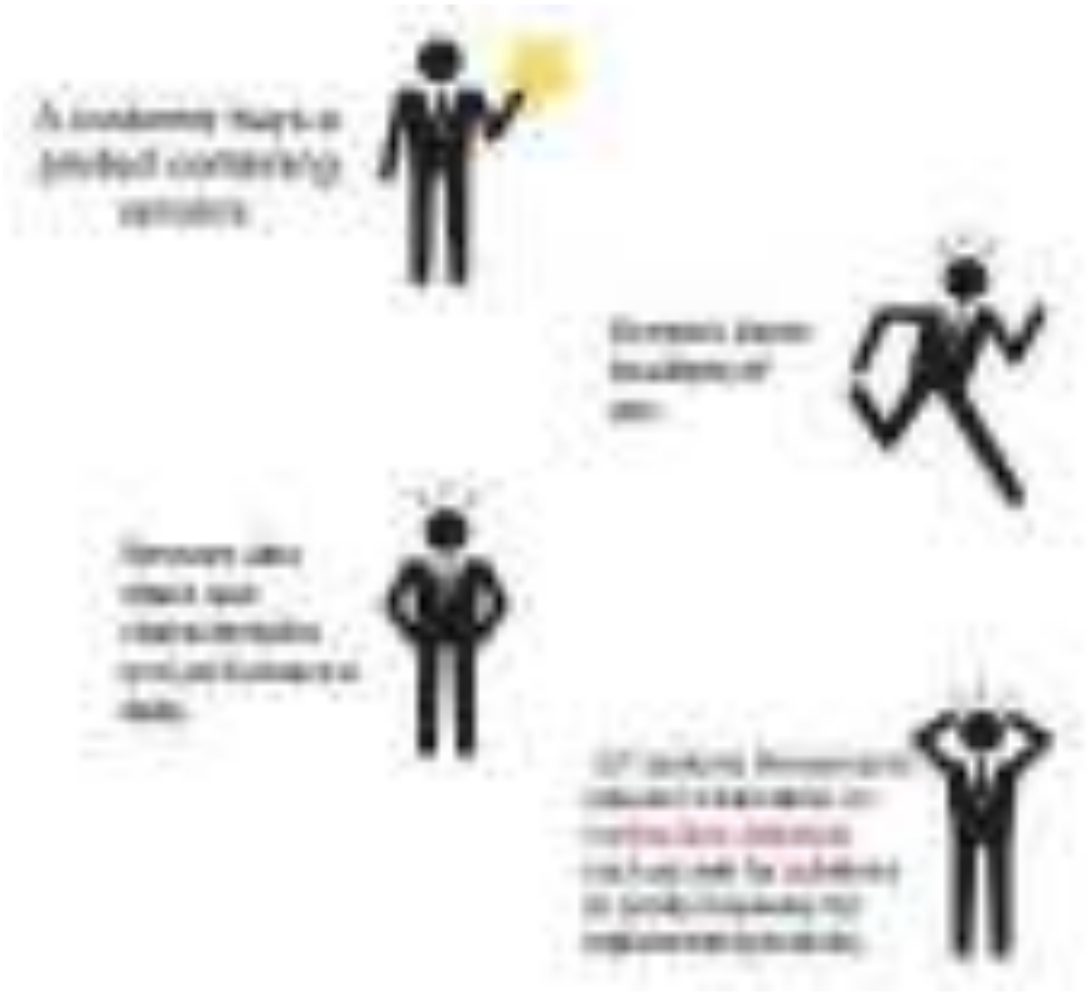
6. IoT – Media, Marketing, & Advertising

The applications of IoT in media and advertising involve a customized experience in which the system analyzes and responds to the needs and interests of each customer. This includes their general behavior patterns, buying habits, preferences, culture, and other characteristics.

Marketing and Content Delivery

IoT functions in a similar and deeper way to current technology, analytics, and big data. Existing technology collects specific data to produce related metrics and patterns over time, however, that data often lacks depth and accuracy. IoT improves this by observing more behaviors and analyzing them differently.

- This leads to more information and detail, which delivers more reliable metrics and patterns.
- It allows organizations to better analyze and respond to customer needs or preferences.
- It improves business productivity and strategy, and improves the consumer experience by only delivering relevant content and solutions.



Improved Advertising

Current advertising suffers from excess and poor targeting. Even with today's analytics, modern advertising fails. IoT promises different and personalized advertising rather than one-size-fits-all strategies. It transforms advertising from noise to a practical part of life because consumers interact with advertising through IoT rather than simply receiving it. This makes advertising more functional and useful to people searching the marketplace for solutions or wondering if those solutions exist.

7. IoT – Environmental Monitoring

The applications of IoT in environmental monitoring are broad: environmental protection, extreme weather monitoring, water safety, endangered species protection, commercial farming, and more. In these applications, sensors detect and measure every type of environmental change.

Air and Water Pollution

Current monitoring technology for air and water safety primarily uses manual labor along with advanced instruments, and lab processing. IoT improves on this technology by reducing the need for human labor, allowing frequent sampling, increasing the range of sampling and monitoring, allowing sophisticated testing on-site, and binding response efforts to detection systems. This allows us to prevent substantial contamination and related disasters.

Extreme Weather

Though powerful, advanced systems currently in use allow deep monitoring, they suffer from using broad instruments, such as radar and satellites, rather than more granular solutions. Their instruments for smaller details lack the same accurate targeting of stronger technology.

New IoT advances promise more fine-grained data, better accuracy, and flexibility. Effective forecasting requires high detail and flexibility in range, instrument type, and deployment. This allows early detection and early responses to prevent loss of life and property.

Commercial Farming

Today's sophisticated commercial farms have exploited advanced technology and biotechnology for quite some time, however, IoT introduces more access to deeper automation and analysis.

Much of commercial farming, like weather monitoring, suffers from a lack of precision and requires human labor in the area of monitoring. Its automation also remains limited.

IoT allows operations to remove much of the human intervention in system function, farming analysis, and monitoring. Systems detect changes to crops, soil, environment, and more. They optimize standard processes through analysis of large, rich data collections. They also prevent health hazards (e.g., *e. coli*) from happening and allow better control.

8. IoT – Manufacturing Applications

Manufacturing technology currently in use exploits standard technology along with modern distribution and analytics. IoT introduces deeper integration and more powerful analytics. This opens the world of manufacturing in a way never seen before, as organizations become fully-developed for product delivery rather than a global network of suppliers, makers, and distributors loosely tied together.

Intelligent Product Enhancements

Much like IoT in content delivery, IoT in manufacturing allows richer insight in real-time. This dramatically reduces the time and resources devoted to this one area, which traditionally requires heavy market research before, during, and well after the products hit the market.

IoT also reduces the risks associated with launching new or modified products because it provides more reliable and detailed information. The information comes directly from market use and buyers rather than assorted sources of varied credibility.

Dynamic Response to Market Demands

Supplying the market requires maintaining a certain balance impacted by a number of factors such as economy state, sales performance, season, supplier status, manufacturing facility status, distribution status, and more. The expenses associated with supply present unique challenges given today's global partners. The associated potential or real losses can dramatically impact business and future decisions.

IoT manages these areas through ensuring fine details are managed more at the system level rather than through human evaluations and decisions. An IoT system can better assess and control the supply chain (with most products), whether demands are high or low.

Lower Costs, Optimized Resource Use, and Waste Reduction

IoT offers a replacement for traditional labor and tools in a production facility and in the overall chain which cuts many previously unavoidable costs; for example, maintenance checks or tests traditionally requiring human labor can be performed remotely with instruments and sensors of an IoT system.

IoT also enhances operation analytics to optimize resource use and labor, and eliminate various types of waste, e.g., energy and materials. It analyzes the entire process from the source point to its end, not just the process at one point in a particular facility, which allows improvement to have a more substantial impact. It essentially reduces waste throughout the network, and returns those savings throughout.

This XRS relay box connects all truck devices (e.g., diagnostics and driver cell) to the XRS fleet management supporting software, which allows data collection.

Improved Facility Safety

A typical facility suffers from a number of health and safety hazards due to risks posed by processes, equipment, and product handling. IoT aids in better control and visibility. Its monitoring extends throughout the network of devices for not only performance, but for dangerous malfunctions and usage. It aids (or performs) analysis and repair, or correction, of critical flaws.

Product Safety

Even the most sophisticated system cannot avoid malfunctions, nonconforming product, and other hazards finding their way to market. Sometimes these incidents have nothing to do with the manufacturing process, and result from unknown conflicts.

In manufacturing, IoT helps in avoiding recalls and controlling nonconforming or dangerous product distribution. Its high level of visibility, control, and integration can better contain any issues that appear.

9. IoT – Energy Applications

The optimization qualities of IoT in manufacturing also apply to energy consumption. IoT allows a wide variety of energy control and monitoring functions, with applications in devices, commercial and residential energy use, and the energy source. Optimization results from the detailed analysis previously unavailable to most organizations and individuals.

Residential Energy

The rise of technology has driven energy costs up. Consumers search for ways to reduce or control consumption. IoT offers a sophisticated way to analyze and optimize use not only at device level, but throughout the entire system of the home. This can mean simple switching off or dimming of lights, or changing device settings and modifying multiple home settings to optimize energy use.

IoT can also discover problematic consumption from issues like older appliances, damaged appliances, or faulty system components. Traditionally, finding such problems required the use of often multiple professionals.

Commercial Energy

Energy waste can easily and quietly impact business in a major way, given the tremendous energy needs of even small organizations. Smaller organizations wrestle with balancing costs of business while delivering a product with typically smaller margins, and working with limited funding and technology. Larger organizations must monitor a massive, complex ecosystem of energy use that offers few simple, effective solutions for energy use management.

A smart-meter still requires a reader to visit the site. This automated meter reader makes visits unnecessary, and also allows energy companies to bill based on real-time data instead of estimates over time.

IoT simplifies the process of energy monitoring and management while maintaining a low cost and high level of precision. It addresses all points of an organization's consumption across devices. Its depth of analysis and control provides organizations with a strong means of managing their consumption for cost shaving and output optimization. IoT systems discover energy issues in the same way as functional issues in a complex business network, and provide solutions.

Reliability

The analytics and action delivered by IoT also help to ensure system reliability. Beyond consumption, IoT prevents system overloads or throttling. It also detects threats to system performance and stability, which protects against losses such as downtime, damaged equipment, and injuries.

10. IoT – Healthcare Applications

IoT systems applied to healthcare enhance existing technology, and the general practice of medicine. They expand the reach of professionals within a facility and far beyond it. They increase both the accuracy and size of medical data through diverse data collection from large sets of real-world cases. They also improve the precision of medical care delivery through more sophisticated integration of the healthcare system.

Research

Much of current medical research relies on resources lacking critical real-world information. It uses controlled environments, volunteers, and essentially leftovers for medical examination. IoT opens the door to a wealth of valuable information through real-time field data, analysis, and testing.

IoT can deliver relevant data superior to standard analytics through integrated instruments capable of performing viable research. It also integrates into actual practice to provide more key information. This aids in healthcare by providing more reliable and practical data, and better leads; which yields better solutions and discovery of previously unknown issues.

It also allows researchers to avoid risks by gathering data without manufactured scenarios and human testing.

Devices

Current devices are rapidly improving in precision, power, and availability; however, they still offer less of these qualities than an IoT system integrating the right system effectively. IoT unlocks the potential of existing technology, and leads us toward new and better medical device solutions.

IoT closes gaps between equipment and the way we deliver healthcare by creating a logical system rather than a collection of tools. It then reveals patterns and missing elements in healthcare such as obvious necessary improvements or huge flaws.

The ClearProbe portable connected ultrasound device can use any computer anywhere as a supporting machine. The device sends all imaging records to the master system.

Care

Perhaps the greatest improvement IoT brings to healthcare is in the actual practice of medicine because it empowers healthcare professionals to better use their training and knowledge to solve problems. They utilize far better data and equipment, which gives them a window into blind spots and supports more swift, precise actions. Their decision-making is no longer limited by the disconnects of current systems, and bad data.

IoT also improves their professional development because they actually exercise their talent rather than spending too much time on administrative or manual tasks. Their organizational decisions also improve because technology provides a better vantage point.

Medical Information Distribution

One of the challenges of medical care is the distribution of accurate and current information to patients. Healthcare also struggles with guidance given the complexity of following guidance. IoT devices not only improve facilities and professional practice, but also health in the daily lives of individuals.

IoT devices give direct, 24/7 access to the patient in a less intrusive way than other options. They take healthcare out of facilities and into the home, office, or social space. They empower individuals in attending to their own health, and allow providers to deliver better and more granular care to patients. This results in fewer accidents from miscommunication, improved patient satisfaction, and better preventive care.

Emergency Care

The advanced automation and analytics of IoT allows more powerful emergency support services, which typically suffer from their limited resources and disconnect with the base facility. It provides a way to analyze an emergency in a more complete way from miles away. It also gives more providers access to the patient prior to their arrival. IoT gives providers critical information for delivering essential care on arrival. It also raises the level of care available to a patient received by emergency professionals. This reduces the associated losses, and improves emergency healthcare.

11. IoT – Building/Housing Applications

IoT applied to buildings and various structures allows us to automate routine residential and commercial tasks and needs in a way that dramatically improves living and working environments. This, as seen with manufacturing and energy applications, reduces costs, enhances safety, improves individual productivity, and enhances quality of life.

Environment and Conditioning

One of the greatest challenges in the engineering of buildings remains management of environment and conditions due to many factors at work. These factors include building materials, climate, building use, and more. Managing energy costs receives the most attention, but conditioning also impacts the durability and state of the structure.

IoT aids in improving structure design and managing existing structures through more accurate and complete data on buildings. It provides important engineering information such as how well a material performs as insulation in a particular design and environment.

Health and Safety

Buildings, even when constructed with care, can suffer from certain health and safety issues. These issues include poor performing materials, flaws that leave the building vulnerable to extreme weather, poor foundations, and more.

The Boss 220 smart plug allows the user to monitor, control, optimize, and automate all plug-in devices. Users employ their mobile device or desktop to view performance information and control devices from anywhere.

Current solutions lack the sophistication needed to detect minor issues before they become major issues, or emergencies. IoT offers a more reliable and complete solution by observing issues in a fine-grained way to control dangers and aid in preventing them; for example, it can measure changes in a system's state impacting fire safety rather than simply detecting smoke.

Productivity and Quality of Life

Beyond safety or energy concerns, most people desire certain comforts from housing or commercial spaces like specific lighting and temperature. IoT enhances these comforts by allowing faster and easier customizing.

Adjustments also apply to the area of productivity. They personalize spaces to create an optimized environment such as a smart office or kitchen prepared for a specific individual.

12. IoT – Transportation Applications

At every layer of transportation, IoT provides improved communication, control, and data distribution. These applications include personal vehicles, commercial vehicles, trains, UAVs, and other equipment. It extends throughout the entire system of all transportation elements such as traffic control, parking, fuel consumption, and more.

Rails and Mass Transit

Current systems deliver sophisticated integration and performance, however, they employ older technology and approaches to MRT. The improvements brought by IoT deliver more complete control and monitoring. This results in better management of overall performance, maintenance issues, maintenance, and improvements.

Mass transit options beyond standard MRT suffer from a lack of the integration necessary to transform them from an option to a dedicated service. IoT provides an inexpensive and advanced way to optimize performance and bring qualities of MRT to other transportation options like buses. This improves services and service delivery in the areas of scheduling, optimizing transport times, reliability, managing equipment issues, and responding to customer needs.

Road

The primary concerns of traffic are managing congestion, reducing accidents, and parking. IoT allows us to better observe and analyze the flow of traffic through devices at all traffic observation points. It aids in parking by making storage flow transparent when current methods offer little if any data.

This smart road sign receives data and modifications to better inform drivers and prevent congestion or accidents.

Accidents typically result from a number of factors, however, traffic management impacts their frequency. Construction sites, poor rerouting, and a lack of information about traffic status are all issues that lead to incidents. IoT provides solutions in the form of better information sharing with the public, and between various parties directly affecting road traffic.

Automobile

Many in the automotive industry envision a future for cars in which IoT technology makes cars “smart,” attractive options equal to MRT. IoT offers few significant improvements to personal vehicles. Most benefits come from better control over related infrastructure and the inherent flaws in automobile transport; however, IoT does improve personal vehicles as personal spaces. IoT brings the same improvements and customization to a vehicle as those in the home.

Commercial Transportation

Transportation benefits extend to business and manufacturing by optimizing the transport arm of organizations. It reduces and eliminates problems related to poor fleet management through better analytics and control such as monitoring idling, fuel consumption, travel conditions, and travel time between points. This results in product transportation operating more like an aligned service and less like a collection of contracted services.

13. IoT – Education Applications

IoT in the classroom combines the benefits of IoT in content delivery, business, and healthcare. It customizes and enhances education by allowing optimization of all content and forms of delivery. It enables educators to give focus to individuals and their method. It also reduces costs and labor of education through automation of common tasks outside of the actual education process.

Education Organizations

Education organizations typically suffer from limited funding, labor issues, and poor attention to actual education. They, unlike other organizations, commonly lack or avoid analytics due to their funding issues and the belief that analytics do not apply to their industry.

IoT not only provides valuable insight, but it also democratizes that information through low-cost, low-power small devices, which still offer high performance. This technology aids in managing costs, improving the quality of education, professional development, and facility management improvement through rich examinations of key areas:

- Student response, performance, and behavior
- Instructor response, performance, and behavior
- Facility monitoring and maintenance
- Data from other facilities

Data informs them about ineffective strategies and actions, whether educational efforts or facility qualities. Removing these roadblocks makes them more effective.

Educators

Information provided by IoT empowers educators to deliver improved education. They have a window into the success of their strategies, their students' perspective, and other aspects of their performance. IoT relieves them of administrative and management duties, so they can focus on their mission. It automates manual and clerical labor, and facilitates supervising through features like system flags or controls to ensure students remain engaged.

A school in Richmond, California, embeds RFID chips in ID cards to track the presence of students. Even if students are not present for check-in, the system will track and log their presence on campus.

IoT provides instructors with easy access to powerful educational tools. Educators can use IoT to perform as a one-on-one instructor providing specific instructional designs for each pupil; for example, using data to determine the most effective supplements for each student, and auto-generating content from lesson materials on-demand for any student.

The application of technology improves the professional development of educators because they truly see what works, and learn to devise better strategies, rather than simply repeating old or ineffective methods.

IoT also enhances the knowledge base used to devise education standards and practices. Education research suffers from accuracy issues and a general lack of data. IoT introduces large high quality, real-world datasets into the foundation of educational design. This comes from IoT's unique ability to collect enormous amounts of varied data anywhere.

Personalized Education

IoT facilitates the customization of education to give every student access to what they need. Each student can control their experience and participate in instructional design, and much of this happens passively. The student simply utilizes the system, and performance data primarily shapes their design. This combined with organizational and educator optimization delivers highly effective education while reducing costs.

14. IoT – Government Applications

IoT supports the development of *smart* nations and *smart* cities. This includes enhancement of infrastructure previously discussed (e.g., healthcare, energy, transportation, etc.), defense, and also the engineering and maintenance of communities.

City Planning and Management

Governing bodies and engineers can use IoT to analyze the often complex aspects of city planning and management. IoT simplifies examining various factors such as population growth, zoning, mapping, water supply, transportation patterns, food supply, social services, and land use. It gathers detailed data in these areas and produces more valuable and accurate information than current analytics given its ability to actually "live" with people in a city.

Smart trashcans in New York tell garbage collectors when they need to be emptied. They optimize trash service by ensuring drivers only make necessary stops, and drivers modify their route to reduce fuel consumption.

In the area of management, IoT supports cities through its implementation in major services and infrastructure such as transportation and healthcare. It also aids in other key areas like water control, waste management, and emergency management. Its real-time and detailed information facilitate more prompt decisions in contrast to the traditional process plagued by information lag, which can be critical in emergency management.

Standard state services are also improved by IoT, which can automate otherwise slow processes and trim unnecessary state expenses; for example, it can automate motor vehicle services for testing, permits, and licensing.

IoT also aids in urban improvement by skipping tests or poor research, and providing functional data for how the city can be optimized. This leads to faster and more meaningful changes.

Creating Jobs

IoT offers thorough economic analysis. It makes previous blind spots visible and supports better economic monitoring and modeling. It analyzes industry and the marketplace to spot opportunities for growth and barriers.

National Defense

National threats prove diverse and complicated. IoT augments armed forces systems and services, and offers the sophistication necessary to manage the landscape of national defense. It supports better protection of borders through inexpensive, high performance devices for rich control and observation.

IoT automates the protection tasks typically spread across several departments and countless individuals. It achieves this while improving accuracy and speed.

15. IoT – Law Enforcement Applications

IoT enhances law enforcement organizations and practice, and improves the justice system. The technology boosts transparency, distributes critical data, and removes human intervention where it proves unnecessary.

Policing

Law enforcement can be challenging. IoT acts as an instrument of law enforcement which reduces manual labor and subjective decisions through better data, information sharing, and advanced automation. IoT systems shave costs by reducing human labor in certain areas such as certain traffic violations.

IoT aids in creating better solutions to problems by using technology in the place of force; for example, light in-person investigations of suspicious activities can be replaced with remote observation, logged footage of violations, and electronic ticketing. It also reduces corruption by removing human control and opinion for some violations.

This dart planted in a truck gate prevents dangerous car chases. A patrol car launches the tracking dart which pierces the vehicle. Then the main system receives all data needed to locate the vehicle.

Court System

Current court systems utilize traditional technology and resources. They generally do not exploit modern analytics or automation outside of minor legal tasks. IoT brings superior analytics, better evidence, and optimized processes to court systems which accelerate processes, eliminate excessive procedures, manage corruption, reduce costs, and improve satisfaction.

In the criminal court system, this can result in a more effective and fair system. In routine court services, it introduces automation similar to that of common government office services; for example, IoT can automate forming an LLC.

IoT combined with new regulations can remove lawyers from many common legal tasks or reduce the need for their involvement. This reduces costs and accelerates many processes which often require months of traversing legal procedures and bureaucracy.

16. IoT – Consumer Applications

Consumers benefit personally and professionally from the optimization and data analysis of IoT. IoT technology behaves like a team of personal assistants, advisors, and security. It enhances the way we live, work, and play.

Home

IoT takes the place of a full staff:

- **Butler –** IoT waits for you to return home, and ensures your home remains fully prepared. It monitors your supplies, family, and the state of your home. It takes actions to resolve any issues that appear.
- **Chef –** An IoT kitchen prepares meals or simply aids you in preparing them.
- **Nanny –** IoT can somewhat act as a guardian by controlling access, providing supplies, and alerting the proper individuals in an emergency.
- **Gardner –** The same IoT systems of a farm easily work for home landscaping.
- **Repairman –** Smart systems perform key maintenance and repairs, and also request them.
- **Security Guard –** IoT watches over you 24/7. It can observe suspicious individuals miles away, and recognize the potential of minor equipment problems to become disasters well before they do.

This smart, connected stove from Whirlpool allows two different heat settings on the same surface, remote monitoring, and remote control.

Work

A smart office or other workspace combines customization of the work environment with smart tools. IoT learns about you, your job, and the way you work to deliver an optimized environment. This results in practical accommodations like adjusting the room temperature, but also more advanced benefits like modifying your schedule and the tools you use to increase your output and reduce your work time. IoT acts as a manager and consultant capable of seeing what you cannot.

Play

IoT learns as much about you personally as it does professionally. This enables the technology to support leisure:

- **Culture and Night Life –** IoT can analyze your real-world activities and response to guide you in finding more of the things and places you enjoy such as recommending restaurants and events based on your preferences and experiences.
- **Vacations –** Planning and saving for vacations proves difficult for some, and many utilize agencies, which can be replaced by IoT.
- **Products and Services –** IoT offers better analysis of the products you like and need than current analytics based on its deeper access. It integrates with key information like your finances to recommend great solutions.

17. IoT – ThingWorx

Thingworx is a platform for the rapid development and deployment of smart, connected devices. Its set of integrated IoT development tools support connectivity, analysis, production, and other aspects of IoT development.

It offers Vuforia for implementing augmented reality development, and Kepware for industrial connectivity. KEPServerEX provides a single point for data distribution, and facilitates interoperability when partnered with a ThingWorx agent.

Components

Thingworx offers several key tools for building applications. These tools include the Composer, the Mashup Builder, storage, a search engine, collaboration, and connectivity. The Composer provides a modeling environment for design testing. The Mashup Builder delivers easy dashboard building through common components (or widgets); for example, buttons, lists, wikis, gauges, and etc.

Thingworx uses a search engine known as SQUEAL, meaning Search, Query, and Analysis. Users employ SQUEAL in analyzing and filtering data, and searching records.

Interface

The ThingWorx platform uses certain terms you must familiarize yourself with. In the main screen's top menu, you search for **entities** or create them. "Entity" refers to something created in ThingWorx. You can also import/export files and perform various operations on them.

In the left menu, you find entity groups, which are used to produce models and visualize data; and manage storage, collaboration, security, and the system.

When you select the Modeling category in the menu, you begin the process by creating an entity. The entity can be any physical device or software element, and it produces an **event** on changes to its property values; for example, a sensor detects a temperature change. You can set events to trigger actions through a subscription which makes decisions based on device changes.

Data Shapes consist of one or more fields. They describe the data structure of custom events, infotables, streams, and datatables. Data shapes are considered entities.

Thing Templates and **Thing Shapes** allow developers to avoid repeating device property definitions in large IoT systems. Developers create Thing Templates to allow new devices to inherit properties. They use Thing Shapes to define Templates, properties, or execute services.

Note a Thing only inherits properties, services, events, and other qualities from a single template, however, Things and templates can inherit properties from multiple Thing Shapes.

Development

ThingWorx actually requires very little programming. Users connect devices, establish a data source, establish device behaviors, and build an interface without any coding. It also offers scalability appropriate for both hobbyist projects and industrial applications.

18. IoT – Cisco Virtualized Packet Core

Cisco Virtualized Packet Core (VPC) is a technology providing all core services for 4G, 3G, 2G, WiFi, and small cell networks. It delivers networking functionality as virtualized services to allow greater scalability and faster deployment of new services at a reduced cost. It distributes and manages packet core functions across all resources, whether virtual or physical. Its key features include packet core service consolidation, dynamic scaling, and system agility.

Its technology supports IoT by offering network function virtualization, SDN (software-defined networking), and rapid networked system deployment. This proves critical because its virtualization and SDN support low-power, high flow networking, and the simple deployment of a wide variety of small devices. It eliminates many of the finer details of IoT systems, and conflicts, through consolidating into a single system and single technology for connecting and integrating all elements.

Use Case: Smart Transportation

Rail transportation provides a viable example of the power of VPC. The problems VPC solves relate to safety, mobility, efficiency, and service improvement:

- Rail applications use their own purpose-built networks, and suffer from interoperability issues; for example, trackside personnel cannot always communicate with local police due to different technologies.

- Determining if passengers need extra time to board remains a mostly manual task.
- Data updates, like schedules, remain manual.
- Each piece of equipment, e.g., a surveillance camera, requires its own network and power source.

A smart MRT sign in New York

VPC improves service by introducing direct communication over a standard network, more and automated monitoring, automatic data updates through smart signs, and native IP networks for all devices along with PoE (Power over Ethernet) technology. This results in passengers who feel safer, and enjoy a better quality service.

19. IoT – Salesforce

The Salesforce IoT Cloud is a platform for storing and processing IoT data. It uses the Thunder engine for scalable, real-time event processing. Its collection of application development components, known as Lightning, powers its applications. It gathers data from devices, websites, applications, customers, and partners to trigger actions for real-time responses.

Salesforce, a CRM leader, decided to enter this space due to the need to remain competitive in the coming era. The IoT cloud adds to Salesforce by expanding its reach, and the depth of its analytics.

Salesforce combined with IoT delivers dramatically improved customer service with tighter integration and responses to real-time events; for example, adjustments in wind turbines could trigger automatic rebooking of delayed/canceled connecting flights before airline passengers land.

Electric Imp

The Electric Imp platform is Salesforce's recommended method for quickly connecting devices to the cloud. You develop applications through the Squirrel language; a high level, OO, lightweight scripting language. Applications consist of two modules: the device module, which runs on the device; and the agent module, which runs in the Electric Imp cloud. The platform ensures secure communication between the modules, and you send devices messages with a simple call:

```
agent.send("nameOfmessage", data);
```

Listen for messages on the agent with the following code:

```
device.on("nameOfmessage", function(data) {
  //Data operations
});
```

Beyond these basic tasks, coding for device interaction, monitoring, and response resembles standard web application development, and uses a simple, easy-to-learn syntax.

20. IoT – GE Predix

GE (General Electric) Predix is a software platform for data collection from industrial instruments. It provides a cloud-based PaaS (platform as a service), which enables industrial-grade analytics for operations optimization and performance management. It connects data, individuals, and equipment in a standard way.

Predix was designed to target factories, and give their ecosystems the same simple and productive function as operating systems that transformed mobile phones. It began as a tool for General Electric's internal IoT, specifically created to monitor products sold.

Ge Predix Partnered with Microsoft Azure

Microsoft's Azure is a cloud computing platform and supporting infrastructure. It provides PaaS and IaaS, and assorted tools for building systems. Predix, recently made available on Azure, exploits a host of extra features like AI, advanced data visualization, and natural language technology. Microsoft plans to eventually integrate Predix with its Azure IoT suite and Cortana Intelligence suite, and also their well-established business applications. Azure will also allow users to build applications using Predix data. Note AWS and Oracle also support Predix.

Developer Kits

GE offers inexpensive developer kits consisting of general components and an Intel Edison processor module. Developers have the options of a dual core board and a Raspberry Pi board. Developers need only provide an IP address, Ethernet connection, power supply, and light programming to set data collection.

The kit automatically establishes the necessary connection, registers with the central Predix system, and begins transmitting environmental data from sensors. Users subscribe to hardware/software output, and GE Digital owns and manages the hardware and software for the user.

This kit replaces the awkward and involved assemblies of simulations and testing environments. In other simulations, developers typically use a large set of software (one for each device), and specific configurations for each connection. They also program the monitoring of each device, which can sometimes take hours. The kit reduces much of the time spent performing these tasks from hours to only minutes.

The Predix developer kit

The kit also includes software components for designing an IoT application that partners with Predix services. GE plans to release other versions of the kit for different applications.

21. IoT – Eclipse IoT

Eclipse IoT is an ecosystem of entities (industry and academia) working together to create a foundation for IoT based exclusively on open source technologies. Their focus remains in the areas of producing open source implementations of IoT standard technology; creating open source frameworks and services for utilization in IoT solutions; and developing tools for IoT developers.

Smarthome Project

SmartHome is one of Eclipse IoT's major services. It aims to create a framework for building smart home solutions, and its focus remains heterogeneous environments, meaning assorted protocols and standards integration.

SmartHome provides uniform device and information access to facilitate interaction between devices. It consists of OSGi bundles capable of deployment in an OSGi runtime, with OSGi services defined as extension points.

OSGi bundles are Java class groups and other resources, which also include detailed manifest files. The manifest contains information on file contents, services needed to enhance class behavior, and the nature of the aggregate as a component. Review an example of a manifest below:

```
Bundle-Name: Hi Everyone                          // Bundle Name

Bundle-SymbolicName: xyz.xyz.hievery1             // Header specifying an identifier

Bundle-Description: A Hi Everyone bundle          // Functionality description

Bundle-ManifestVersion: 2                         // OSGi specification

Bundle-Version: 1.0.0                             // Version number of bundle
```

```
Bundle-Activator: xyz.xyz.Activator             // Class invoked on bundle activation
Export-Package: xyz.xyz.helloworld;version="1.0.0"   // Java packages available externally
Import-Package: org.osgi.framework;version="1.3.0"   // Java packages needed from
                                                      // external source
```

Eclipse SCADA

Eclipse SCADA, another major Eclipse IoT service, delivers a means of connecting various industrial instruments to a shared communication system. It also post-processes data and sends data visualizations to operators. It uses a SCADA system with a communication service, monitoring system, archive, and data visualization.

It aims to be a complete, state-of-the-art open source SCADA system for developing custom solutions. Its supported technologies and tools include shell applications, JDBC, Modbus TCP and RTU, Simatic S7 PLC, OPC, and SNMP.

22. IoT – Contiki

Contiki is an operating system for IoT that specifically targets small IoT devices with limited memory, power, bandwidth, and processing power. It uses a minimalist design while still packing the common tools of modern operating systems. It provides functionality for management of programs, processes, resources, memory, and communication.

Contiki

The Open Source OS for the Internet of Things

It owes its popularity to being very lightweight (by modern standards), mature, and flexible. Many academics, organization researchers, and professionals consider it a go-to OS. Contiki only requires a few kilobytes to run, and within a space of under 30KB, it fits its entire operating system: a web browser, web server, calculator, shell, telnet client and daemon, email client, vnc viewer, and ftp. It borrows from operating systems and development strategies from decades ago, which easily exploited equally small space.

Contiki Communication

Contiki supports standard protocols and recent enabling protocols for IoT:

- **uIP (for IPv4) –** This TCP/IP implementation supports 8-bit and 16-bit microcontrollers.
- **uIPv6 (for IPv6)** – This is a fully compliant IPv6 extension to uIP.
- **Rime –** This alternative stack provides a solution when IPv4 or IPv6 prove prohibitive. It offers a set of primitives for low-power systems.
- **6LoWPAN –** This stands for IPv6 over low-power wireless personal area networks. It provides compression technology to support the low data rate wireless needed by devices with limited resources.
- **RPL –** This distance vector IPv6 protocol for LLNs (low-power and lossy networks) allows the best possible path to be found in a complex network of devices with varied capability.
- **CoAP –** This protocol supports communication for simple devices, typically devices requiring heavy remote supervision.

Dynamic Module Loading

Dynamic module loading and linking at run-time supports environments in which application behavior changes after deployment. Contiki's module loader loads, relocates, and links ELF files.

The Cooja Network Simulator

Cooja, the Contiki network simulator, spawns an actual compiled and working Contiki system controlled by Cooja.

Using Cooja proves simple. Simply create a new mote type by selecting the **Motes** menu and **Add Motes > Create New Mote Type**. In the dialog that appears, you choose a name for the mote, select its firmware, and test its compilation.

After creation, add motes by clicking **Create**. A new mote type will appear to which you can attach nodes. The final step requires saving your simulation file for future use.

23. IoT – Security

Every connected device creates opportunities for attackers. These vulnerabilities are broad, even for a single small device. The risks posed include data transfer, device access, malfunctioning devices, and always-on/always-connected devices.

The main challenges in security remain the security limitations associated with producing low-cost devices, and the growing number of devices which creates more opportunities for attacks.

Security Spectrum

The definition of a secured device spans from the most simple measures to sophisticated designs. Security should be thought of as a spectrum of vulnerability which changes over time as threats evolve.

Security must be assessed based on user needs and implementation. Users must recognize the impact of security measures because poorly designed security creates more problems than it solves.

Example: A German report revealed hackers compromised the security system of a steel mill. They disrupted the control systems, which prevented a blast furnace from being shut down properly, resulting in massive damage. Therefore, users must understand the impact of an attack before deciding on appropriate protection.

Challenges

Beyond costs and the ubiquity of devices, other security issues plague IoT:

- **Unpredictable Behavior –** The sheer volume of deployed devices and their long list of enabling technologies means their behavior in the field can be unpredictable. A specific system may be well designed and within administration control, but there are no guarantees about how it will interact with others.
- **Device Similarity –** IoT devices are fairly uniform. They utilize the same connection technology and components. If one system or device suffers from a vulnerability, many more have the same issue.
- **Problematic Deployment –** One of the main goals of IoT remains to place advanced networks and analytics where they previously could not go. Unfortunately, this creates the problem of physically securing the devices in these strange or easily accessed places.
- **Long Device Life and Expired Support –** One of the benefits of IoT devices is longevity, however, that long life also means they may outlive their device support. Compare this to traditional systems which typically have support and upgrades long after many have stopped using them. Orphaned devices and abandonware lack the same security hardening of other systems due to the evolution of technology over time.
- **No Upgrade Support –** Many IoT devices, like many mobile and small devices, are not designed to allow upgrades or any modifications. Others offer inconvenient upgrades, which many owners ignore, or fail to notice.
- **Poor or No Transparency –** Many IoT devices fail to provide transparency with regard to their functionality. Users cannot observe or access their processes, and are left to assume how devices behave. They have no control over unwanted functions or data collection; furthermore, when a manufacturer updates the device, it may bring more unwanted functions.
- **No Alerts –** Another goal of IoT remains to provide its incredible functionality without being obtrusive. This introduces the problem of user awareness. Users do not monitor the devices or know when something goes wrong. Security breaches can persist over long periods without detection.

24. IoT – Identity Protection

IoT devices collect data about their environment, which includes people. These benefits introduce heavy risk. The data itself does not present the danger, however, its depth does. The highly detailed data collection paints a very clear picture of an individual, giving criminals all the information they need to take advantage of someone.

People may also not be aware of the level of privacy; for example, entertainment devices may gather A/V data, or "watch" a consumer, and share intimate information. The demand and price for this data exacerbates the issue considering the number and diversity of parties interested in sensitive data.

Problems specific to IoT technology lead to many of its privacy issues, which primarily stem from the user's inability to establish and control privacy:

Consent

The traditional model for "notice and consent" within connected systems generally enforces existing privacy protections. It allows users to interact with privacy mechanisms, and set preferences typically through accepting an agreement or limiting actions. Many IoT devices have no such accommodations. Users not only have no control, but they are also not afforded any transparency regarding device activities.

The Right to be Left Alone

Users have normal expectations for privacy in certain situations. This comes from the commonly accepted idea of public and private spaces; for example, individuals are not surprised by surveillance cameras in commercial spaces, however, they do not expect them in their personal vehicle. IoT devices challenge these norms people recognize as the "right to be left alone." Even in public spaces, IoT creeps beyond the limits of expected privacy due to its power.

Indistinguishable Data

IoT deploys in a wide variety of ways. Much of IoT implementation remains group targeted rather than personal. Even if users give IoT devices consent for each action, not every system can reasonably process every set of preferences; for example, small devices in a complex assembly cannot honor the requests of tens of thousands of users they encounter for mere seconds.

Granularity

Modern big data poses a substantial threat to privacy, but IoT compounds the issue with its scale and intimacy. It goes not only where passive systems cannot, but it collects data everywhere. This supports creation of highly detailed profiles which facilitate discrimination and expose individuals to physical, financial, and reputation harm.

Comfort

The growth of IoT normalizes it. Users become comfortable with what they perceive as safe technology. IoT also lacks the transparency that warns users in traditional connected systems; consequently, many act without any consideration for the potential consequences.

25. IoT – Liability

The security flaws of IoT and its ability to perform certain tasks open the door to any associated liability. The three main areas of concern are device malfunction, attacks, and data theft. These issues can result in a wide variety of damages.

Device Malfunction

IoT introduces a deeper level of automation which can have control over critical systems, and systems impacting life and property. When these systems fail or malfunction, they can cause substantial damage; for example, if an IoT furnace control system experiences a glitch, it may fail in an unoccupied home and cause frozen pipes and water damage. This forces organizations to create measures against it.

This smart thermostat allows attackers to gain remote access, and breach the rest of the network.

Cyber Attacks

IoT devices expose an entire network and anything directly impacted to the risk of attacks. Though those connections deliver powerful integration and productivity, they also create the perfect opportunity for mayhem like a hacked stove or fire safety sprinkler system. The best measures against this address the most vulnerable points, and provide custom protections such as monitoring and access privileges.

Some of the most effective measures against attacks prove simple:

- **Built-in Security –** Individuals and organizations should seek hardened devices, meaning those with security integrated in the hardware and firmware.
- **Encryption –** This must be implemented by the manufacturer and through user systems.
- **Risk Analysis –** Organizations and individuals must analyze possible threats in designing their systems or choosing them.
- **Authorization –** Devices, whenever possible, must be subject to privilege policies and access methods.

Bitdefender BOX secures all connected devices in the home.

Data Theft

Data, IoT's strength and weakness, proves irresistible to many. These individuals have a number of reasons for their interest: the value of personal data to marketing/advertising, identity theft, framing individuals for crimes, stalking, and a bizarre sense of satisfaction. Measures used to fight attacks are also effective in managing this threat.

26. IoT – Useful Resources

The following resources offer more in-depth information on IoT development and administration. You can refer them to increase your knowledge on IoT further.

Useful IoT Websites

- Internet of Things Council – This European think tank offers the best and latest IoT information. They analyze every aspect of IoT from forecasting to discussing prototype development, and the social implications of IoT.
- LinkedIn Pulse Content – LinkedIn, described as the world's largest professional network, allows over 100 million professionals to network. It opened its publishing platform, Pulse, to the public in 2014, resulting in a wealth of valuable professional and industry information. This information includes rare insight, training media, and more related to IoT systems and technologies.
- Lynda.com IoT Videos – This online learning organization offers thousands of videos on various topics (including IoT) supplied by professionals, organizations, and individuals.
- YouTube IoT Videos – Individuals just like you produce thousands of IoT videos on YT to address particular topics not covered elsewhere, or covered poorly elsewhere. These videos use different languages, styles, and offer unique voices.

Useful IoT Literature

Building Internet of Things with the Arduino (Volume 1) by *Charalampos Doukas*	**Making Things Talk, 2nd Edition** by *Tom Igoe*	**Building Wireless Sensor Networks: with ZigBee, Xbee, Arduino, and Processing** by *Robert Faludi*
Building the Web of Things with Examples in Node.js and Raspberry Pi by *Dominique D. Guinard and Vlad M. Trifa*	**Internet of Things (A Hands-on-Approach)** by *Arshdeep Bahga and Vijay Madisetti*	**The Internet of Things in the Cloud: A Middleware Perspective** by *Honbo Zhou*

If you would like your site or literature listed on this page, please contact us at **contact@tutorialspoint.com**

Interrupts in 8051 Microcontroller

The most powerful and important features are interrupts in 8051 microcontroller. In most of the real-time processes, to handle certain conditions properly, the actual task must be halt for some time – it takes required action – and then must return to the main task. For executing such type of programs, interrupts are necessary. It entirely differs from the polling method wherein the processor must check sequentially each device and ask whether the service is required or not while consuming more processor time.

Interrupts in 8051 microcontroller are more desirable to reduce the regular status checking of the interfaced devices or inbuilt devices. Interrupt is an event that temporarily suspends the main program, passes the control to a special code section, executes the event-related function and resumes the main program flow where it had left off.

Interrupts are of different types like software and hardware, maskable and non-maskable, fixed and vector interrupts, and so on. Interrupt Service Routine (ISR) comes into the picture when interrupt occurs, and then tells the processor to take appropriate action for the interrupt, and after ISR execution, the controller jumps into the main program.

Types of Interrupts in 8051 Microcontroller

The 8051 microcontroller can recognize five different events that cause the main program to interrupt from the normal execution. These five sources of interrupts in 8051are:

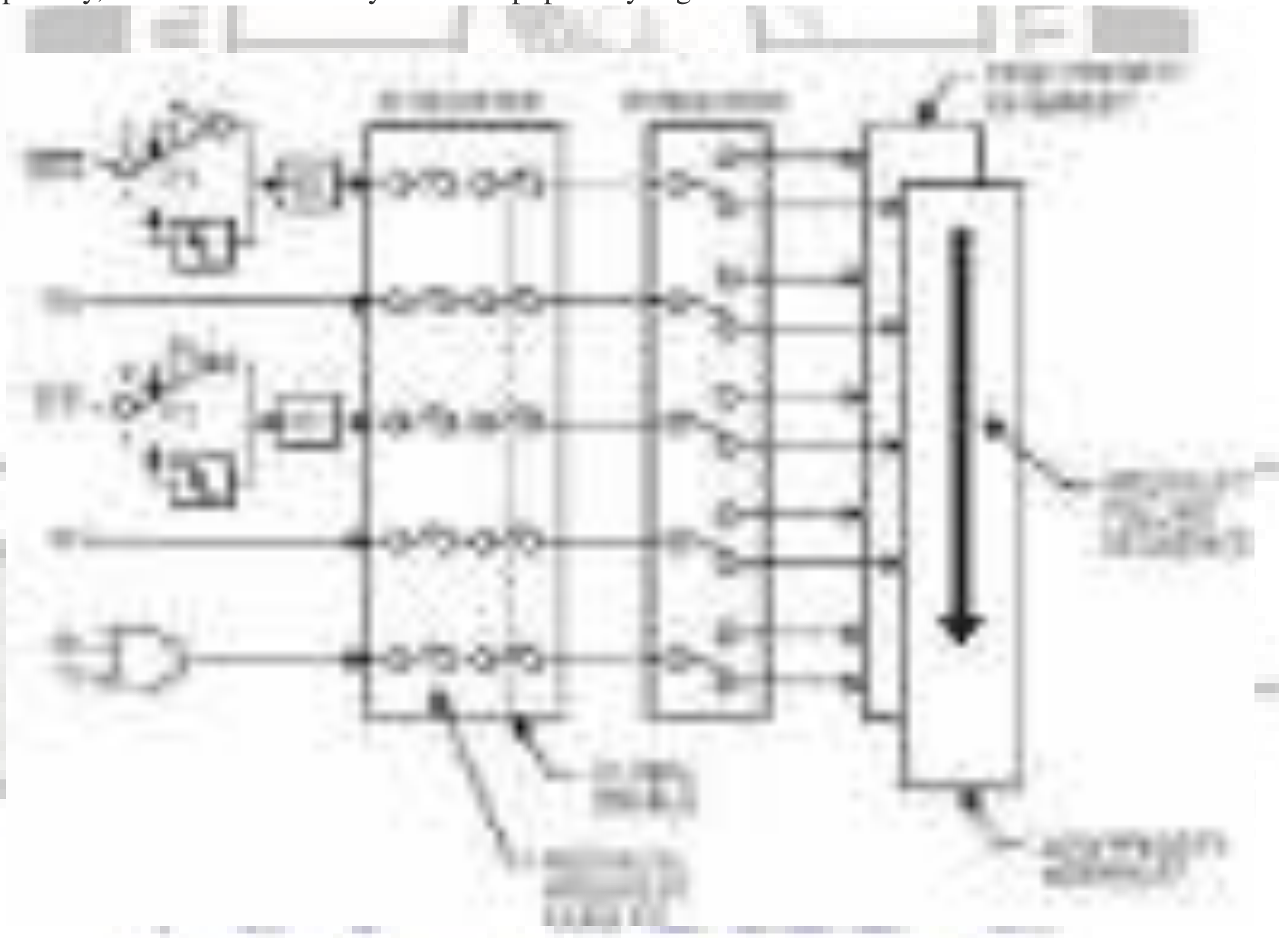
1. Timer 0 overflow interrupt- TF0
2. Timer 1 overflow interrupt- TF1
3. External hardware interrupt- INT0
4. External hardware interrupt- INT1
5. Serial communication interrupt- RI/TI

The Timer and Serial interrupts are internally generated by the microcontroller, whereas the external interrupts are generated by additional interfacing devices or switches that are externally connected to the microcontroller. These external interrupts can be edge triggered or level triggered. When an interrupt occurs, the microcontroller executes the interrupt service routine so that memory location corresponds to the interrupt that enables it. The Interrupt corresponding to the memory location is given in the interrupt vector table below.

[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

nterrupt Structure of 8051 Micro controller

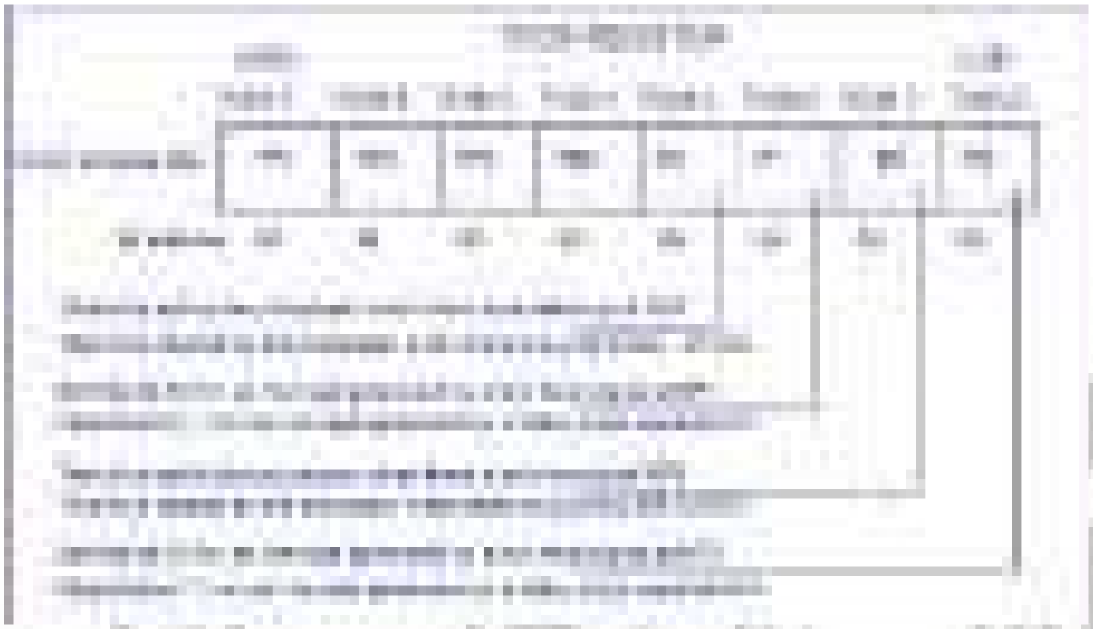
Upon 'RESET' all the interrupts get disabled, and therefore, all these interrupts must be enabled by a software. In all these five interrupts, if anyone or all are activated, this sets the corresponding interrupt flags as shown in the figure. All these interrupts can be set or cleared by bit in some special function register that is Interrupt Enabled (IE), and this in turn depends on the priority, which is executed by IP interrupt priority register.



Interrupt Enable (IE) Register: This register is responsible for enabling and disabling the interrupt. It is a bit addressable register in which EA must be set to one for enabling interrupts. The corresponding bit in this register enables particular interrupt like timer, external and serial inputs. In the below IE register, bit corresponding to 1 activates the interrupt and 0 disables the interrupt.

Interrupt Priority Register (IP): It is also possible to change the priority levels of the interrupts by setting or clearing the corresponding bit in the Interrupt priority (IP) register as shown in the figure. This allows the low priority interrupt to interrupt the high-priority interrupt, but prohibits the interruption by another low-priority interrupt. Similarly, the high-priority interrupt cannot be interrupted. If these interrupt priorities are not programmed, the microcontroller executes in predefined manner and its order is INT0, TF0, INT1, TF1, and SI.

TCON Register: In addition to the above two registers, the TCON register specifies the type of external interrupt to the 8051 microcontroller, as shown in the figure. The two external interrupts, whether edge or level triggered, specify by this register by a set, or cleared by appropriate bits in it. And, it is also a bit addressable register.


[illegible]

3.4 I/O PORTS AND DATA TRANSFER CONCEPTS

Parallel I /O Ports :

The 8051 microcontroller has four parallel I/O ports , each of 8-bits .So, it provides the user 32 I/O lines for connecting the microcontroller to the peripherals. The four ports are P0 (Port 0), P1(Port1) ,P2(Port 2) and P3 (Port3). Upon reset all the ports are output ports. In order to make them input, all the ports must be set i.e a high bit must be sent to all the port pins. This is normally done by the instruction "SETB".

```
Ex:   MOV A,#0FFH      ;  A =  FF
      MOV P0,A         ; make P0 an input port
```

PORT 0:

Port 0 is an 8-bit I/O port with dual purpose. If external memory is used, these port pins are used for the lower address byte address/data (AD0-AD7), otherwise all bits of the port are either input or output.. Unlike other ports, Port 0 is not provided with pull-up resistors internally ,so for PORT0 pull-up resistors of nearly 10k are to be connected externally as shown.

Dual role of port 0:

Port 0 can also be used as address/data bus(AD0-AD7), allowing it to be used for both address and data. When connecting the 8051 to an external memory, port 0 provides both address and data. The 8051 multiplexes address and data through port 0 to save the pins. ALE indicates whether P0 has address or data. When ALE = 0, it provides data D0-D7, and when ALE =1 it provides address and data with the help of a 74LS373 latch.

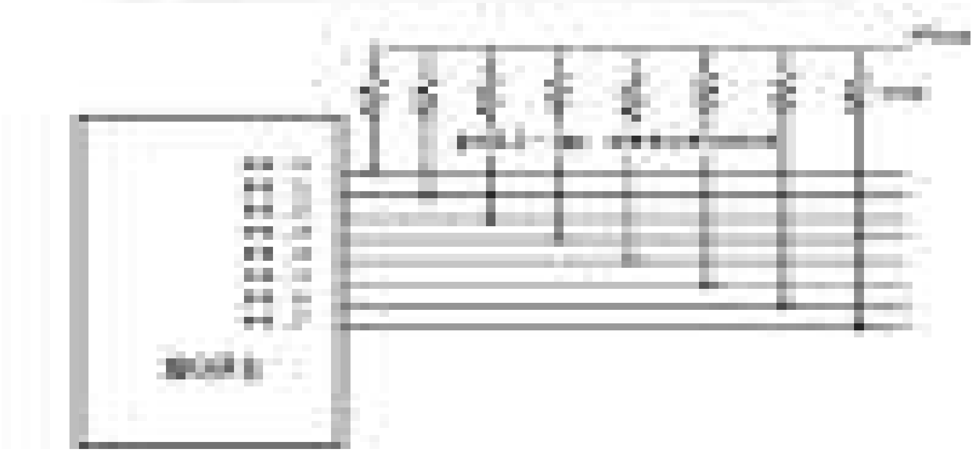


Figure 3.4.1 Dual role of port 0

[Source: "Microprocessor Architecture Programming and Application" by R.S. Gaonkar, page-]

Port 1:

Port 1 occupies a total of 8 pins (pins 1 through 8). It has no dual application and acts only as input or output port. In contrast to port 0, this port does not need any pull-up resistors since pull-up resistors connected internally. Upon reset, Port 1 is configured as an output port. To configure it as an input port, port bits must be set i.e a high bit must be sent to all the port pins. This is normally done by the instruction “SETB”.

```
Ex:     MOV A, #0FFH; A=FF HEX
        MOV P1, A; make P1 an input port by writing 1’s to all of its pins
```

Port 2:

Port 2 is also an eight-bit parallel port. (pins 21- 28). It can be used as input or output port. As this port is provided with internal pull-up resistors it does not need any external pull-up resistors. Upon reset, Port 2 is configured as an output port. If the port is to be used as input port, all the port bits must be made high by sending FF to the port. For

```
Ex:     MOV A, #0FFH        ; A=FF hex
        MOV P2, A           ; make P2 an input port by writing all 1’s to it
```

Dual role of port 2:

Port2 lines are also associated with the higher order address lines A8-A15. In systems based on the 8751, 8951, and DS5000, Port2 is used as simple I/O port. But, in 8031-based systems, port 2 is used along with P0 to provide the 16-bit address for the external memory. Since an 8031 is capable of accessing 64K bytes of external memory, it needs a path for the 16 bits of the address. While P0 provides the lower 8 bits via A0-A7, it is the job of P2 to provide bits A8-A15 of the address. In other words, when 8031 is connected to external memory, Port 2 is used for the upper 8 bits of the 16-bit address, and it cannot be used for I/O operations.

PORT 3:

Port3 is also an 8-bit parallel port with dual function. (pins 10 to 17). The port pins can be used for I/O operations as well as for control operations. The details of these additional operations are given below in the table. Port 3 also do not need any external pull-up resistors as they are provided internally similar to the case of Port2 & Port 1. Upon reset port 3 is configured as an output port. If the port is to be used as input port, all the port bits must be made high by sending FF to the port.

```
Ex:        MOV A, #0FFH      ; A= FF hex
           MOV P3, A         ; make P3 an input port by writing all 1's to it
```

Alternate Functions of Port 3 :

P3.0 and P3.1 are used for the RxD (Receive Data) and TxD (Transmit Data) serial communications signals. Bits P3.2 and P3.3 are meant for external interrupts. Bits P3.4 and P3.5 are used for Timers 0 and 1 and P3.6 and P3.7 are used to provide the write and read signals of external memories connected in 8031 based systems

S.No	Port 3 bit	Pin No	Function
1	P3.0	10	RxD
2	P3.1	11	TxD
3	P3.2	12	$\overline{INT0}$
4	P3.3	13	$\overline{INT1}$
5	P3.4	14	T0
6	P3.5	15	T1
7	P3.6	16	$\overline{WR}$
8	P3.7	17	$\overline{RD}$

Serial communication

Serial communication uses only one or two data lines to transfer data and is generally used for long distance communication. In serial communication the data is sent as one bit at a time in a timed sequence on a single wire. Serial Communication takes place in two methods, Asynchronous data Transfer and Synchronous Data Transfer.

Asynchronous data transfer:

It allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, special bits will be added to each word in order to synchronize the sending and receiving of the data. When a word is given to the UART for Asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter.

After the Start Bit, the individual bits of the word of data are sent. Here each bit in the word is transmitted for exactly the same amount of time as all of the other bits. When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates. The Parity bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter. If the Stop Bit does not appear when it is supposed to, the UART considers the entire word to be corrupted and will report a Framing Error.

Baud rate is a measurement of transmission speed in asynchronous communication, it represents the number of bits/sec that are actually being sent over the serial link. The Baud count includes the overhead bits Start, Stop and Parity that are generated by the sending UART and removed by the receiving UART.

Synchronous data transfer:

In this method the receiver knows when to “read” the next bit coming from the sender. This is achieved by sharing a clock between sender and receiver. In most forms of serial Synchronous communication, if there is no data available at a given time to transmit, a fill character will be sent instead so that data is always being transmitted. Synchronous communication is usually more efficient because only data bits are transmitted between sender and receiver, however it will be costlier because extra wiring and control circuits are required to share a clock signal between the sender and receiver. Devices that use serial cables for their communication are split into two categories.

1. DTE (Data Terminal Equipment). Examples of DTE are computers, printers & terminals.
2. DCE (Data Communication Equipment). Example of DCE is modems.

Parallel Data Transfer:

Parallel communication uses multiple wires (bus) running parallel to each other, and can transmit data on all the wires simultaneously. i.e all the bits of the byte are transmitted at a time. So, speed of the parallel data transfer is extremely high compared to serial data transfer. An 8-bit parallel data transfer is 8-times faster than serial data transfer. Hence with in the computer all data transfer is mainly based on Parallel data transfer. But only limitation is due to the high cost, this method is limited to only short distance communications.

S.No	Serial Communication	Parallel Communication
1	Data is transmitted bit after the bit in a single line	Data is transmitted simultaneously through group of lines(Bus)
2	Data congestion takes place	No, Data congestion
3	Low speed transmission	High speed transmission
4	Implementation of serial links is not an easy task.	Parallel data links are easily implemented in hardware
5.	In terms of transmission channel costs such as data bus cable length, data bus buffers, interface connectors, it is less expensive	It is more expensive
6	No , crosstalk problem	Crosstalk creates interference between the parallel lines.
7	No effect of inter symbol interference and noise	Parallel ports suffer extremely from inter-symbol interference (ISI) and noise, and therefore the data can be corrupted over long distances.
8	The bandwidth of serial wires is much higher.	The bandwidth of parallel wires is much lower.
9	Serial interface is more flexible to upgrade , without changing the hardware	Parallel data transfer mechanism rely on hardware resources and hence not flexible to upgrade.
10	Serial communication work effectively even at high frequencies.	Parallel buses are hard to run at high frequencies.

4.1 ARCHITECTURE OF 8051

MICROCONTROLLER EVOLUTION

First, microcontrollers were developed in the mid-1970s. These were basically calculator- based processors with small ROM program memories, very limited RAM data memories and a handful of input/output ports.

As silicon technology developed, more powerful, 8-bit microcontrollers were produced. In addition to their improved instruction sets, these microcontrollers included on-chip counter/timers, interrupt facilities, and improved I/O handling. On-chip memory capacity was still small and was not adequate for many applications. One of the most significant developments at this time was the availability of on-chip ultraviolet erasable EPROM memory. This simplified the product development time considerably and for the first time, also allowed the use of microcontrollers in low-volume applications.

The 8051 family was introduced in the early 1980s by Intel. Since its introduction, the 8051 has been one of the most popular microcontrollers and has been second- sourced by many manufacturers. The 8051 currently has many different versions and some types include on-chip analogue-to-digital converters, a considerably large size of program and data memories.

INTRODUCTION TO 8051:

The **Intel MCS-51** (commonly referred to as **8051**) is a Harvard architecture, single chip microcontroller (µC) series which was developed by Intel in 1980 for use in embedded systems. The 8051 architecture provides many functions (CPU, RAM, ROM, I/O, interrupt logic, timer,etc.) in a single package

Features of 8051:

- 8-bit ALU, Accumulator, 8-bit Registers and 8-bit data bus; hence it is an 8-bit microcontroller
- 16-bit program counter
- 8-bit Processor Status Word(PSW)
- 8-bit Stack Pointer

- Internal RAM of128bytes
- On chip ROM is4KB
- Special Function Registers (SFRs) of 128bytes
- 32 I/O pins arranged as four 8-bit ports (P0 -P3)
- Two 16-bit timer/counters : T0 andT1
- Two external and three internal vectored interrupts
- Full duplex UART (serialport)

BLOCK DIAGRAM

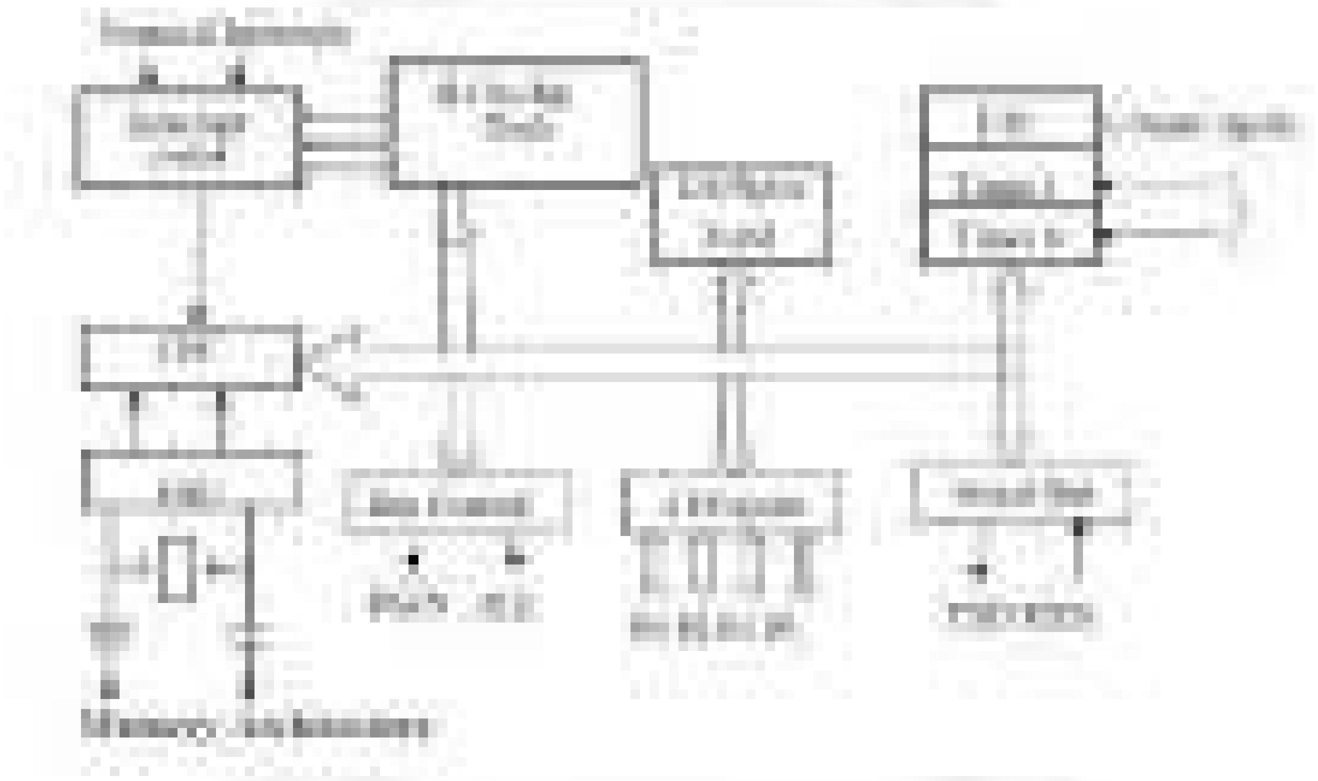


Figure 4.1.1 8051 Microcontroller Block Diagram

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay , pg.no.29]

128 BYTES OF INTERNAL RAM STRUCTURE (LOWER ADDRESS SPACE)

30H-7FH	**Scratch Pad**	**80 Bytes**
20H -2FH	**Bit Addressable RAM**	**16 Bytes**
1FH	R7	
	REGISTER BANK 3	
18H	R0	
17H	R7	
	REGISTER BANK 2	**32 Bytes**
10H	R0	
0FH	R7	
	REGISTER BANK 1	
08H	R0	
07H	R7	
	REGISTER BANK 0	
00H	R0	

Figure 4.1.2 RAM Allocation in 8051

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay]

The lower 32 bytes are divided into 4 separate banks. Each register bank has 8 registers of one byte each. A register bank is selected depending upon two bank select bits in the PSW register as shown in Figure 4.1.2.

- Next 16 bytes are bit addressable. In total, 128bits (16X8) are available in addressable area. Each bit can be accessed and modified by suitable instructions. The bit addresses are from 00H (LSB of the first byte in 20H) to 2FH (MSB of the last byte in 2FH).
- Remaining 80bytes of RAM (30H TO 7FH) are available for general purpose.

INTERNAL ARCHITECTURE OF 8051 MICROCONTROLLER

The Internal architecture is shown in Figure 4.1.4 and the various Registers and units are described below.

Accumulator (Acc):

•Operand register

• Implicit or specified in the instruction

•Has an address in on chip SFR bank

B Register:

Used to store one of the operands for multiplication and division, otherwise, scratch pad considered as a SFR.

Stack Pointer (SP):

8 bit wide register. Incremented before data is stored on to the stack using PUSH or CALL instructions. Stack defined anywhere on the 128 byte RAM.

Data Pointer (DPTR)*:*

*1*6 bit register contains DPH and DPL Pointer to external RAM address. DPH and DPL allotted separate addresses in SFR bank

Port 0 To 3 Latches & Drivers:

Each I/O port allotted a latch and a driver Latches allotted address in SFR. User can communicate via these ports P0, P1, P2, and P3.

Serial Data Buffer:

Internally had TWO independent registers, TRANSMIT buffer (parallel in serial out – PISO) and RECEIVE buffer (serial in parallel out –SIPO) identified by SBUF and allotted an address in SFR.

Program Status Word (PSW):

Set of flags contains status information as detailed below in the Figure 4.1.3.

Figure 4.1.3 Bits of PSW Register

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay , pg.no.52]

Timer Registers: for Timer0 (16 bit register – TL0 & TH0) and for Timer1 (16 bit register – TL1 & TH1) four addresses allotted in SFR

Control Registers: Control registers are IP, IE, TMOD, TCON, SCON, and PCON. These registers contain control and status information for interrupts, timers/counters and serial port. Allotted separate address in SFR.

Timing and Control Unit: This unit derives necessary timing and control signals for internal circuit and external system bus.

Oscillator: generates basic timing clock signal using crystal oscillator.

Instruction Register: Decodes the opcode and gives information to timing and control unit.

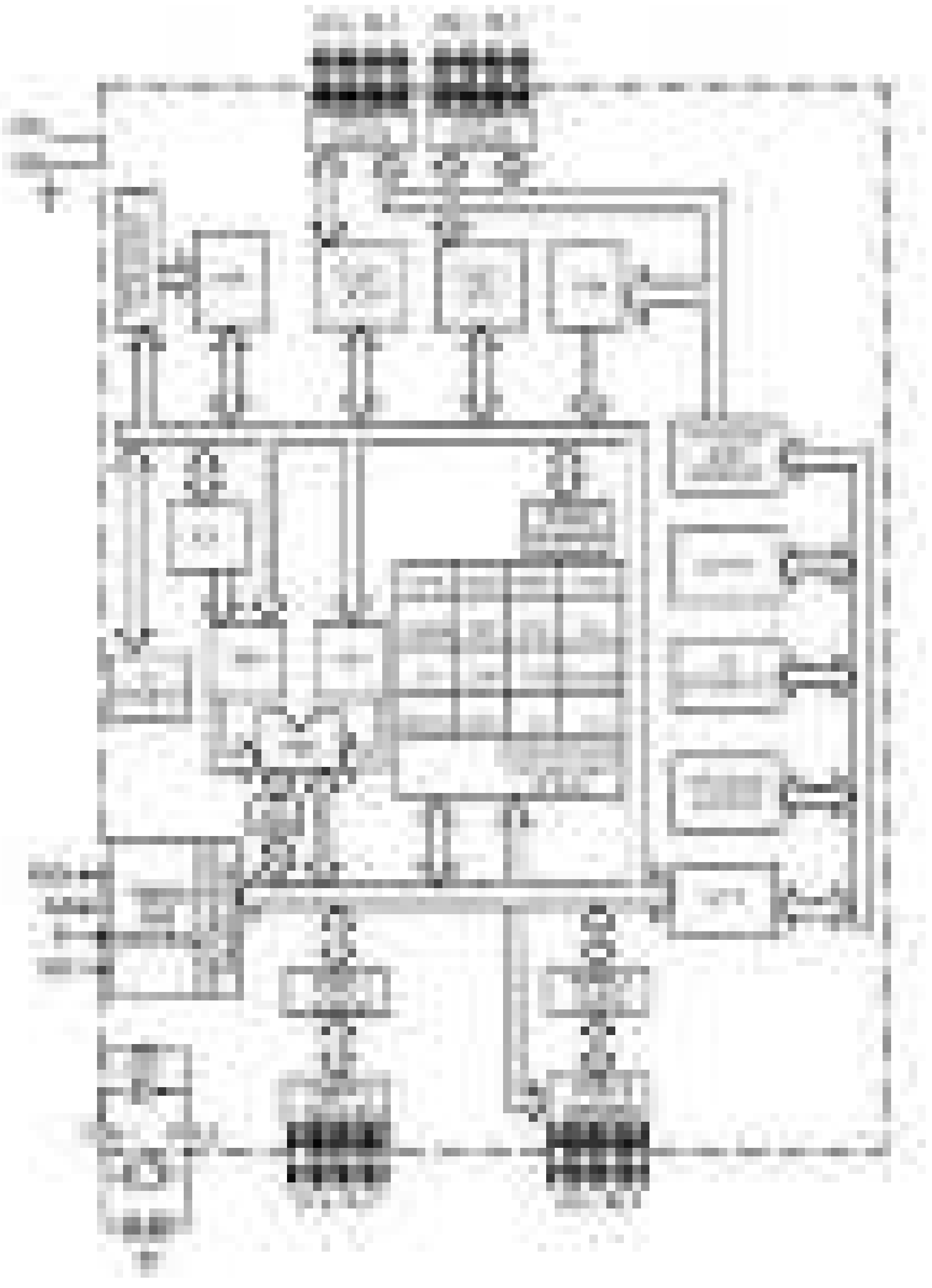


Figure 4.1.4 8051 Architecture Block diagram

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay , pg.no.29]

EPROM & program address Register: provide on chip EPROM and mechanism to address it. All versions don't have EPROM.

Ram & Ram Address Register: provide internal 128 bytes RAM and a mechanism to address internally

ALU: Performs 8 bit arithmetic and logical operations over the operands held by TEMP1 and TEMP 2.User cannot access temporary registers.

SFR Register Bank: set of special function registers address range: 80 H to FF H. Interrupt, serial port and timer units control and perform specific functions under the control of timing and control unit.

8051 PIN CONFIGURATION

The pin diagram of 8051 microcontroller is shown in Figure 4.1.5 and the pin details are described below.

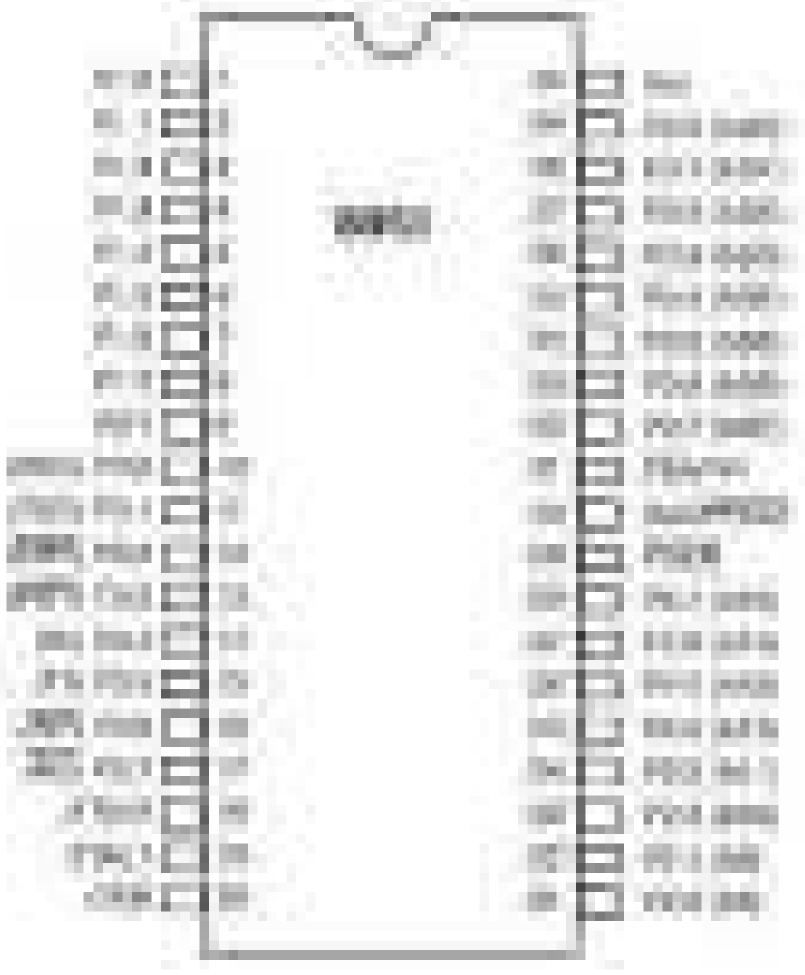


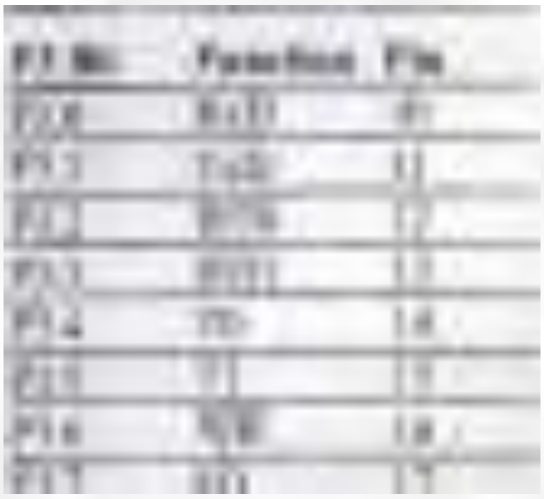
Figure 4.1.5 8051 Pin Configuration

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay , pg.no.94]

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

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

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



P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{INT0}$	12
P3.3	$\overline{INT1}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{WR}$	16
P3.7	$\overline{RD}$	17

Figure 4.1.6 Port 3 Alternate Functions

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay , pg.no.97]

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

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

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

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

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

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

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

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

SESHASAYEE INSTITUTE OF TECHNOLOGY
ARIYAMANGALAM , TRICHY – 620 010

ARCHITECTURE OF 8051 & THEIR PIN DETAILS

UNIT I

WELCOME

ARCHITECTURE OF 8051 & THEIR PIN DETAILS

U1.1 : Introduction to microprocessor & microcontroller : Architecture of 8085 -Functions of each block. Comparison of Microprocessor & Microcontroller - Features of microcontroller -Advantages of microcontroller -Applications Of microcontroller -Manufactures of microcontroller.

U1.2 : Architecture of 8051 : Block diagram of Microcontroller – Functions of each block. Pin details of 8051 -Oscillator and Clock -Clock Cycle -State - Machine Cycle -Instruction cycle –Reset - Power on Reset - Special function registers :Program Counter -PSW register -Stack - I/O Ports .

U1.3 : **Memory Organisation & I/O port configuration:** ROM RAM - Memory Organization of 8051,Interfacing external memory to 8051

Microcontroller vs. Microprocessors

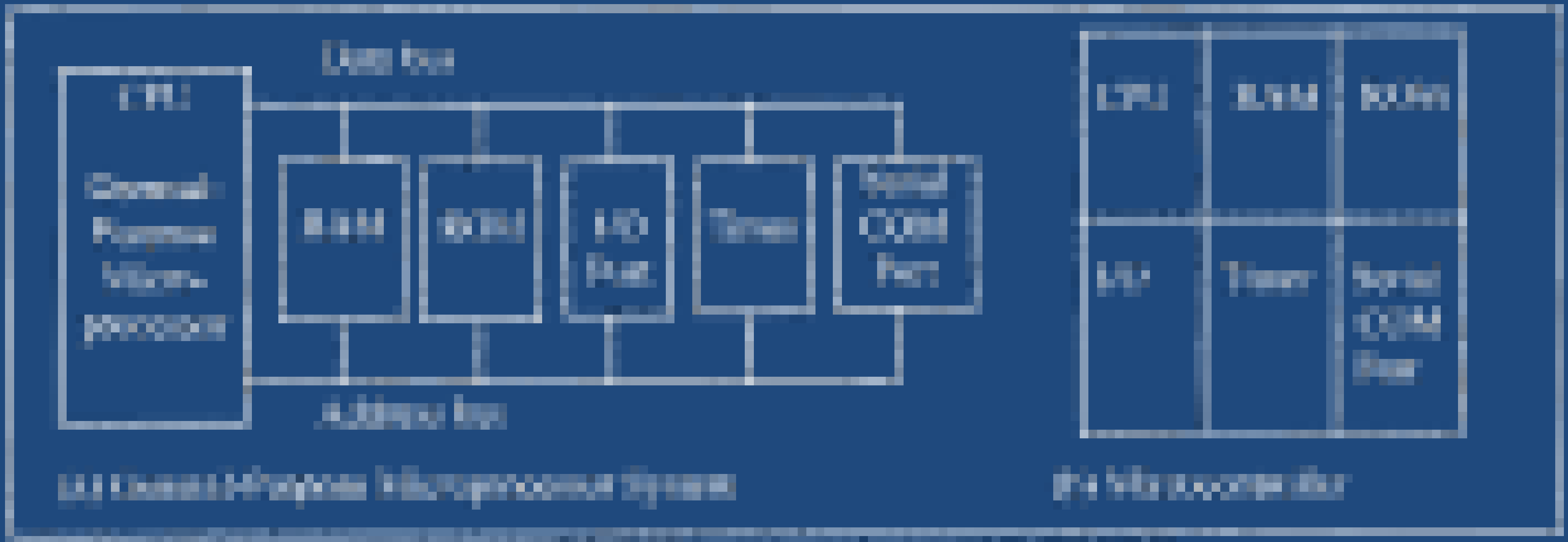
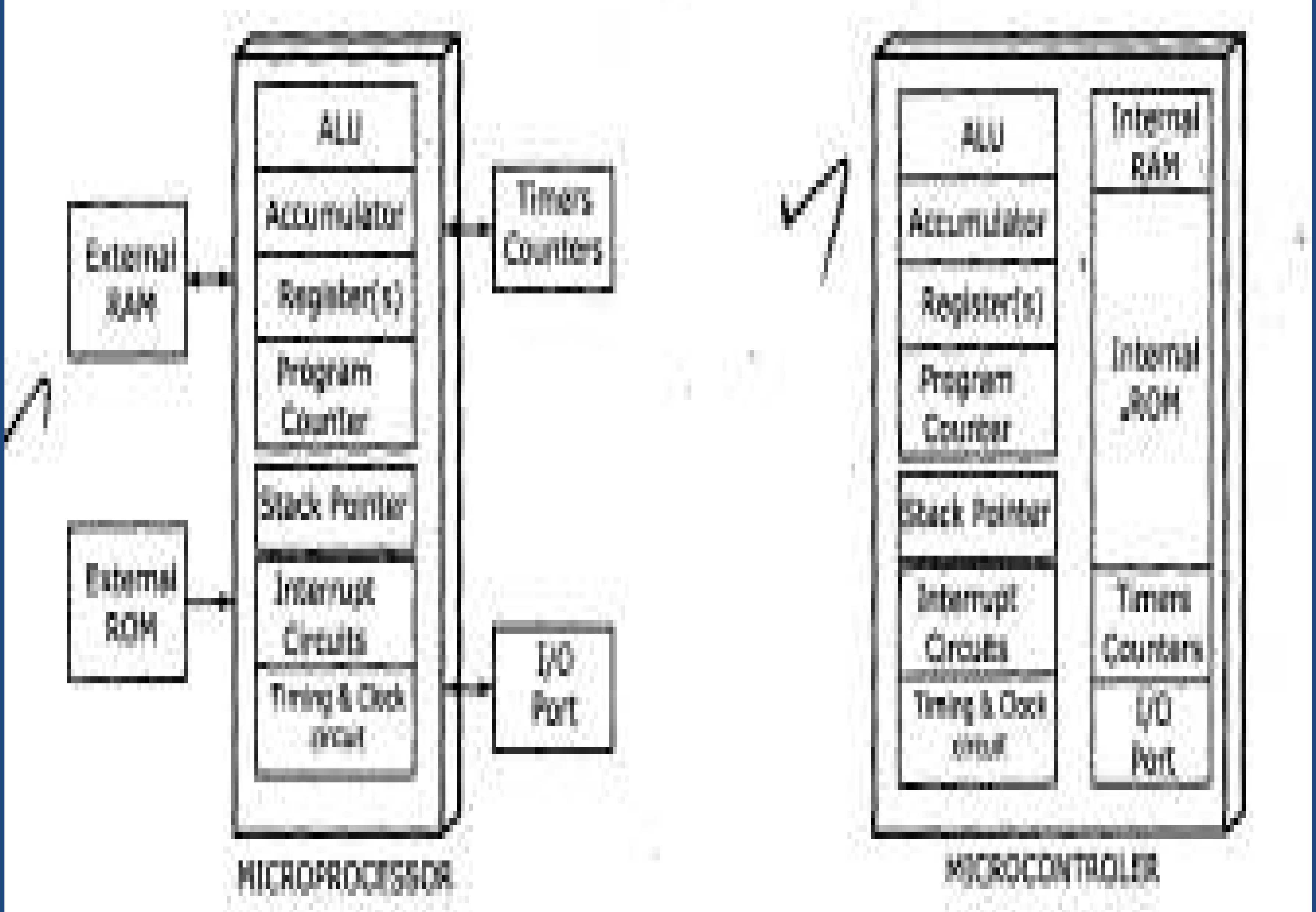


Figure 1-1. Microprocessor System Contrasted With Microcontroller System

1. CPU for Computers
2. No RAM, ROM, I/O on CPU chip itself
3. Example：Intel's x86, Motorola's 680x0

1. A smaller computer
2. On-chip RAM, ROM, I/O ports...
3. Example：Motorola's 6811, Intel's 8051, Zilog's Z8 and PIC


External RAM
External ROM
ALU
Accumulator
Register(s)
Program Counter
Stack Pointer
Interrupt Circuits
Timing & Clock circuit
Timers Counters
I/O Port
MICROPROCESSOR
ALU
Accumulator
Register(s)
Program Counter
Stack Pointer
Interrupt Circuits
Timing & Clock circuit
Internal RAM
Internal ROM
Timers Counters
I/O Port
MICROCONTROLLER

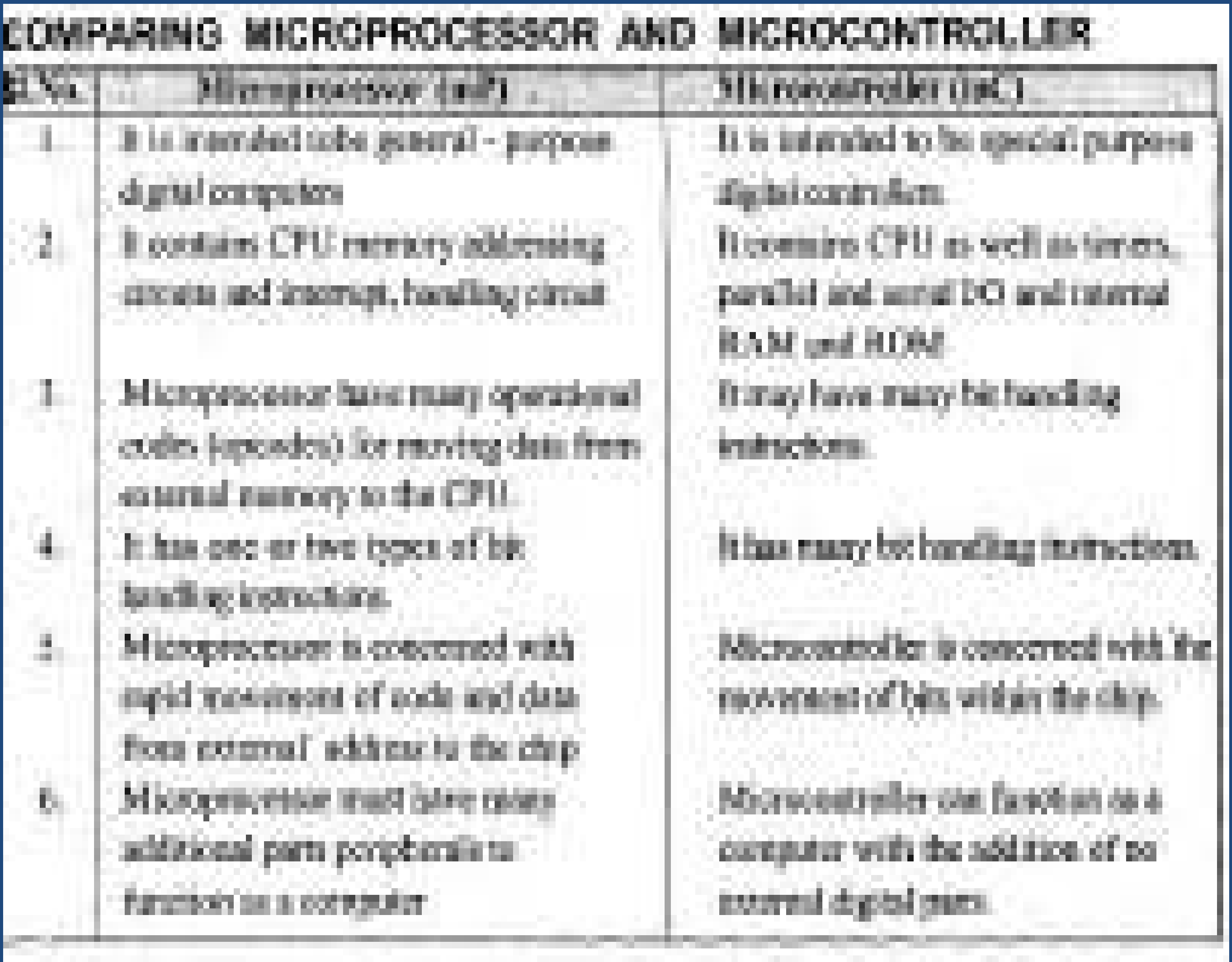
Microcontroller vs. Microprocessors

Microprocessor

1. CPU is stand-alone, RAM, ROM, I/O, timer are separate
2. designer can decide on the amount of ROM, RAM and I/O ports.
3. expansive
4. versatility
5. general-purpose

Microcontroller

1. CPU, RAM, ROM, I/O and timer are all on a single chip
2. fix amount of on-chip ROM, RAM, I/O ports
3. for applications in which cost, power and space are critical
4. single-purpose



COMPARING MICROPROCESSOR AND MICROCONTROLLER

S.No.	Microprocessor (mP)	Microcontroller (mC)
1.	It is intended to be general - purpose digital computer.	It is intended to be special purpose digital controller.
2.	It contains CPU memory addressing circuits and interrupt, handling circuit.	It contains CPU as well as timers, parallel and serial I/O and internal RAM and ROM.
3.	Microprocessor have many operational codes (opcodes) for moving data from external memory to the CPU.	It may have many bit handling instructions.
4.	It has one or two types of bit handling instructions.	It has many bit handling instructions.
5.	Microprocessor is concerned with rapid movement of code and data from external addresses to the chip	Microcontroller is concerned with the movement of bits within the chip.
6.	Microprocessor must have many additional parts peripherals to function as a computer	Microcontroller can function as a computer with the addition of no external digital parts.

uP vs. uC – cont.

Applications

- uCs are suitable to control of I/O devices in designs requiring a minimum component
- uPs are suitable to processing information in computer systems.

uP vs. uC – cont.

uC is easy to use and design.

- Only single chip can be a complete system
- interfacing to other devices,
  - for example, motors, displays, sensors, and communicate with PC.

In contrast, similar system that builds from uP would require a lot of additional units,

- such as RAM, UART, I/O , TIMER and etc.

Embedded Products Using Microcontrollers

Home

- Appliances, intercom, telephones, security systems, garage door openers, answering machines, fax machines, home computers, TVs, cable TV tuner, VCR, camcorder, remote controls, video games, cellular phones, musical instruments, sewing machines, lighting control, paging, camera, pinball machines, toys, exercise equipment

Embedded Products Using Microcontrollers

Office

- Telephones, computers, security systems, fax machines, microwave, copier, laser printer, color printer, paging

Embedded Products Using Microcontrollers

Auto

- Trip computer, engine control, air bag, ABS, instrumentation, security system, transmission control, entertainment, climate control, cellular phone, keyless entry

Choosing A Microcontroller

- Computing needs
  - Speed, packaging, power consumption, RAM, ROM, I/O pins, timers, upgrade to high performance or low-power versions, cost
- Software development tools
  - Assembler, debugger, C compiler, emulator, technical support
- Availability & source

Types of microcontrollers

- ARM core processors (many vendors)
- includes ARM9, ARM Cortex-A8, Sitara ARM Microprocessor
- Atmel AVR (8-bit), AVR32 (32-bit), and AT91SAM (32-bit)
- Cypress Semiconductor's M8C Core used in their PSoC
- Freescale ColdFire (32-bit) and S08 (8-bit)
- Freescale 68HC11 (8-bit)
- Intel 8051
- Infineon: 8, 16, 32 Bit microcontrollers[9]
- MIPS
- Microchip Technology PIC,
- NXP Semiconductors LPC1000, LPC2000, LPC3000, LPC4000
- Parallax Propeller
- PowerPC ISE
- Rabbit 2000 (8-bit)
- Silicon Laboratories Pipelined 8-bit 8051 Microcontrollers and …
- Texas Instruments TI MSP430 (16-bit)
- Toshiba TLCS-870 (8-bit/16-bit).

Various 8051 Microcontrollers

- 8751 microcontroller
  - UV-EPROM
- AT89C51 from *Atmel Corporation*
  - Flash (erase before write)
- DS5000 from *Dallas Semiconductor*
  - NV-RAM (changed one byte at a time), RTC (real-time clock)
- OTP (one-time-programmable) version of 8051
- 8051 family from *Philips*
  - AD, DA, extended I/O, OTP and flash

Companies Producing 8051

Table 1-2:Some Companies Producing a Member of the 8051 Family

Company	**Web Site**
Intel	www.intel.com/design/mcs51
Atmel	www.atmel.com
Philips/Signetics	www.semiconductors.philips.com
Siemens	www.sci.siemens.com
Dallas Semiconductor	www.dalsemi.com

Inside 8051 Microcontroller

Introduced by Intel in 1981

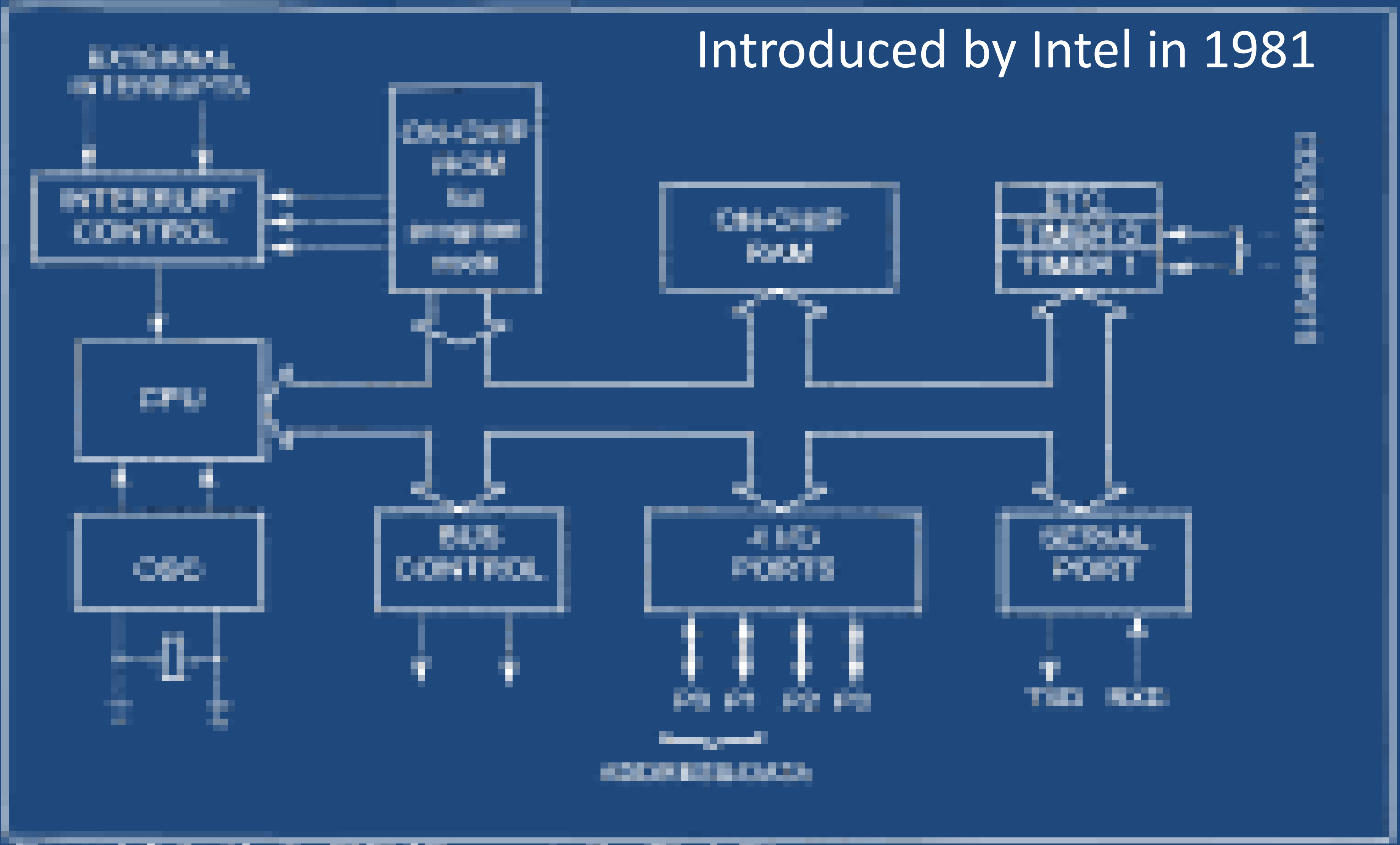
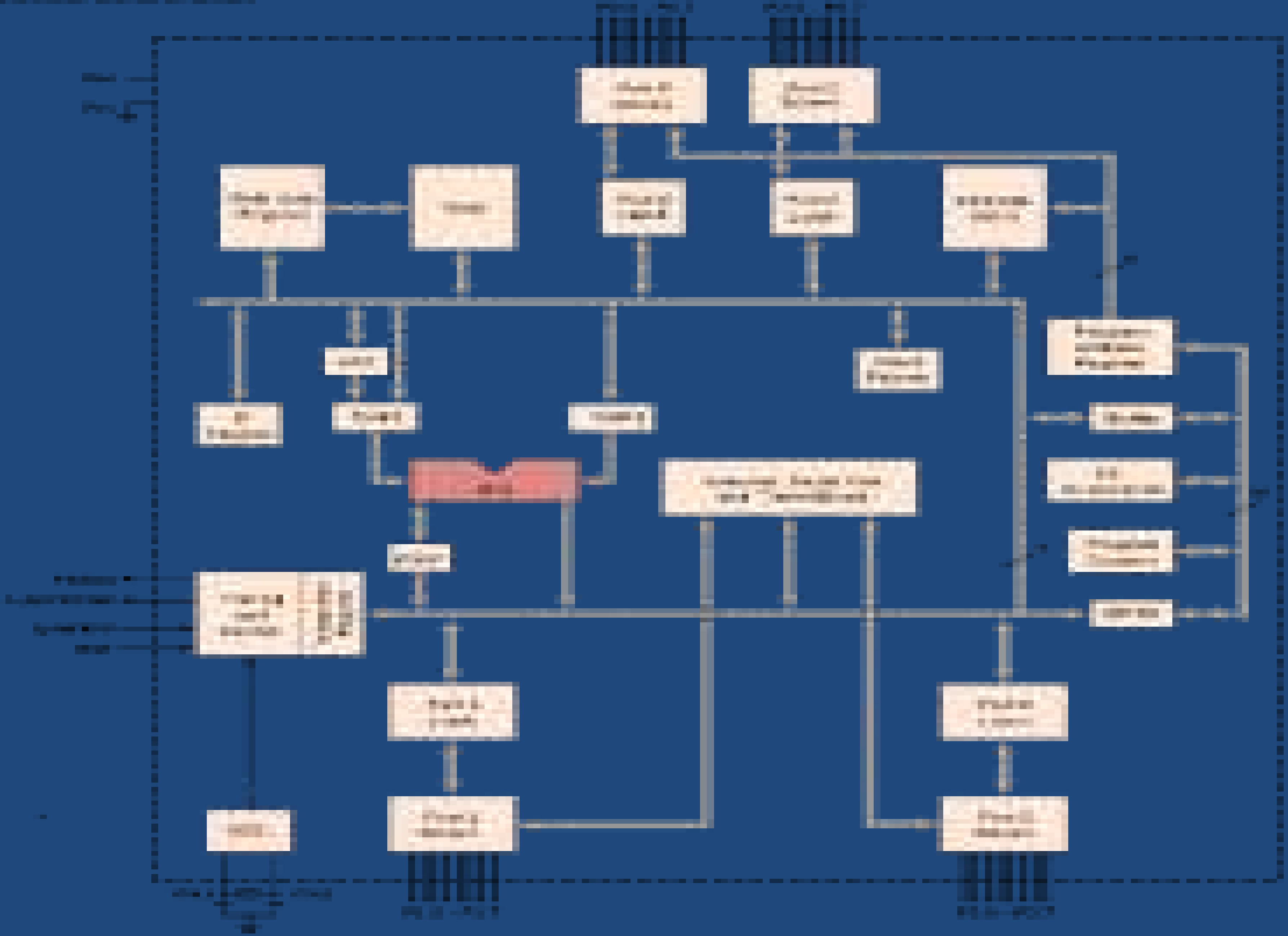


Figure 1-2. Inside the 8051 Microcontroller Block Diagram

ARCHITECTURE

- ❑Memory Organization
- ❑CPU Clock
- ❑Interrupt Structure
- ❑Port Structures
- ❑Timer/Counters
- ❑Reset

[illegible]



Functional Description Of Each Block :

- **Accumulator: (Acc)** Acc is an 8 bit special function register. It acts an operand register. Result is temporarily stored in this register. It is used in parallel I/O transfer.
- **B Register :** B register is 8 bit SFR. It is used during multiply and divide operations. For other operations , it can be used as a scratchpad register.
- **Program Status Word (PSW) :** PSW register is 8 bit SFR. It contains program status information. It is also used to select any one of the required register bank.
- **Stack Pointer (SP):** It is 8 bit register. It is used to point the stack memory. The stack may reside in anywhere in on-chip memory. It is incremented before data is stored during PUSH & CALL instructions. After reset SP is initialized to 07h. This causes the stack begin at location 08h.
- **Data Pointer (DPTR) :** It is 16 bit register. It may be manipulated as a 16 bit register or as two independent 8 bit registers. Its function is to hold a 16 bit address. This register is used for external reference.

CONTINUED..

- **Port 0 to Port 3 :** Each port contains separate address. Using this address,User can communicate with these ports. Each port contains latch,output driver & input buffer.

- **Serial Data Buffer :** Serial data buffer contains two independent registers of a transmit buffer register and a receiver buffer register.
  - Transmit buffer is a parallel in and Serial out register.
  - Receiver buffer is a Serial in and parallel out register.
  - When data is moved to SBUF, it goes to transmit buffer .
  - When data is moved from SBUF, it comes from the receive buffer.

- **Timer Registers :** Register pairs (TH0,TL0) & (TH1,TL1) are the two 16 bit counting registers for Timer/Counter 0 and 1 respectively.

CONTINUED...

- **Control Registers :** The special function registers IP,IE , TMOD, TCON SCON and PCON contain control and status information for interrupts, timer/counters and serial port.

- **Timing & Control Unit :** This unit derives an necessary timing and control signals required for the internal operations of the circuit. It derives control signals required for controlling the external system bus. The interrupt, serial port and timer circuits are controlled by the control signals generated by timing & control unit.

- **Oscillator :** This circuit generates the basic timing clock signal for the operation of the circuit using crystal oscillator.

- **Instruction Register :** This register decodes the opcode of an instruction to be executed and gives information to the timing & control unit, and to generate necessary signals for the execution of the instruction.

CONTINUED.....

- **EPROM & Program Address Register :** These blocks provide an onchip EPROM and a Mechanism to internally address it.
- **RAM & RAM address register :** These blocks provide an onchip RAM and a mechanism to internally address it.
- **ALU :** The **Arithmetic And Logic Unit** performs 8 bit arithmetic and logical operations over the operands held by the temporary registers TMP 1 and TMP 2. User cannot access these temporary registers
- **SFR Register Banks :** It is a set of registers, which can be addressed using their respective addresses which lie in the range 80h to FFh.

XTAL Connection to 8051

- Find the machine cycle for
- (a) XTAL = 11.0592 MHz
- (b) XTAL = 16 MHz.
- **Solution:**

(a) 11.0592 MHz / 12

= 921.6 kHz;

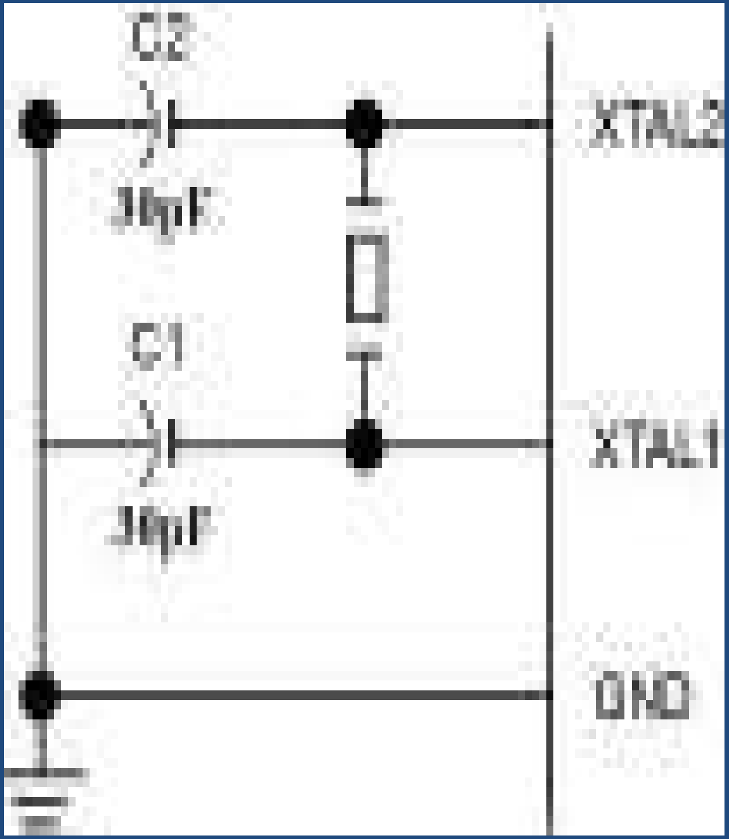
machine cycle = 1 / 921.6 kHz

= 1.085 µ

(b) 16 MHz / 12 = 1.333 MHz;

machine cycle = 1 / 1.333 MHz

= 0.75 µs



Power-On RESET

- RST（pin 9） ： reset

(i) input pin and active high （normally low）.

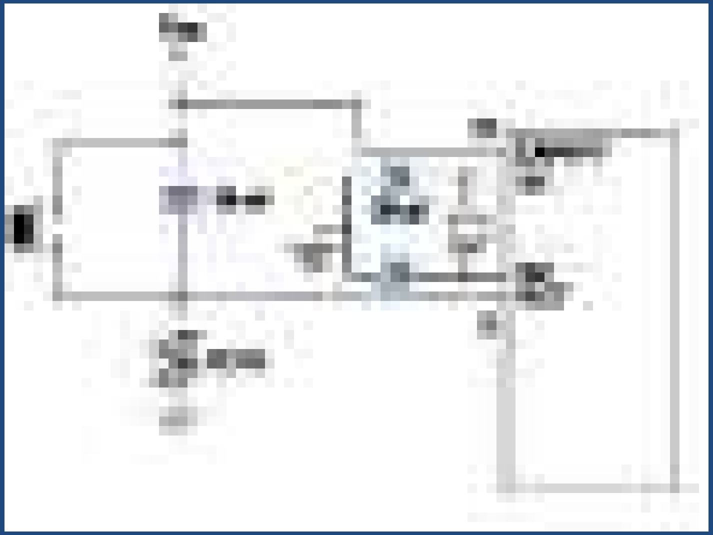
- The high pulse must be high at least 2 machine cycles.

(ii)power-on reset.

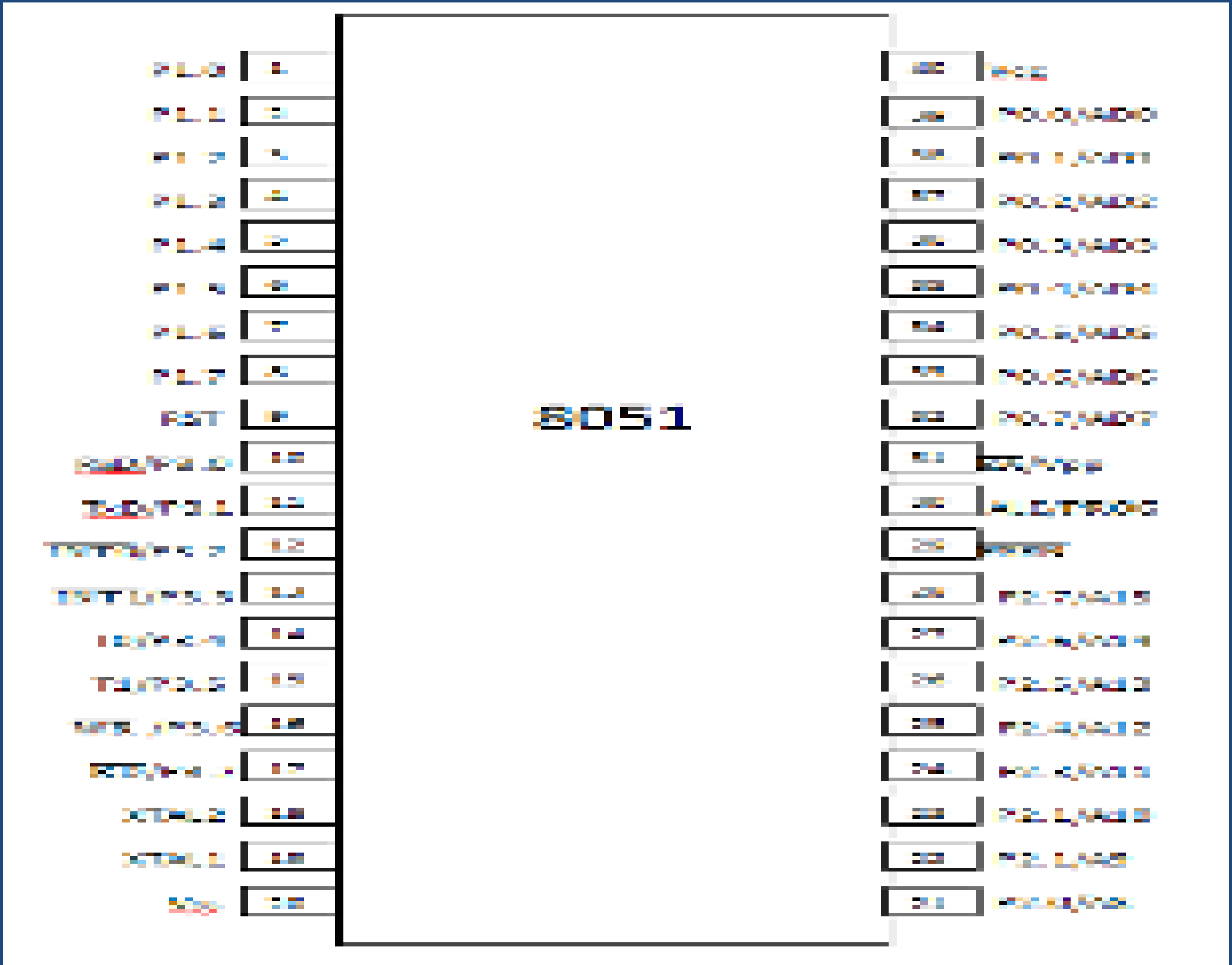
Upon applying a high pulse to RST, the microcontroller will reset and all values in registers will be lost. Reset values of some 8051 registers

(A) ← 00 ;(SP) ←07

- (iii)power-on reset circuit -as shown in figure



PIN DIAGRAM OF 8051



PIN DETAILS

- **Pins 1 – 8:-** Known as Port 1. Unlike other ports, this port does not serve any other functions. Port 1 is an internally pulled up, quasi bi directional I/O port.
- **Pin 9:-** As explained before RESET pin is used to set the 8051 microcontroller to its initial values, while the microcontroller is working or at the initial start of application. To reset the microcontroller ,the reset pin must be set high for 2 machine cycles.

CONTD…

- **Pins 10 – 17:-** Known as Port 3. This port also serves some other functions like interrupts, timer input, control signals for external memory interfacing RD and WR , serial communication signals RxD and TxD etc. This is a quasi bi directional port with internal pull up resister. It is also called multifunctional port
- **Pins 18 and 19:-** Used for connecting an crystal externally to provide system clock.
- **Pin 20:-** Named as Vss – it represents ground (0 V) connection.

CONTD…

- **Pins- 21-28:-** Known as Port 2 (P 2.0 to P 2.7) – in addition to serving as I/O port, higher order address bus signals are multiplexed with this port.
- **Pin- 29:-** PSEN or Program Store Enable is used to read data from external program memory.
- **Pin-30:-** ALE Address Latch Enable is used to de multiplex the address and data signal of port 0 (for external memory interfacing.) When address moves on port 0,ALE will be high. It is the indication to hold the Address in latch .
- **Pin-31:-** EA External Access input is used to enable or disable external memory interfacing. It is low enable pin. If there is no external memory requirement, this pin is pulled high by connecting it to Vcc.

CONTD…

- **Pins 32-39:** Known as Port 0 (P0.0 to P0.7) – In addition to serving as I/O port, lower order address and data are multiplexed with this port (16 bit address is used for the purpose of external memory interfacing). This is a bi directional I/O port and external pull up resistors are required to use this port as I/O.
- **Pin-40 :** This pin is named as VCC. Usually +5V DC is given to this pin.

8051 Family

Table 1-4:Comparison of 8051 Family Members

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

Thank You

SESHASAYEE INSTITUTE OF TECHNOLOGY
ARIYAMANGALAM , TRICHY – 620 010

ARCHITECTURE OF 8051 & THEIR PIN DETAILS

UNIT I

WELCOME

ARCHITECTURE OF 8051 & THEIR PIN DETAILS

U1.1 : Introduction to microprocessor & microcontroller : Architecture of 8085 -Functions of each block. Comparison of Microprocessor & Microcontroller - Features of microcontroller -Advantages of microcontroller -Applications Of microcontroller -Manufactures of microcontroller.

U1.2 : Architecture of 8051 : Block diagram of Microcontroller – Functions of each block. Pin details of 8051 -Oscillator and Clock -Clock Cycle -State - Machine Cycle -Instruction cycle –Reset - Power on Reset - Special function registers :Program Counter -PSW register -Stack - I/O Ports .

U1.3 : **Memory Organisation & I/O port configuration:** ROM RAM - Memory Organization of 8051,Interfacing external memory to 8051

1.3 MEMORY ORGANISATION

- The 8051 architecture provides both on chip memory as well as off chip memory expansion capabilities.
- It supports several distinctive ‘physical’ address spaces, functionally EA separated at the hardware level by different addressing mechanisms, read and write controls signals or both.
- The 8051 has on chip memory of 4 Kbytes of internal Program Memory and 128 bytes of internal Data memory.
- It can access upto 64 K program memory and 64Kdata memory.

The 3 different address spaces of 8051

i) 64 Kbyte program memory

ii) 64 Kbyte external data memory

iii) 256 byte internal data memory.

<u>**Internal Data Memory**</u> (256bytes) is divided into two physically separate and distinct blocks.

i) 128 Bytes Internal RAM Area. This area is further divided into three.

a) 4 Register banks :

[each bank has eight 8 bit registers]

(address range 00 – 1Fh)

b) 16 bytes bit addressable locations :

(address range 20h – 2Fh)

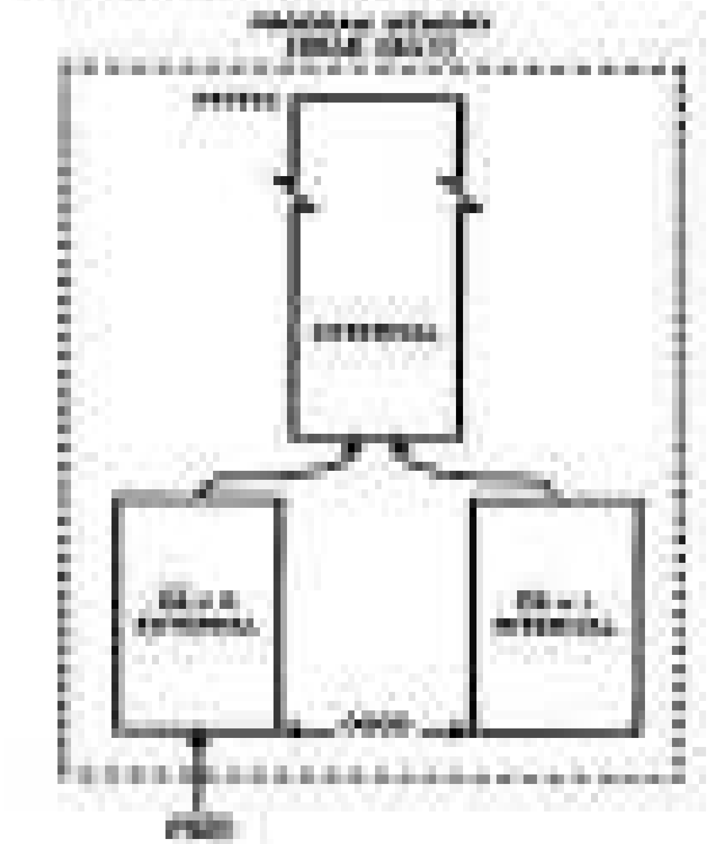
c) 80 bytes byte addressable locations

(address range 30h – 7Fh)

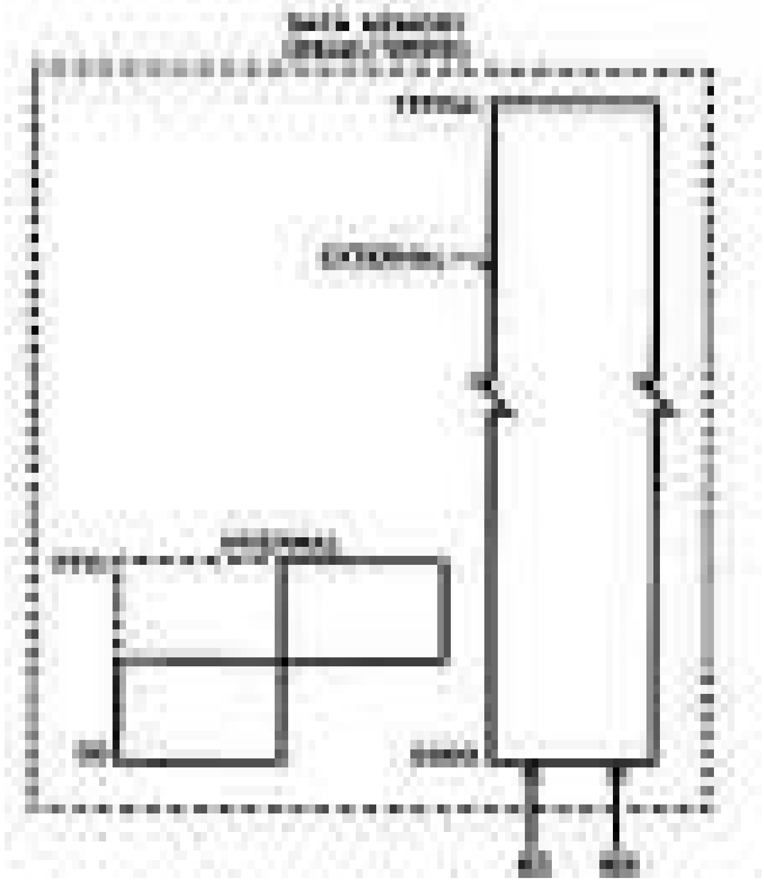
ii) **128 Bytes SFR Area (address range 80h – 0FFh**)

1.3 MEMORY ORGANISATION

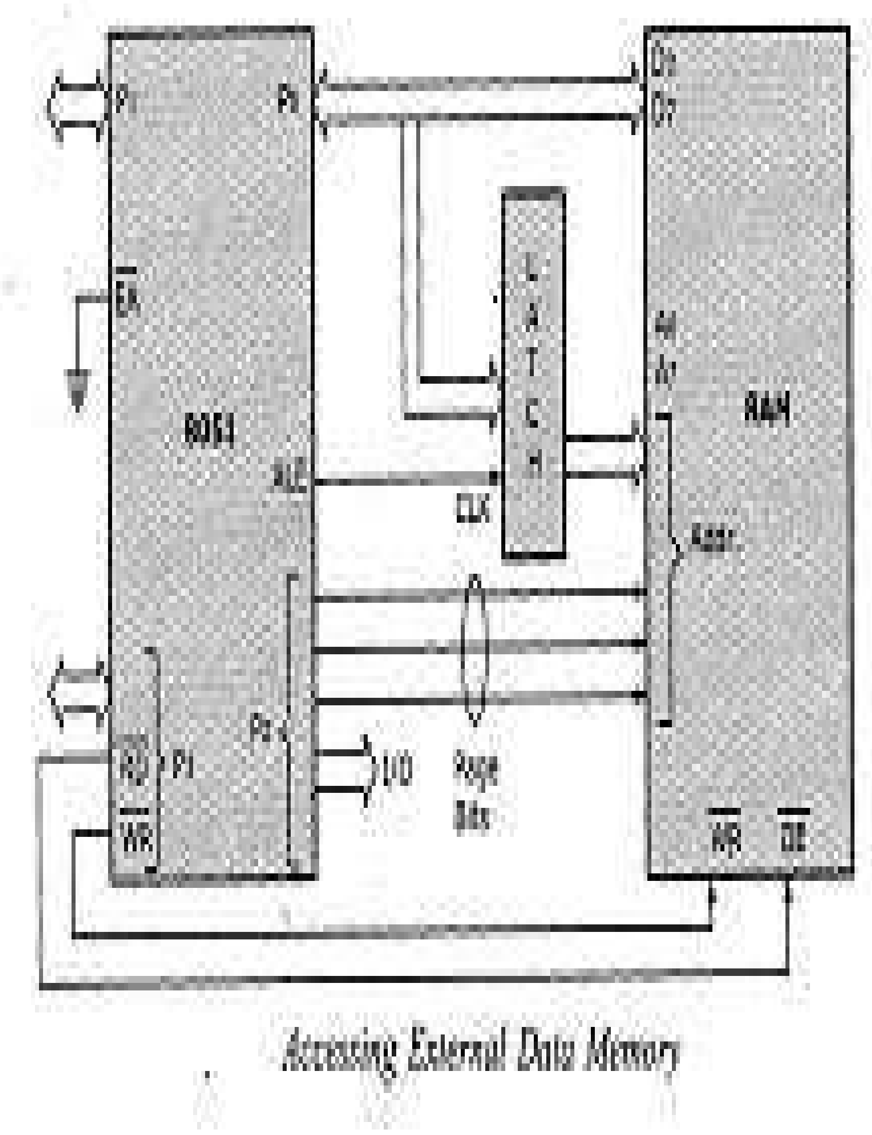
Program memory(Read Only)



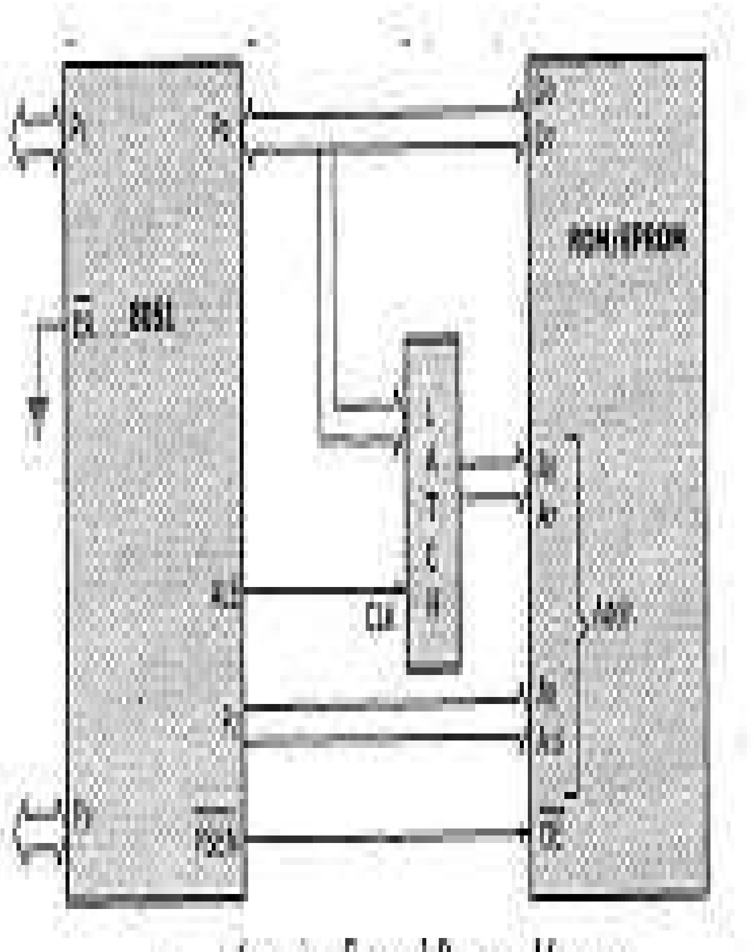
Data Memory(Read/Write)



CONNECTING EXTERNAL MEMORY TO MICROCONTROLLER

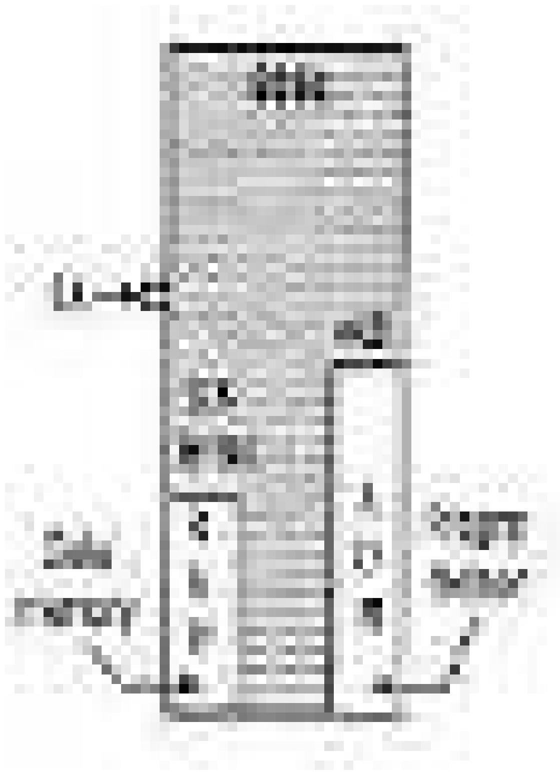


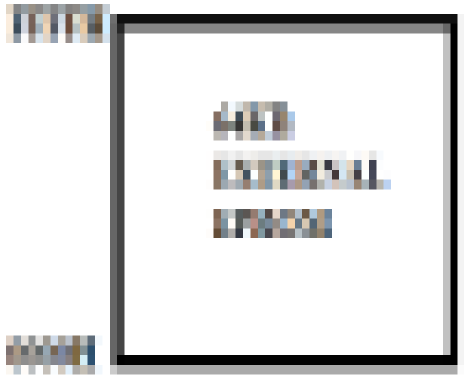
Accessing External Data Memory

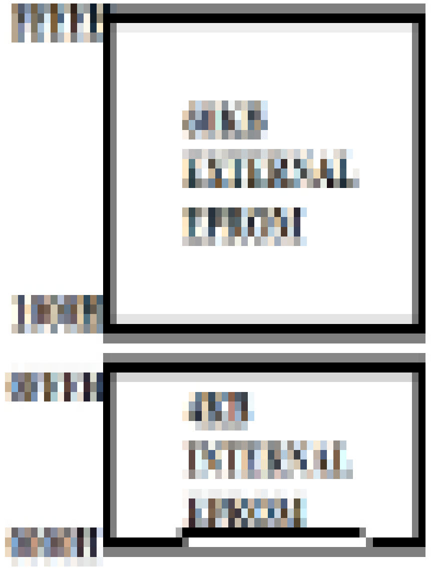


Accessing External Program Memory

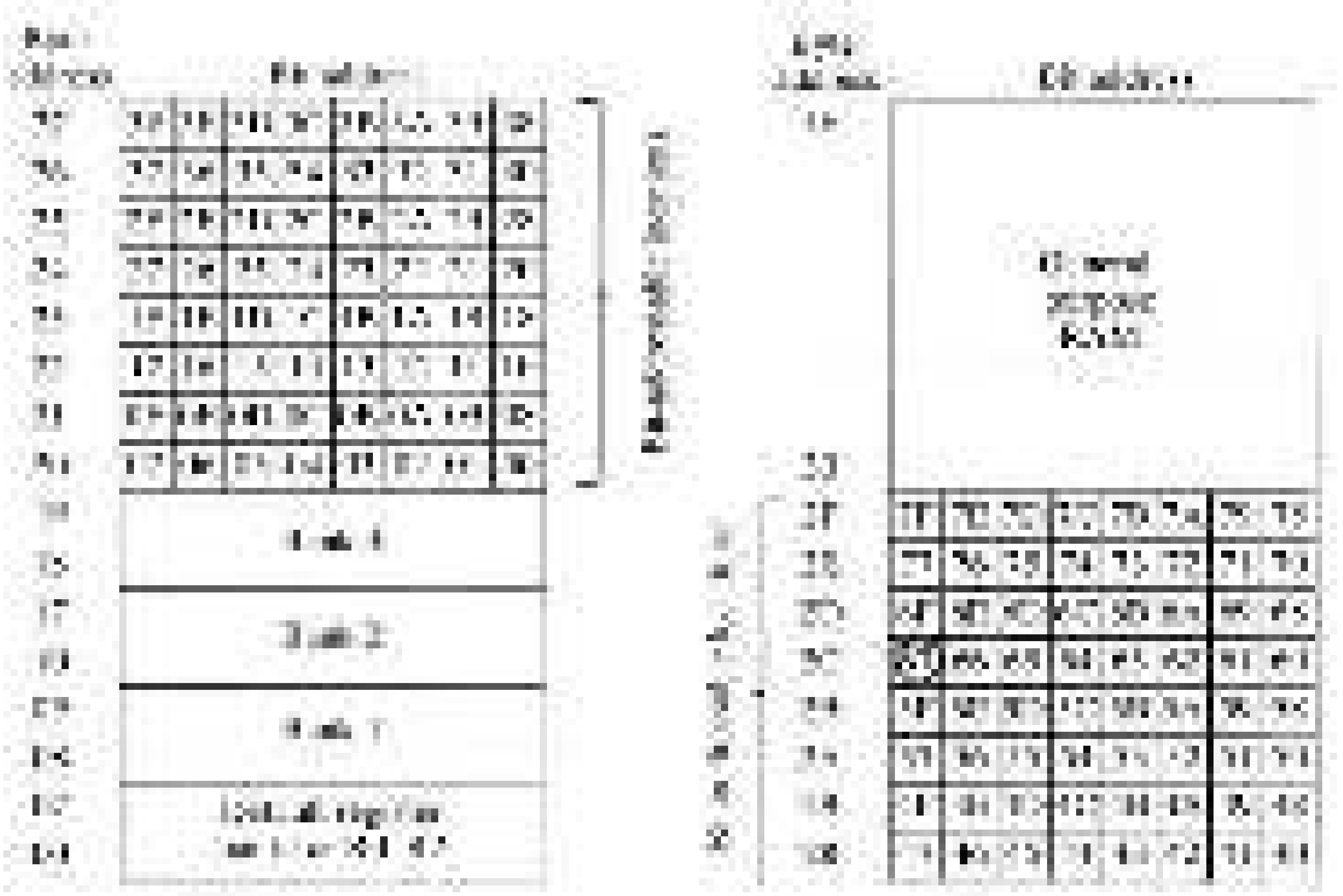
MEMORY ORGANISATION CONTD..



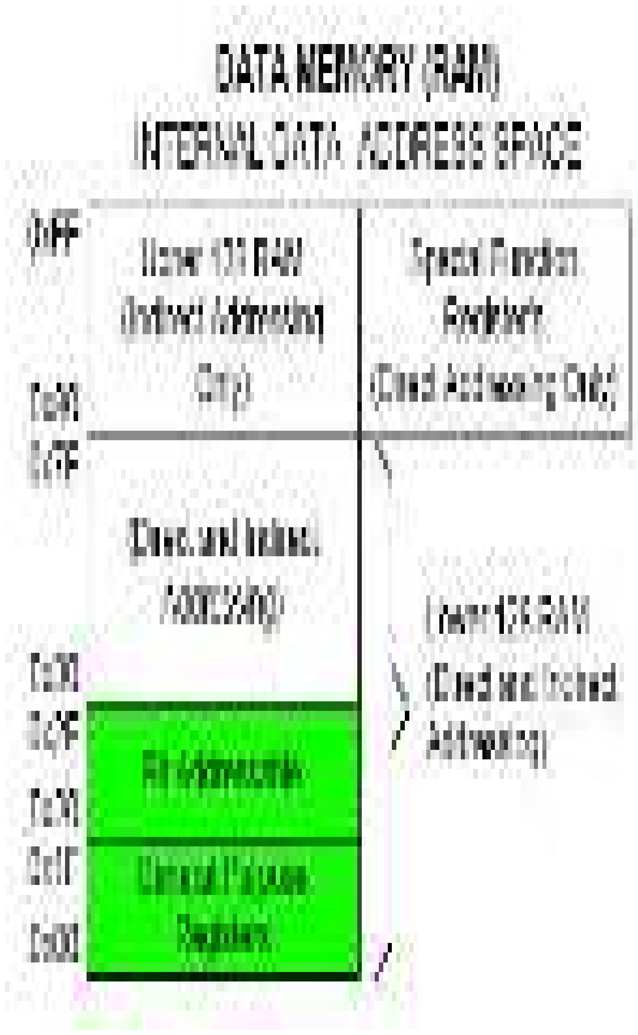


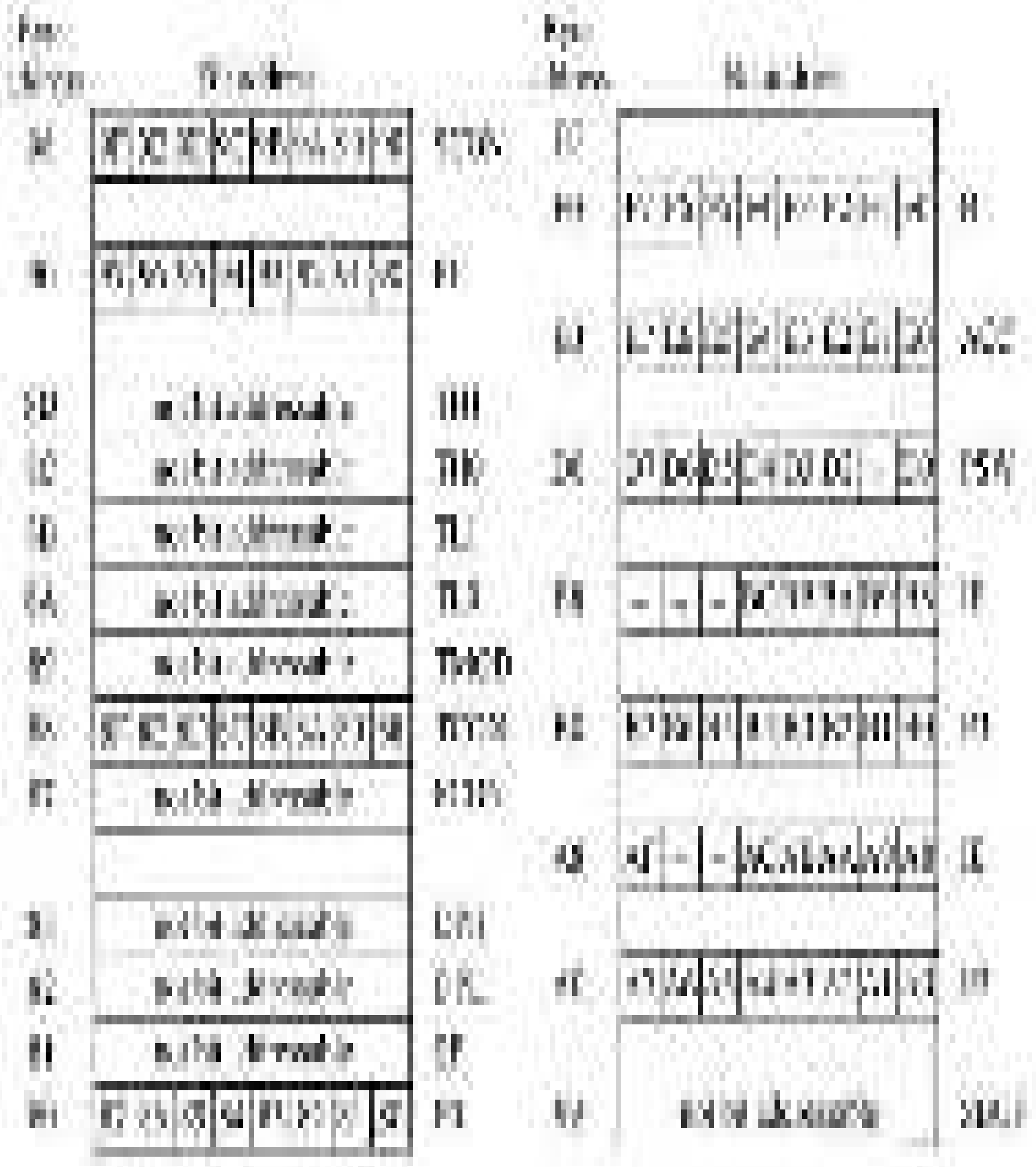


INTERNAL RAM



INTERNAL RAM


DATA MEMORY (RAM)
INTERNAL DATA ADDRESS SPACE
0xFF
Upper 128 RAM (Indirect Addressing Only)
Special Function Registers (Direct Addressing Only)
0x80
0x7F
Direct and Indirect Addressing
Lower 128 RAM (Direct and Indirect Addressing)
0x30
0x2F
Bit Addressable
0x20
0x1F
General Purpose Registers
0x00



SPECIAL FUNCTION REGISTERS (SFRS)

- Special Function Registers (SFRs) are a sort of control table used for running and monitoring the operation of the microcontroller.
- Each of these registers as well as each bit they include, has its name, address in the scope of RAM and precisely defined purpose such as timer control, interrupt control, serial communication control etc.. All types of 8051 microcontrollers, has only 21 such registers..

SPECIAL FUNCTION REGISTERS

SN	SFR	NAME	BYTE ADDRESS IN HEXA	SN	SFR	NAME	BYTE ADDRESS IN HEXA
01	ACC*	Accumulator	0E0	12	IE*	InterruptEnable control	0A8
02	B*	B register	0F0	13	TMOD	Timer mode control	89
03	PSW*	Program status word	0D0	14	TCON*	Timer control	88
04	SP	Stack pointer	81	15	TH0	Timer/counter0 high byte	8C
05	DPL	Low byte	82	16	TL0	Timer/counter0 low byte	8A
06	DPH	High byte	83	17	TH1	Timer/counter1 high byte	8D
07	P0*	Port 0	80	18	TL1	Timer/counter1 low byte	8B
08	P1*	Port 1	90	19	SCON*	Serial control	98
09	P2*	Port 2	0A0	20	SBUF	SerialData buffer	99
10	P3*	Port 3	0B0	21	PCON	Power control	87
11	IP*	Interrupt Priority control	0B8	22	PC	Program Counter (No address)	

SFR’s (BOTH BYTE AND BIT ADDRESSABLE REGISTERS)

SN	SFR	NAME	BYTE ADDRESS IN HEXA	BIT ADDRESS IN HEXA
01	ACC*	Accumulator	0E0	E7 – E0
02	B*	B register	0F0	F7 – F0
03	PSW*	Program status word	0D0	D7 – D0
04	P0*	Port 0	80	87 – 80
05	P1*	Port 1	90	97 – 90
06	P2*	Port 2	0A0	A7 – A0
07	P3*	Port 3	0B0	B7 – B0
08	IP*	Interrupt Priority control	0B8	BF – B8
09	IE*	Interrupt Enable control	0A8	AF – A8
10	TCON*	Timer control	88	8F – 88
11	SCON*	Serial control	98	9F - 98

FORMAT OF PSW

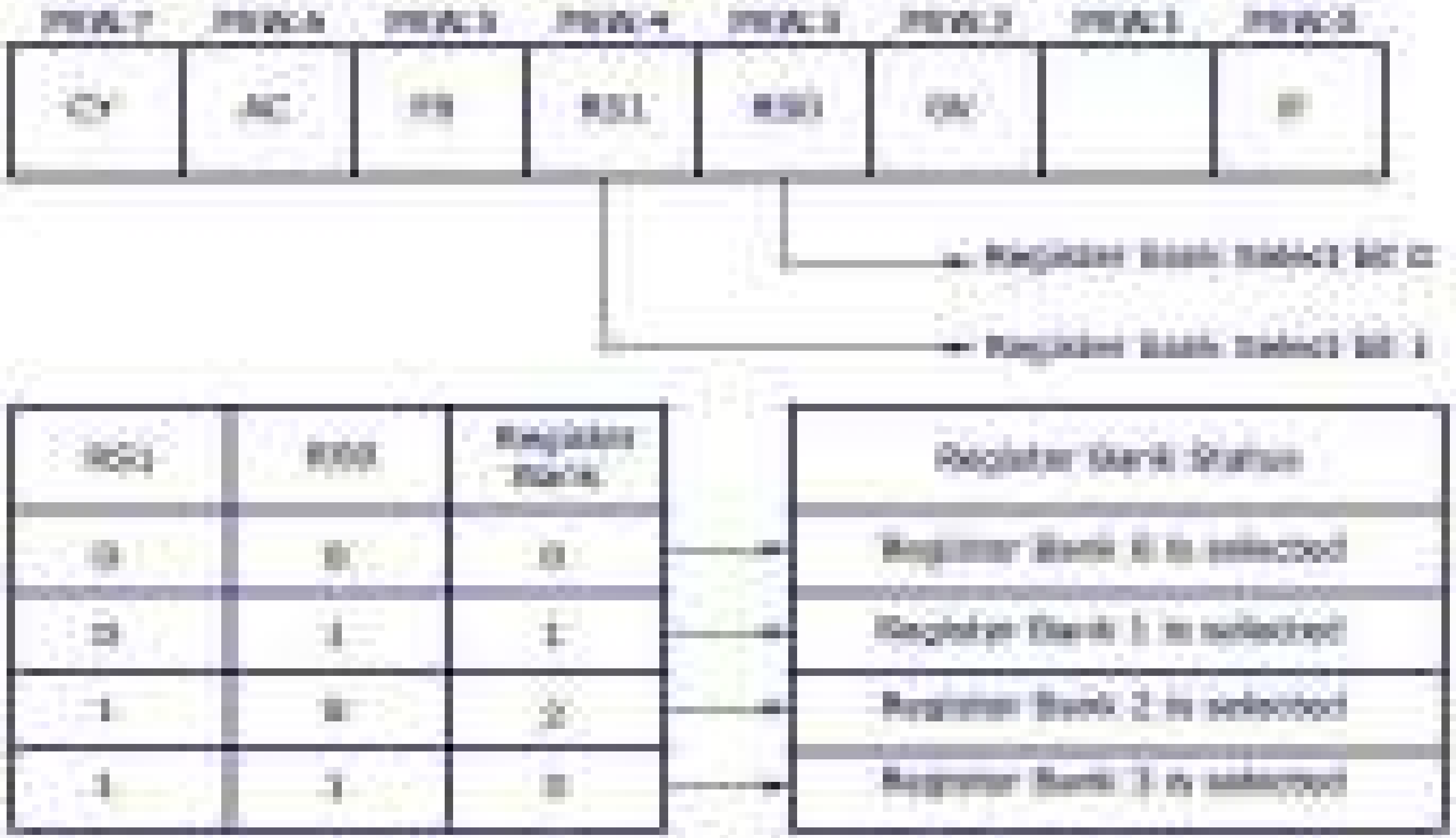
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	OV	-	P

BIT POSITION

BIT ADDRESS

BIT NAME

Processor Status Word



RS1	RS0	Register Bank	Register Bank Status
0	0	0	Register Bank 0 is selected
0	1	1	Register Bank 1 is selected
1	0	2	Register Bank 2 is selected
1	1	3	Register Bank 3 is selected

DETAILS OF PSW

Bit	Symbol	Address	Description
PSW.7	CY	D7H	Carry flag
PSW.6	AC	D6H	Auxiliary carry flag
PSW.5	F0	D5H	Flag 0
PSW.4	RS1	D4H	Register bank select 1
PSW.3	RS0	D3H	Register bank select 0
PSW.2	OV	D2H	Overflow flag
PSW.1	--	D1H	Reserved
PSW.0	P	D0H	Even parity flag

Carry Flag

The carry flag has two functions.

- It is used as the carry-out in 8-bit addition/subtraction.

  For example, if the accumulator contains FDH and we add 3 to the contents of the accumulator (ADD A, #3), the accumulator will then contain zero and the carry flag will be set.

  It is also set if a subtraction causes a borrow into bit 7. In other words, if a number is subtracted from another number smaller than it, the carry flag will be set.

  For example, if A contains 3DH and R3 contains 4BH, the instruction SUBB A, R3 will result in the carry bit being set (4BH is greater than 3DH).

- The carry flag is also used during Boolean operations.

  For example AND the contents of bit 3DH with the carry flag, the result being placed in the carry flag - ANL C, 3DH

Register Bank Select Bits

Bits 3 and 4 of the PSW are used for selecting the register bank. Since there are four register banks, two bits are required for selecting a bank, as detailed below.

REGISTER BANK SELECTION

PSW.4 RS1	PSW.3 RS0	SELECTED Register Bank	Address of Register Bank
0	0	0	00H to 07H
0	1	1	08H to 0FH
1	0	2	10H to 17H
1	1	3	18H to 1FH

Flag 0 : Flag 0 is a general-purpose flag available to the programmer.

Parity Bit : The parity bit is automatically set or cleared in every machine cycle to ensure even parity with the accumulator. The number of 1-bits in the accumulator plus the parity bit is always even.

In other words, if the number of 1s in the accumulator is odd ➔ parity bit is set to 1.

if the number of 1s in the accumulator is even ➔ parity bit is set to 0.

For example, if the accumulator holds the number 05H, this is 0000 0101 in binary => the accumulator has an even number of 1s, therefore the parity bit is cleared.

If the accumulator holds the number F2H, this is 1111 0010 => the accumulator has an odd number of 1s, therefore the parity bit is set to make the overall number of 1s even.

CY, the carry flag

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

AC, the auxiliary carry flag

- If there is a carry from **D3 to D4 during an ADD or SUB** operation, this bit is set; otherwise, it is cleared.
- This flag is used by instructions that perform BCD (binary coded decimal) arithmetic.

OV, the overflow flag

- This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
- In general, the carry flag is used to detect errors in unsigned arithmetic operations.
- The overflow flag is only used to detect errors in signed arithmetic operations

Thank You

SESHASAYEE INSTITUTE OF TECHNOLOGY
ARIYAMANGALAM , TRICHY – 620 010

ARCHITECTURE OF 8051 & THEIR PIN DETAILS

UNIT I

WELCOME

ARCHITECTURE OF 8051 & THEIR PIN DETAILS

U1.1 : Introduction to microprocessor & microcontroller : Architecture of 8085 -Functions of each block. Comparison of Microprocessor & Microcontroller - Features of microcontroller -Advantages of microcontroller -Applications Of microcontroller -Manufactures of microcontroller.

U1.2 : Architecture of 8051 : Block diagram of Microcontroller – Functions of each block. Pin details of 8051 -Oscillator and Clock -Clock Cycle -State - Machine Cycle -Instruction cycle –Reset - Power on Reset - Special function registers :Program Counter -PSW register -Stack - I/O Ports .

U1.3 : **Memory Organisation & I/O port configuration:** ROM RAM - Memory Organization of 8051,Interfacing external memory to 8051

SPECIAL FUNCTION REGISTERS (SFRS)

- Special Function Registers (SFRs) are a sort of control table used for running and monitoring the operation of the microcontroller.
- Each of these registers as well as each bit they include, has its name, address in the scope of RAM and precisely defined purpose such as timer control, interrupt control, serial communication control etc.. All types of 8051 microcontrollers, has only 21 such registers..

SPECIAL FUNCTION REGISTERS

SN	SFR	NAME	BYTE ADDRESS IN HEXA	SN	SFR	NAME	BYTE ADDRESS IN HEXA
01	ACC*	Accumulator	0E0	12	IE*	InterruptEnable control	0A8
02	B*	B register	0F0	13	TMOD	Timer mode control	89
03	PSW*	Program status word	0D0	14	TCON*	Timer control	88
04	SP	Stack pointer	81	15	TH0	Timer/counter0 high byte	8C
05	DPL	Low byte	82	16	TL0	Timer/counter0 low byte	8A
06	DPH	High byte	83	17	TH1	Timer/counter1 high byte	8D
07	P0*	Port 0	80	18	TL1	Timer/counter1 low byte	8B
08	P1*	Port 1	90	19	SCON*	Serial control	98
09	P2*	Port 2	0A0	20	SBUF	SerialData buffer	99
10	P3*	Port 3	0B0	21	PCON	Power control	87
11	IP*	Interrupt Priority control	0B8	22	PC	Program Counter (No address)	

SFR’s (BOTH BYTE AND BIT ADDRESSABLE REGISTERS)

SN	SFR	NAME	BYTE ADDRESS IN HEXA	BIT ADDRESS IN HEXA
01	ACC*	Accumulator	0E0	E7 – E0
02	B*	B register	0F0	F7 – F0
03	PSW*	Program status word	0D0	D7 – D0
04	P0*	Port 0	80	87 – 80
05	P1*	Port 1	90	97 – 90
06	P2*	Port 2	0A0	A7 – A0
07	P3*	Port 3	0B0	B7 – B0
08	IP*	Interrupt Priority control	0B8	BF – B8
09	IE*	Interrupt Enable control	0A8	AF – A8
10	TCON*	Timer control	88	8F – 88
11	SCON*	Serial control	98	9F - 98

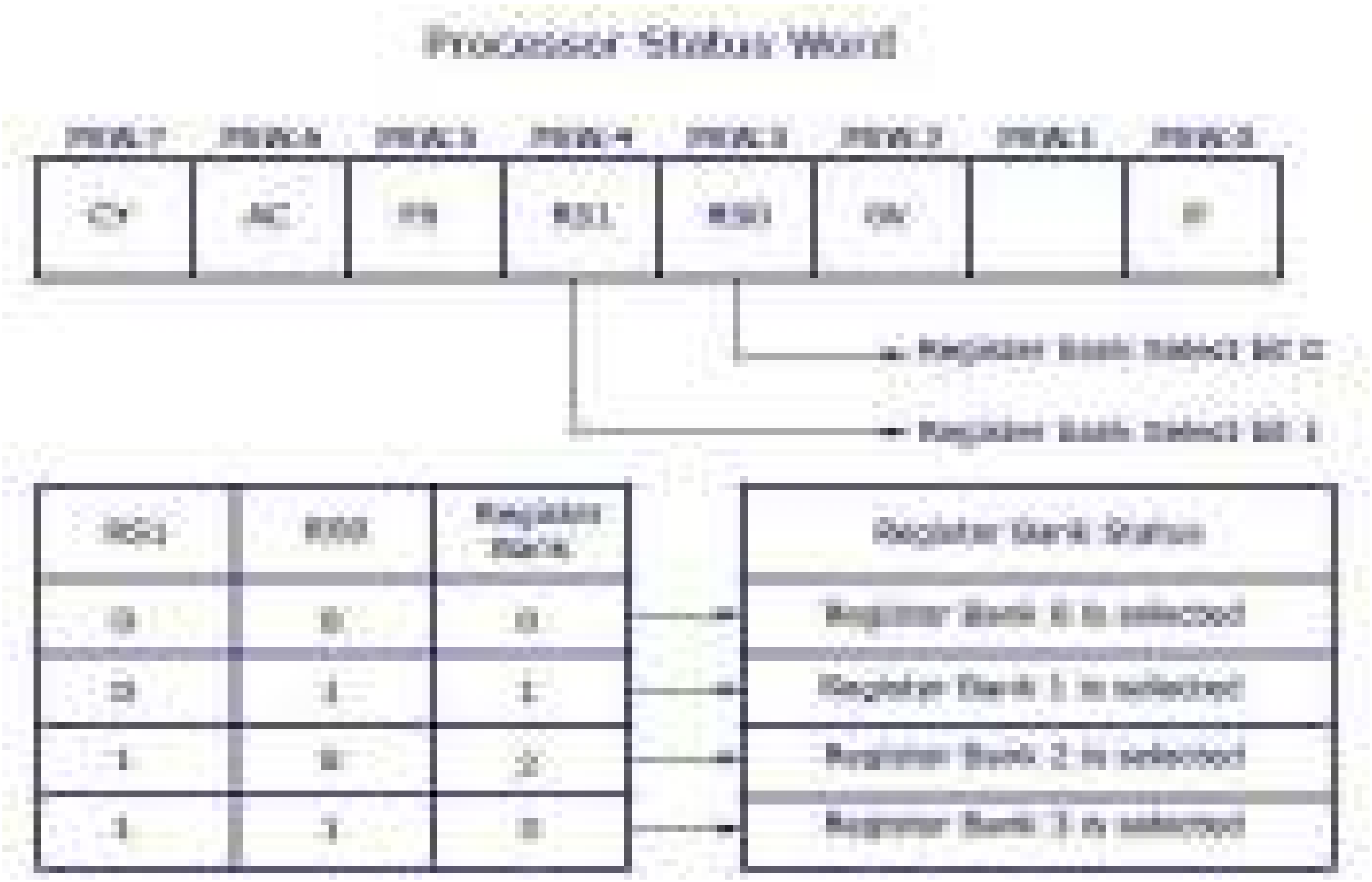
FORMAT OF PSW

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	OV	-	P

BIT POSITION

BIT ADDRESS

BIT NAME



RS1	RS0	Register Bank	Register Bank Status
0	0	0	Register Bank 0 is selected
0	1	1	Register Bank 1 is selected
1	0	2	Register Bank 2 is selected
1	1	3	Register Bank 3 is selected

DETAILS OF PSW

Bit	Symbol	Address	Description
PSW.7	CY	D7H	Carry flag
PSW.6	AC	D6H	Auxiliary carry flag
PSW.5	F0	D5H	Flag 0
PSW.4	RS1	D4H	Register bank select 1
PSW.3	RS0	D3H	Register bank select 0
PSW.2	OV	D2H	Overflow flag
PSW.1	--	D1H	Reserved
PSW.0	P	D0H	Even parity flag

Carry Flag

The carry flag has two functions.

- It is used as the carry-out in 8-bit addition/subtraction.
  For example, if the accumulator contains FDH and we add 3 to the contents of the accumulator (ADD A, #3), the accumulator will then contain zero and the carry flag will be set.
  It is also set if a subtraction causes a borrow into bit 7. In other words, if a number is subtracted from another number smaller than it, the carry flag will be set.
  For example, if A contains 3DH and R3 contains 4BH, the instruction SUBB A, R3 will result in the carry bit being set (4BH is greater than 3DH).
- The carry flag is also used during Boolean operations.
  For example AND the contents of bit 3DH with the carry flag, the result being placed in the carry flag - ANL C, 3DH

Register Bank Select Bits

Bits 3 and 4 of the PSW are used for selecting the register bank. Since there are four register banks, two bits are required for selecting a bank, as detailed below.

REGISTER BANK SELECTION

PSW.4 RS1	PSW.3 RS0	SELECTED Register Bank	Address of Register Bank
0	0	0	00H to 07H
0	1	1	08H to 0FH
1	0	2	10H to 17H
1	1	3	18H to 1FH

Flag 0 : Flag 0 is a general-purpose flag available to the programmer.

Parity Bit : The parity bit is automatically set or cleared in every machine cycle to ensure even parity with the accumulator. The number of 1-bits in the accumulator plus the parity bit is always even.

In other words, if the number of 1s in the accumulator is odd ➔ parity bit is set to 1.

if the number of 1s in the accumulator is even ➔ parity bit is set to 0.

For example, if the accumulator holds the number 05H, this is 0000 0101 in binary => the accumulator has an even number of 1s, therefore the parity bit is cleared.

If the accumulator holds the number F2H, this is 1111 0010 => the accumulator has an odd number of 1s, therefore the parity bit is set to make the overall number of 1s even.

CY, the carry flag

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

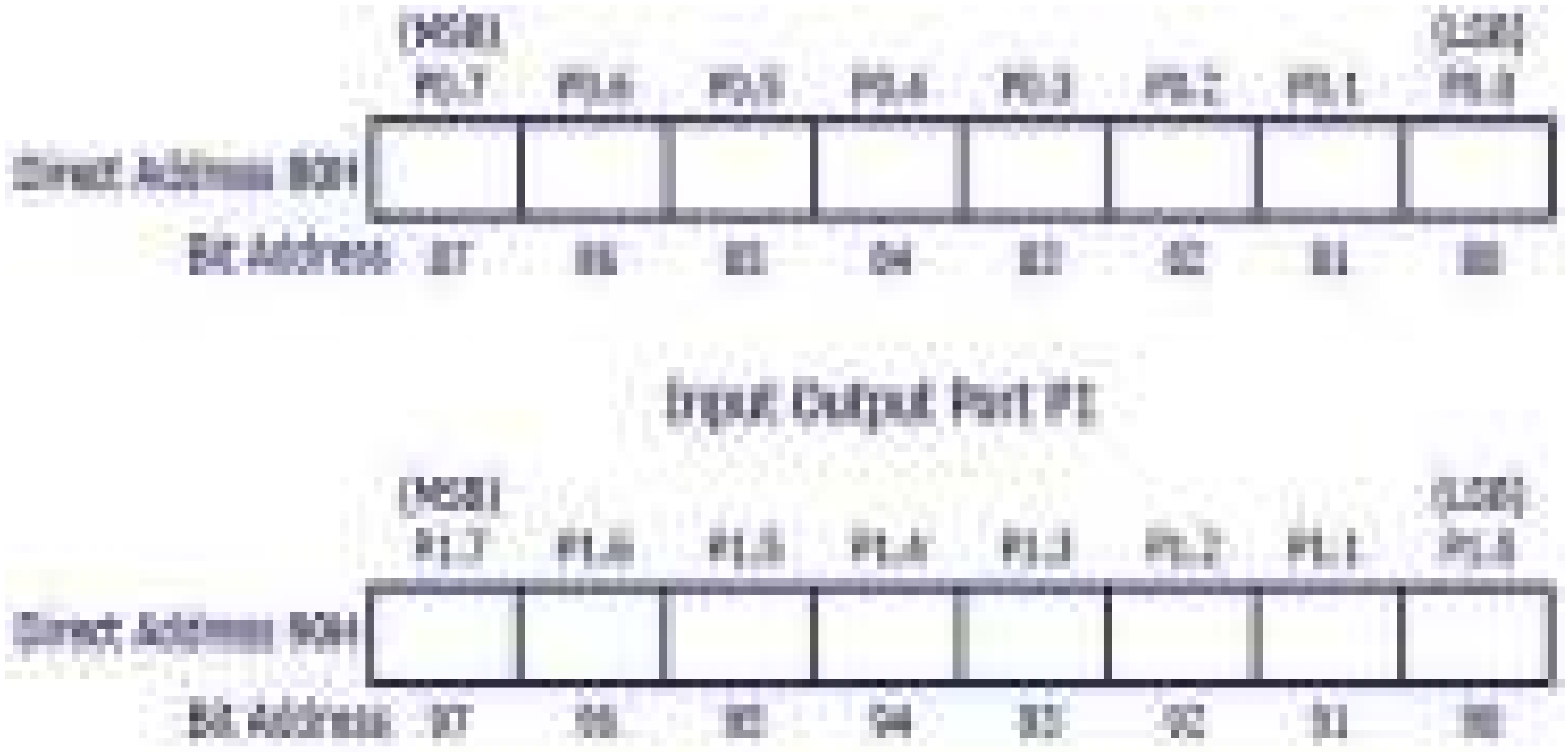
AC, the auxiliary carry flag

- If there is a carry from **D3 to D4 during an ADD or SUB** operation, this bit is set; otherwise, it is cleared.
- This flag is used by instructions that perform BCD (binary coded decimal) arithmetic.

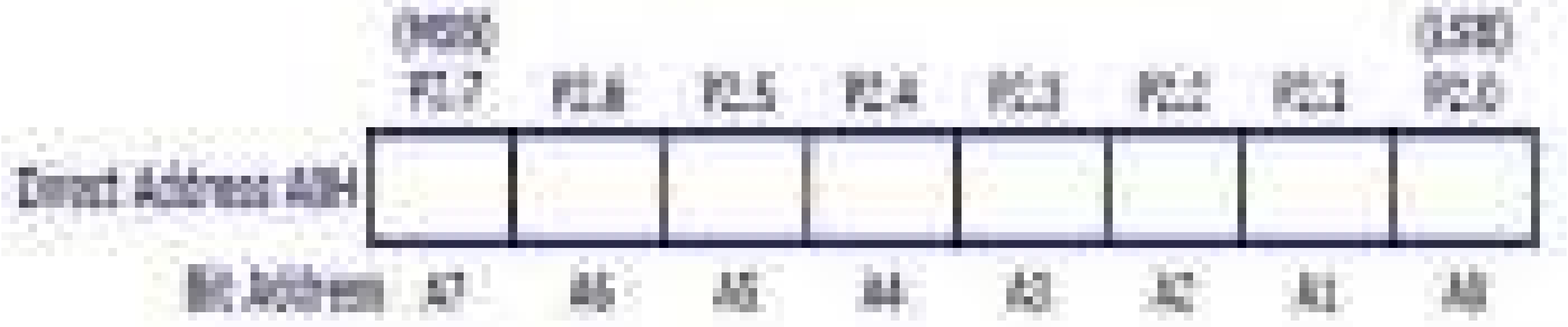
OV, the overflow flag

- This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
- In general, the carry flag is used to detect errors in unsigned arithmetic operations.
- The overflow flag is only used to detect errors in signed arithmetic operations

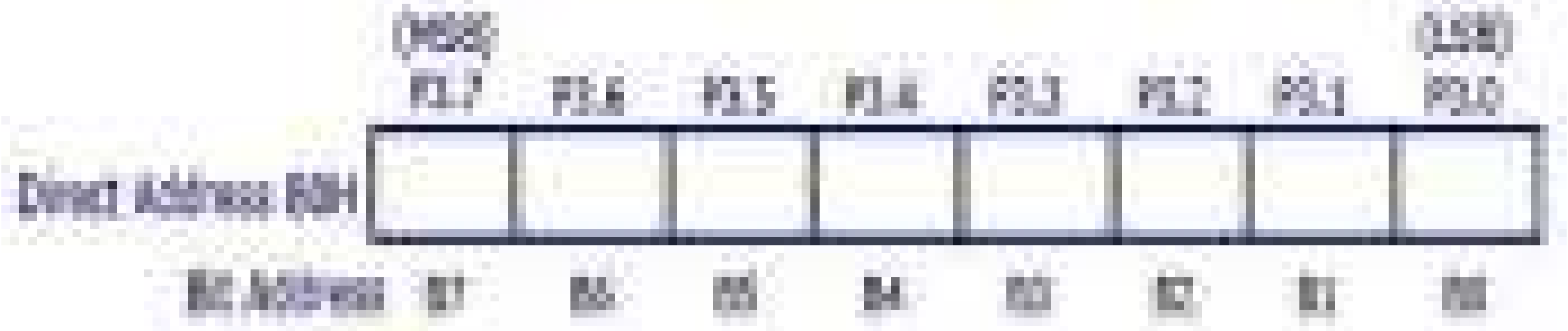
i/o ports


(MSB)
(LSB)
P0.7
P0.6
P0.5
P0.4
P0.3
P0.2
P0.1
P0.0
Direct Address 80H
Bit Address
87
86
85
84
83
82
81
80
Input Output Port P1
(MSB)
(LSB)
P1.7
P1.6
P1.5
P1.4
P1.3
P1.2
P1.1
P1.0
Direct Address 90H
Bit Address
97
96
95
94
93
92
91
90

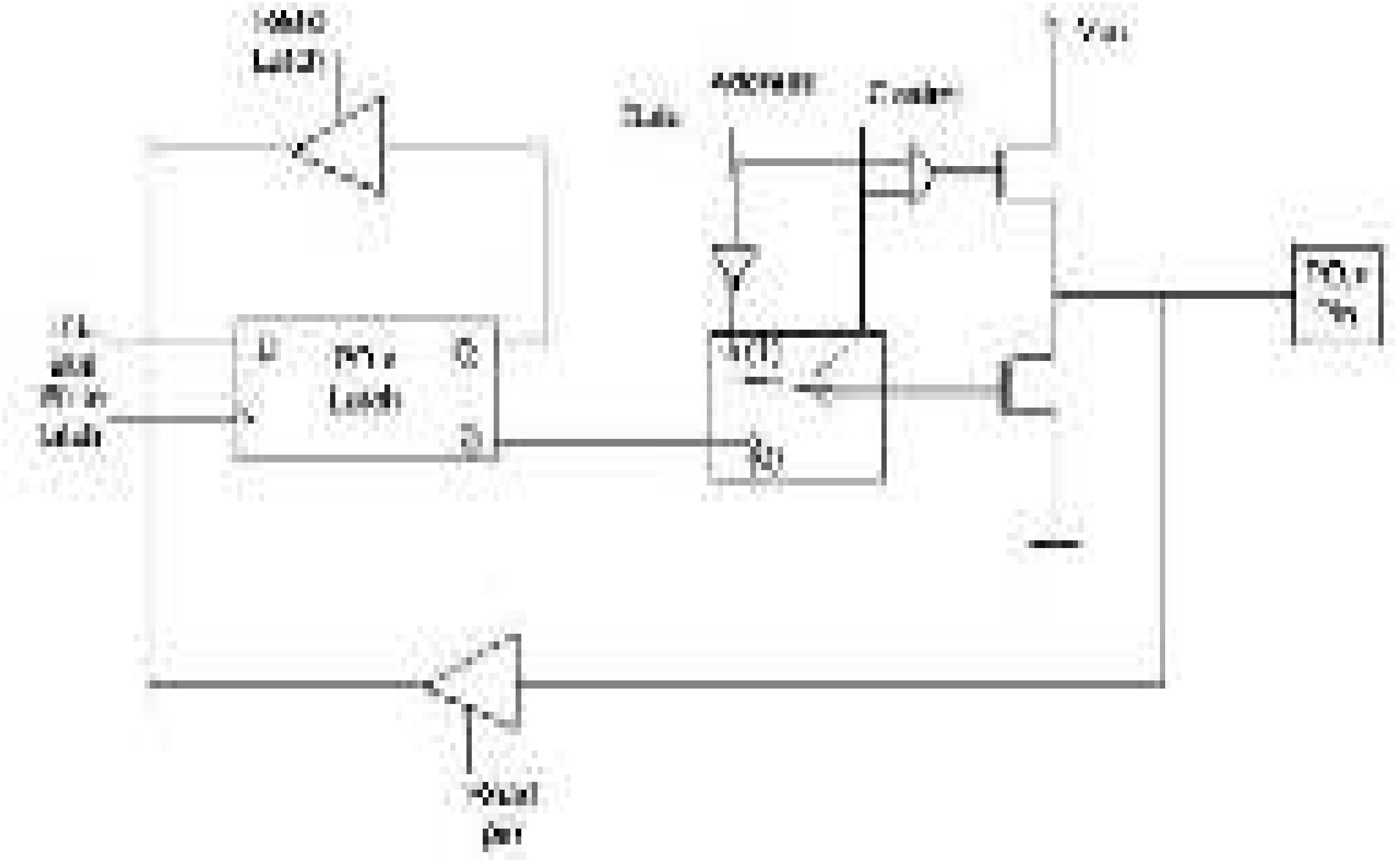
Input Output Port P2



Input Output Port P1



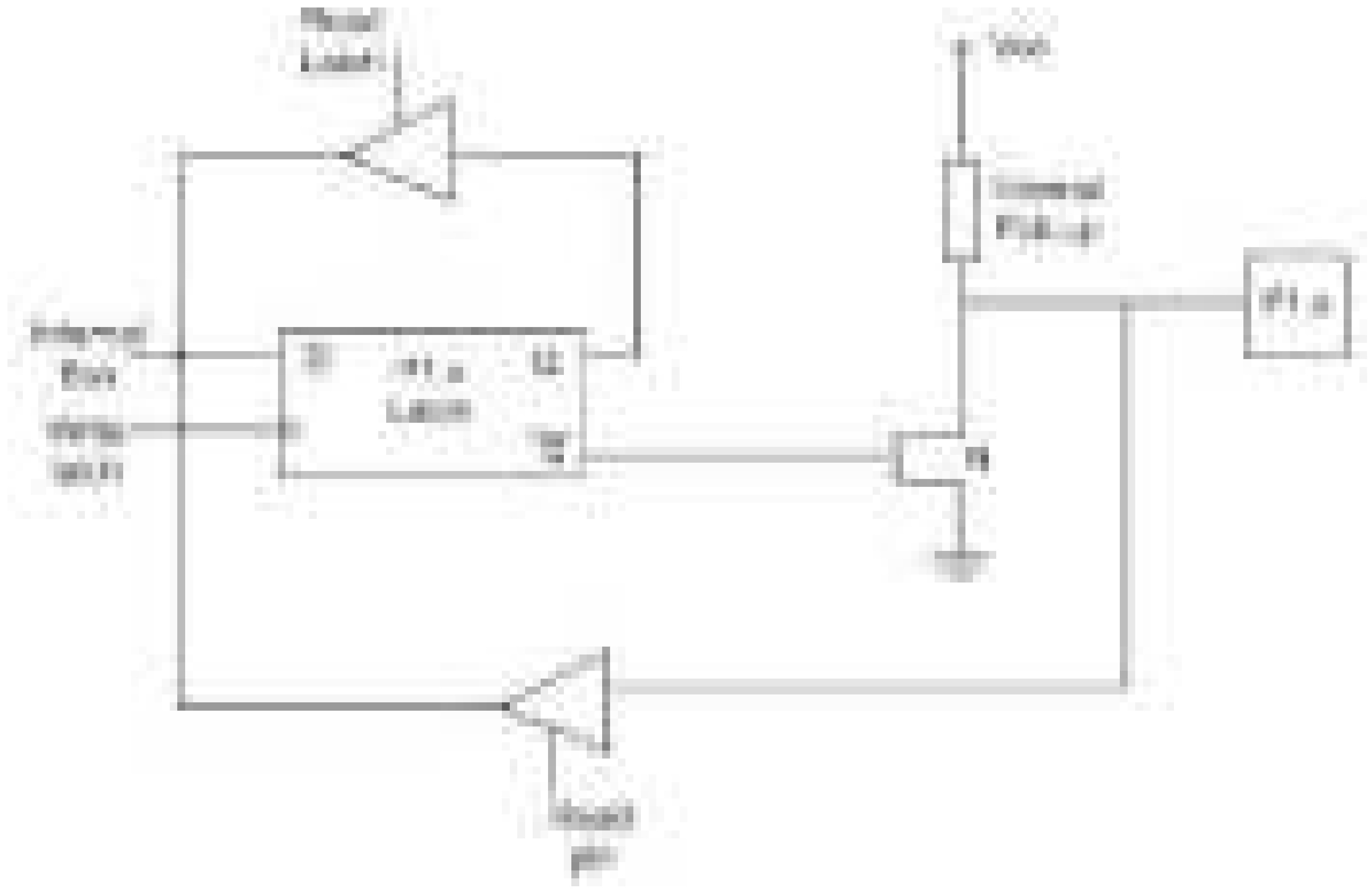
PORT 0 STRUCTURE



- Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory.
- When control is '1', the port is used for address/data interfacing.
- When the control is '0', the port can be used as a normal bidirectional I/O port.

- Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing.
- When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch.
- In this input mode when '1' is written to the pin by the external device then it read fine.
- But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up.
- If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

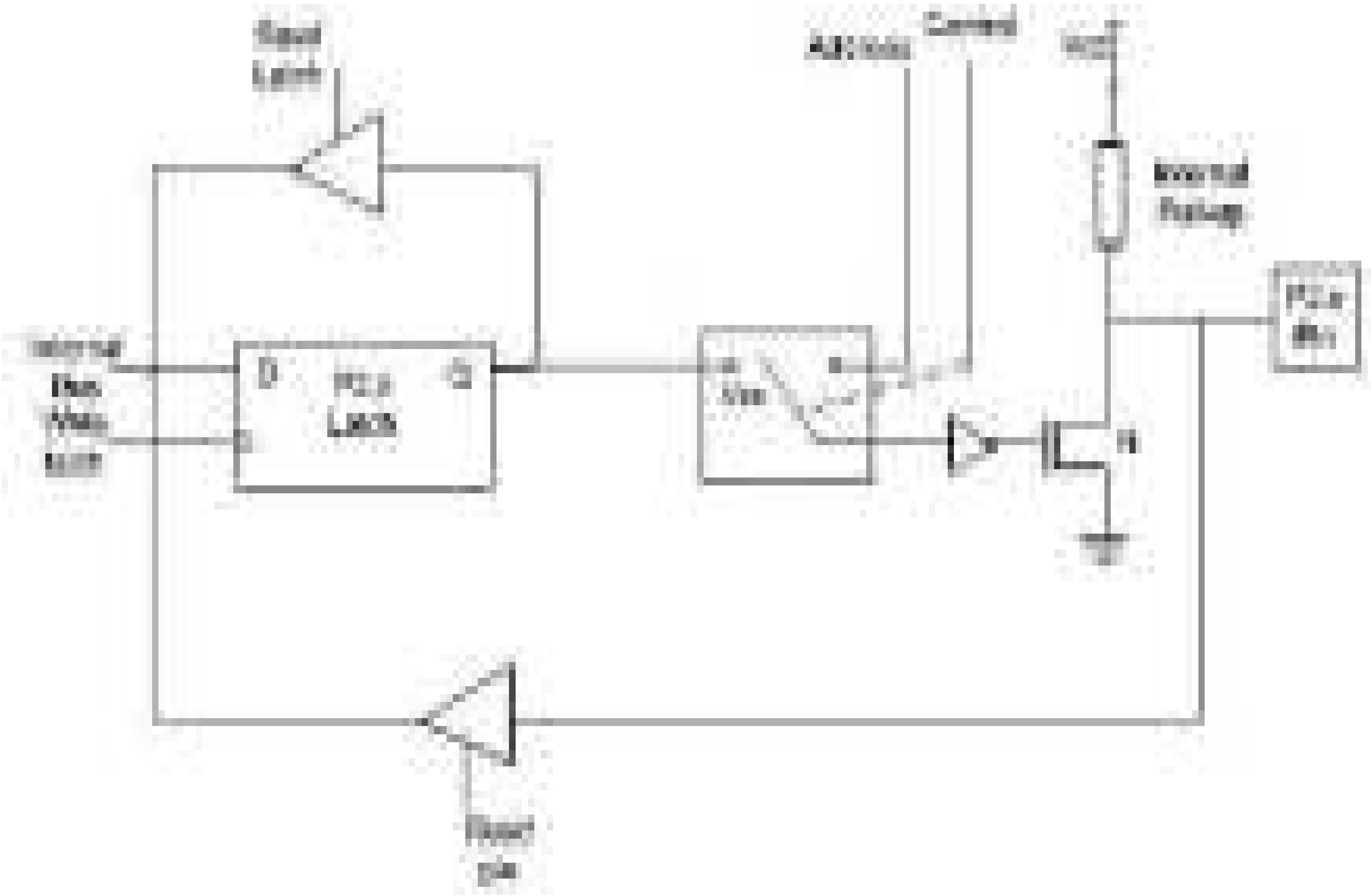
PORT 1 STRUCTURE



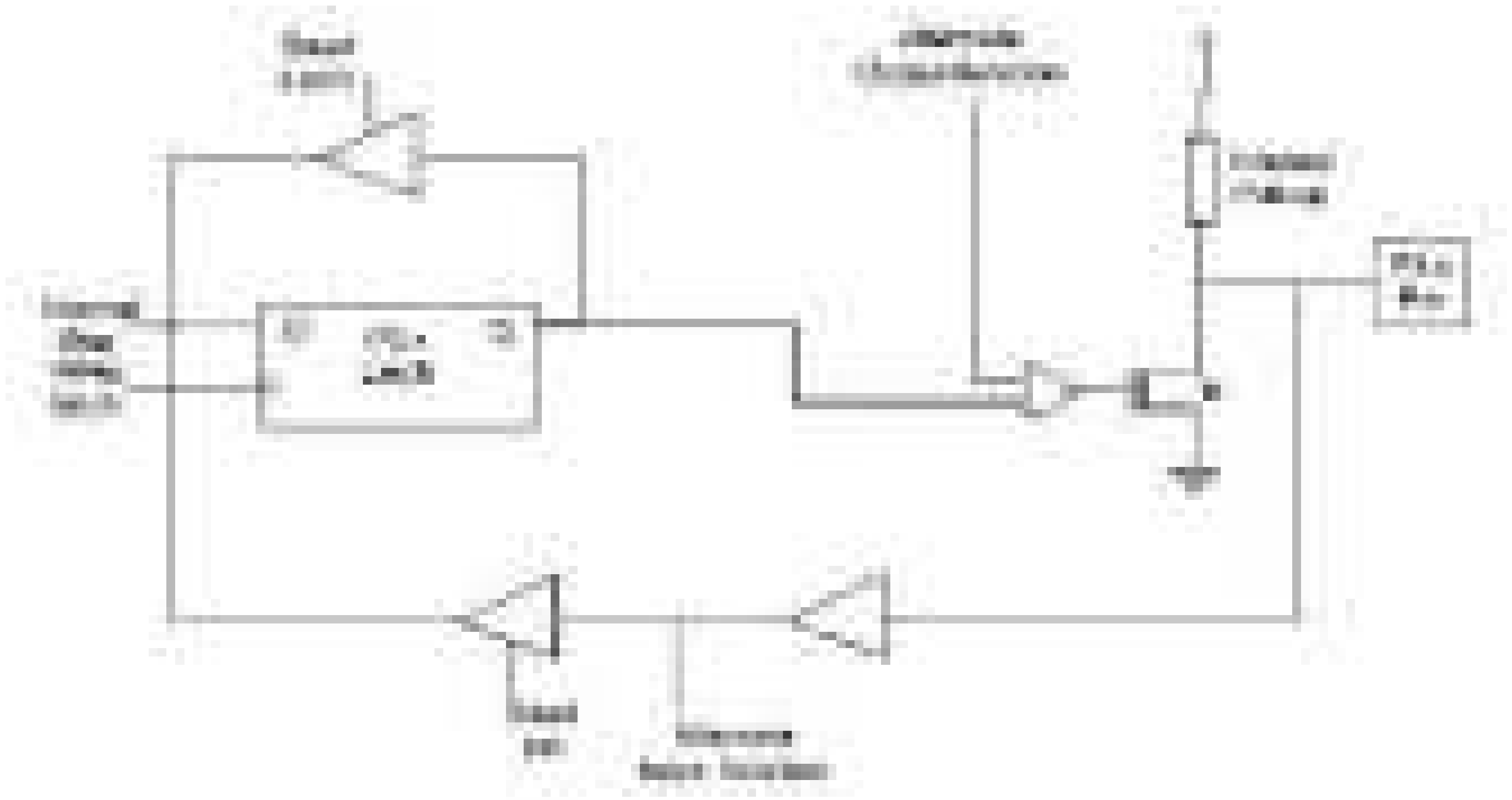
- Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1.
- Port-2 latch remains stable when Port-2 pin are used for external memory access.
- Here again due to internal pull-up there is limited current driving capability.

- Each pin of Port-3 can be individually programmed for I/O operation or for alternate function.
- The alternate function can be activated only if the corresponding latch has been written to '1'.
- To use the port as input port, '1' should be written to the latch.
- This port also has internal pull-up and limited current driving capability.

PORT 2 STRUCTURE



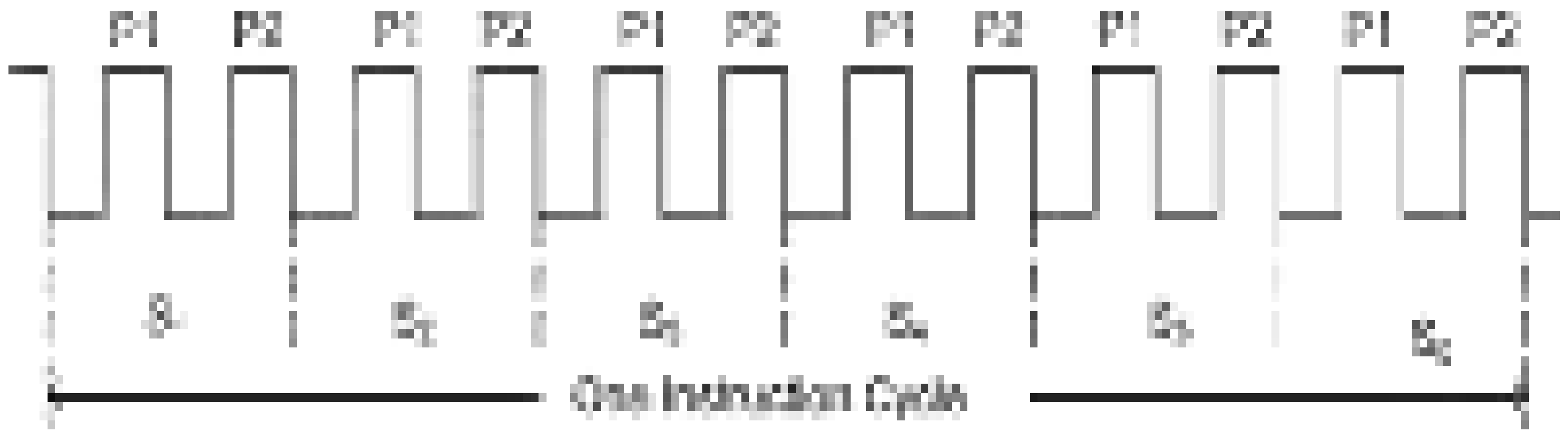
PORT 3 STRUCTURE



PORT 3 ALTERNATE FUNCTIONS :

P3 BIT	FUNCTION	PIN
P3.0	RXD	10
P3.1	TXD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

Clock cycle


P1
P2
P1
P2
P1
P2
P1
P2
P1
P2
P1
P2
S1
S2
S3
S4
S5
S6
One Instruction Cycle

- Machine cycle is the unit of time elapsed during the execution. It is the unit used to describe the operation of 8051operation.The CPU takes a certain number of clock cycles to execute an instruction. In the 8051 family, these time elapsed are referred as machine cycles. The length of the machine cycle depends on the frequency of the crystal oscillator connected to the 8051 system. Usually 12 clock cycles are referred as machine cycles
- Instruction cycle is the time required to complete the execution of an instruction. The Instruction may take different machine cycles to complete execution. So the instruction cycle may be of one or more machine cycles.

INTERRUPTS

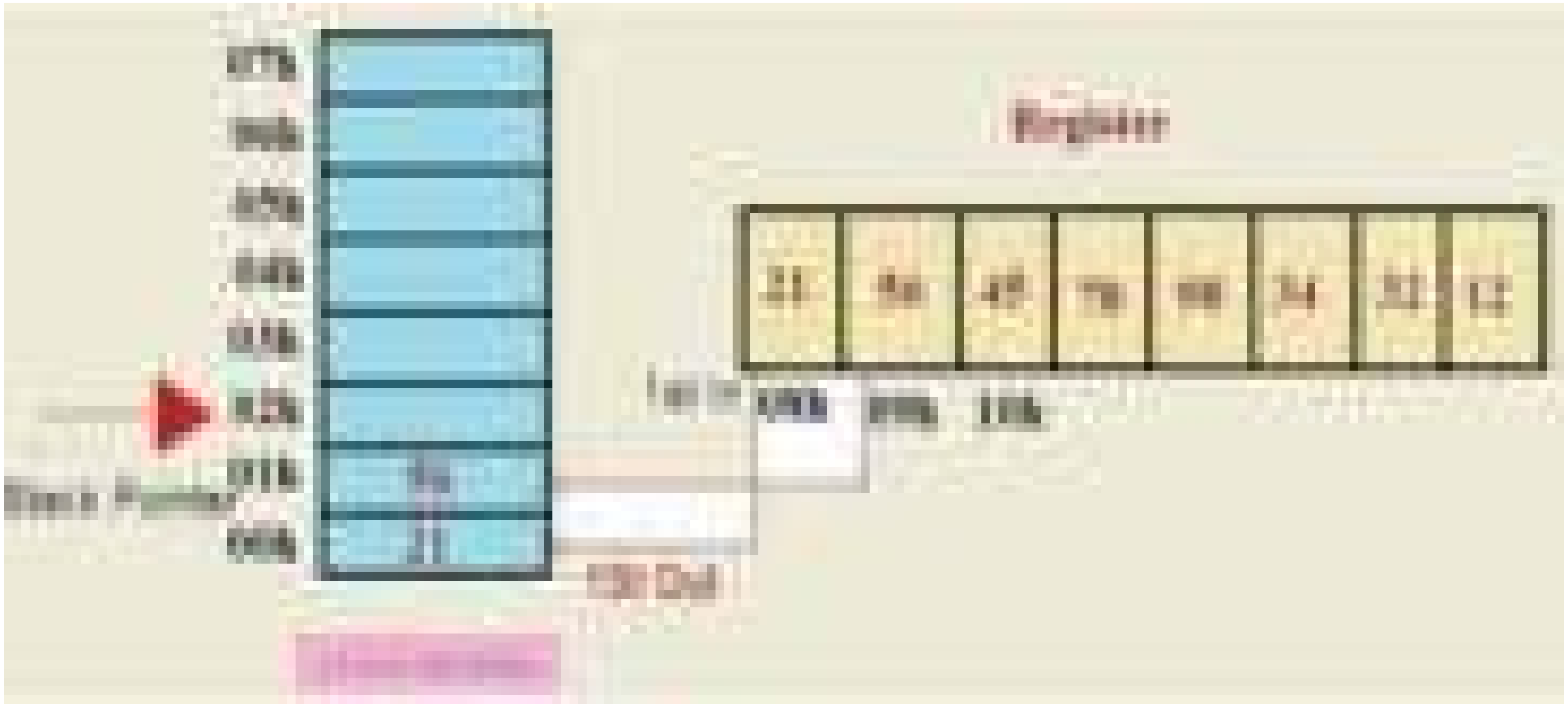
- The interrupts refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller momentarily stops and responds to the interrupt.
- Whenever an interrupt occurs the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or Interrupt Handler.
- **ISR** is a piece of code that tells the processor or controller what to do when the interrupt occurs.
- After the execution of ISR, controller returns back to the instruction it has jumped from (before the interrupt was received).
- A SYSTEM RESET is a special type of interrupt.
- It interrupts the running program and loads the PC with the vector address 0000H. This is the address the microcontroller begins with on power-up. Therefore, reset is similar to powering down and the powering up the system.

Interrupt Vectors : When an interrupt occurs the address of the interrupt service routine is loaded into the PC. This address is known as the interrupt vector.

Interrupt	Flag	Vector Address
External interrupt 0	IE0	0003H
Timer 0	TF0	000BH
External interrupt 1	IE1	0013H
Timer 1	TF1	001BH
Serial port	RI or TI	0023H

STACK

- It is a part of RAM in which data will be stored temporary during execution of program. STACK works on last in first out principle.
- To store and retrieve data during program execution in stack, push and pop instruction are used .
- PUSH is used to store data into stack. POP is used to retrieve data from stack.



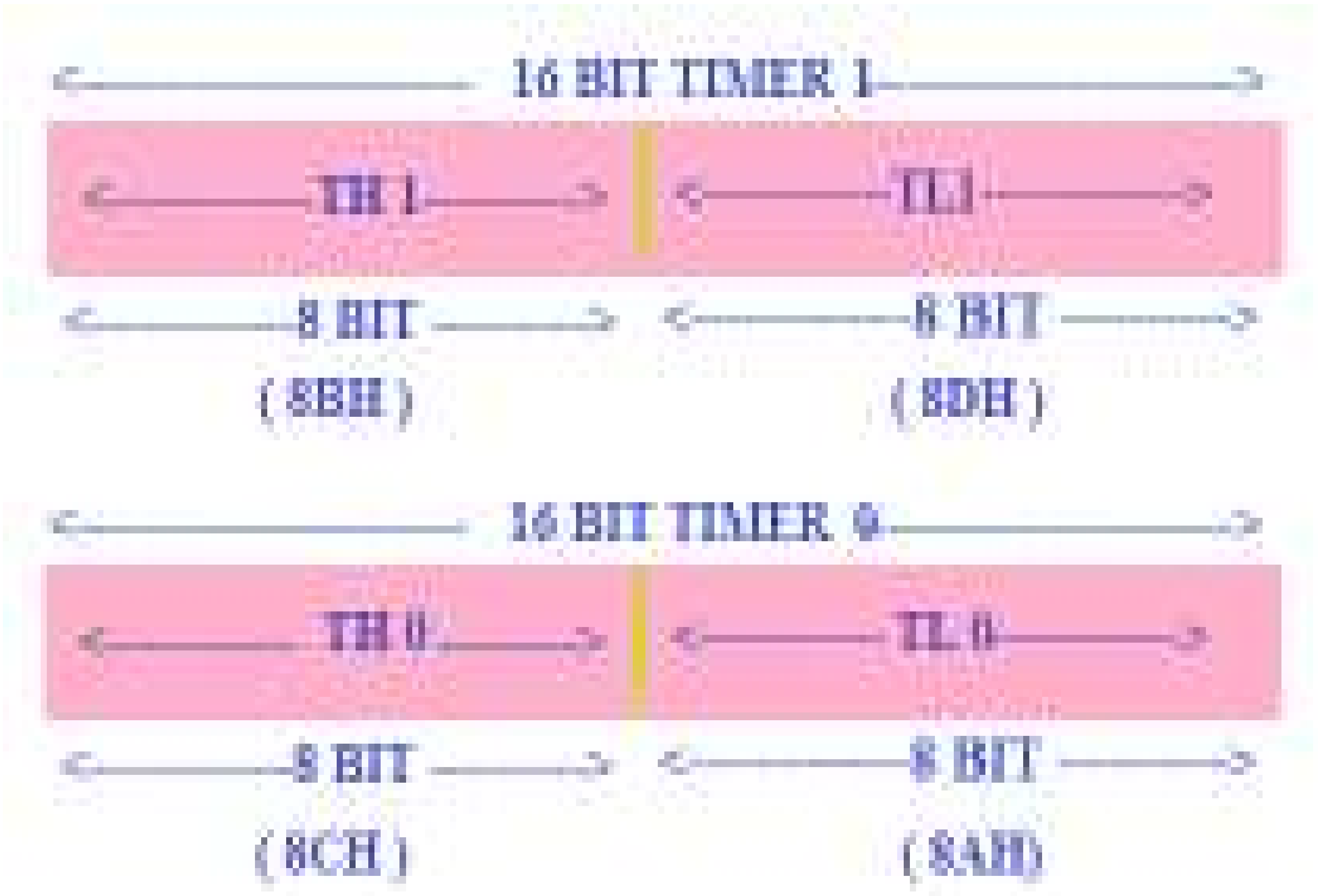
- During the execution of CALL instructions, the microcontroller stores the content of program counter in the stack memory.
- During the RET instructions , the content of stack is moved in to program counter.
- Stack Pointer is the register(SFR) used to point the address of stack.
- The address of stack memory is placed in stack pointer.
- The stack pointer in the 8051 is only 8 bits wide, which means that it can take value 00 to FFH.
- During PUSH operation ,the Stack Pointer has the next address of stack The stack pointer is incremented during PUSH.
- During POP operation ,the Stack Pointer has the address of stack from which data is accessed next.
- The Stack Pointer is used to indicate where the next value to be removed from the stack. The stack pointer is decremented during POP.

TIMER

- 8051 has two on-chip timers that can be used for counting timing durations or for counting external events.
- Interval timing allows the programmer to perform operations at specific instants in time.
- For example, in our LED flashing program the LED was turned on for a specific length of time and then turned off for a specific length of time.
- We achieved this through the use of time delays.
- Since the microcontroller operates at a specific frequency, we could work out exactly how many iterations of the time delay was needed to give us the desired delay.
- However, this is difficult and imperfect. And there is another disadvantage is that CPU is occupied for the maintaining the delay.
- If we use the on-chip timers, the CPU could be doing something more useful work.
- The timers take care on timer work.

Timers' SFRs

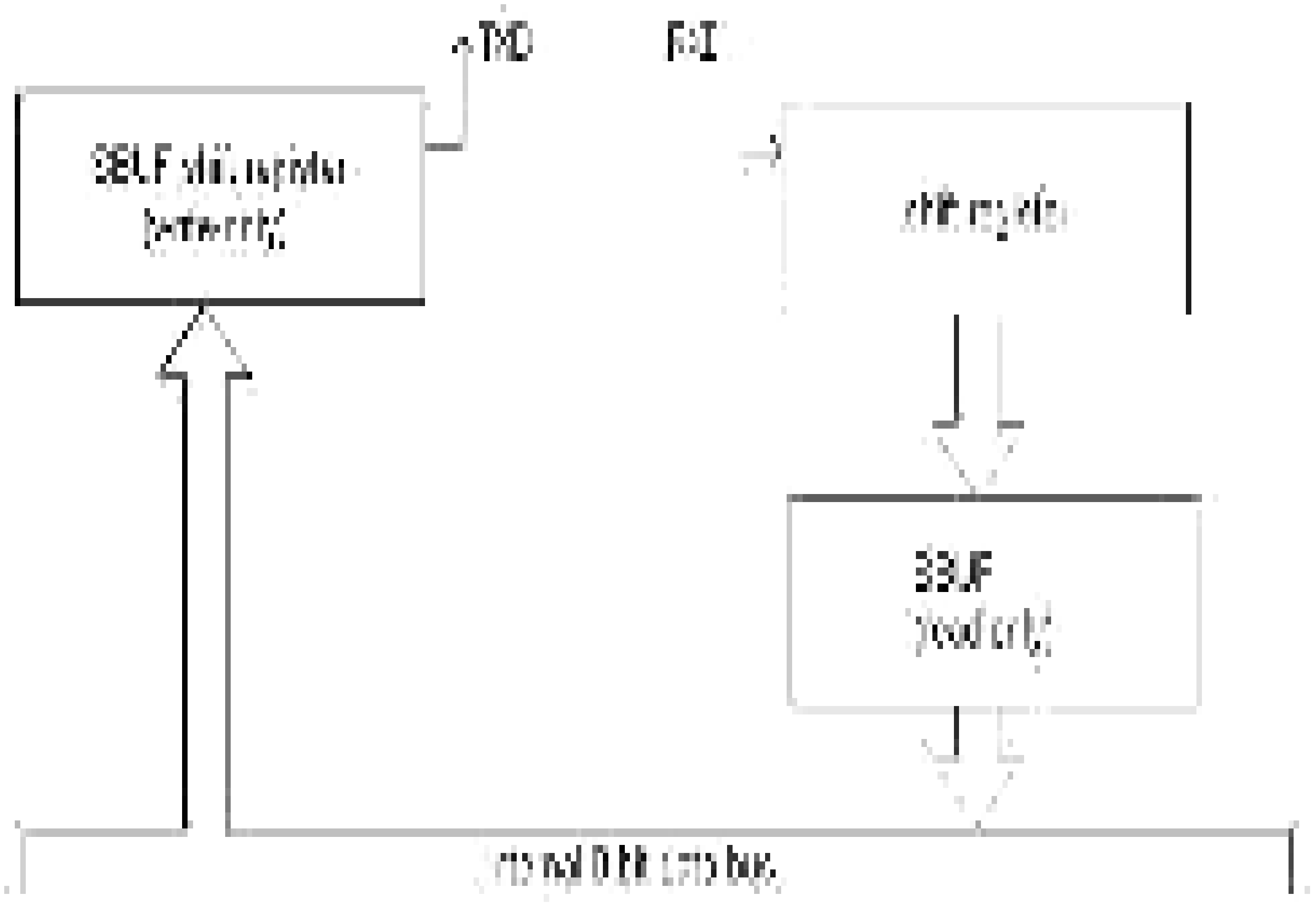
- The 8051 has two 16-bit timers.
    - ❖ The high byte for timer 1 (TH1) is at address 8DH while the low byte (TL1) is at address 8BH.
    - ❖ The high byte for timer 0 (TH0) is at address 8CH while the low byte (TL0) is at address 8AH.
- Both timers can be used in a four number of different modes.
- The programmer sets the timers to a specific mode by loading the appropriate 8-bit number into the Timer Mode Register (TMOD) .


16 BIT TIMER 1
TH 1
TL1
8 BIT
8 BIT
(8BH)
(8DH)
16 BIT TIMER 0
TH 0
TL 0
8 BIT
8 BIT
(8CH)
(8AH)

SERIAL PORT

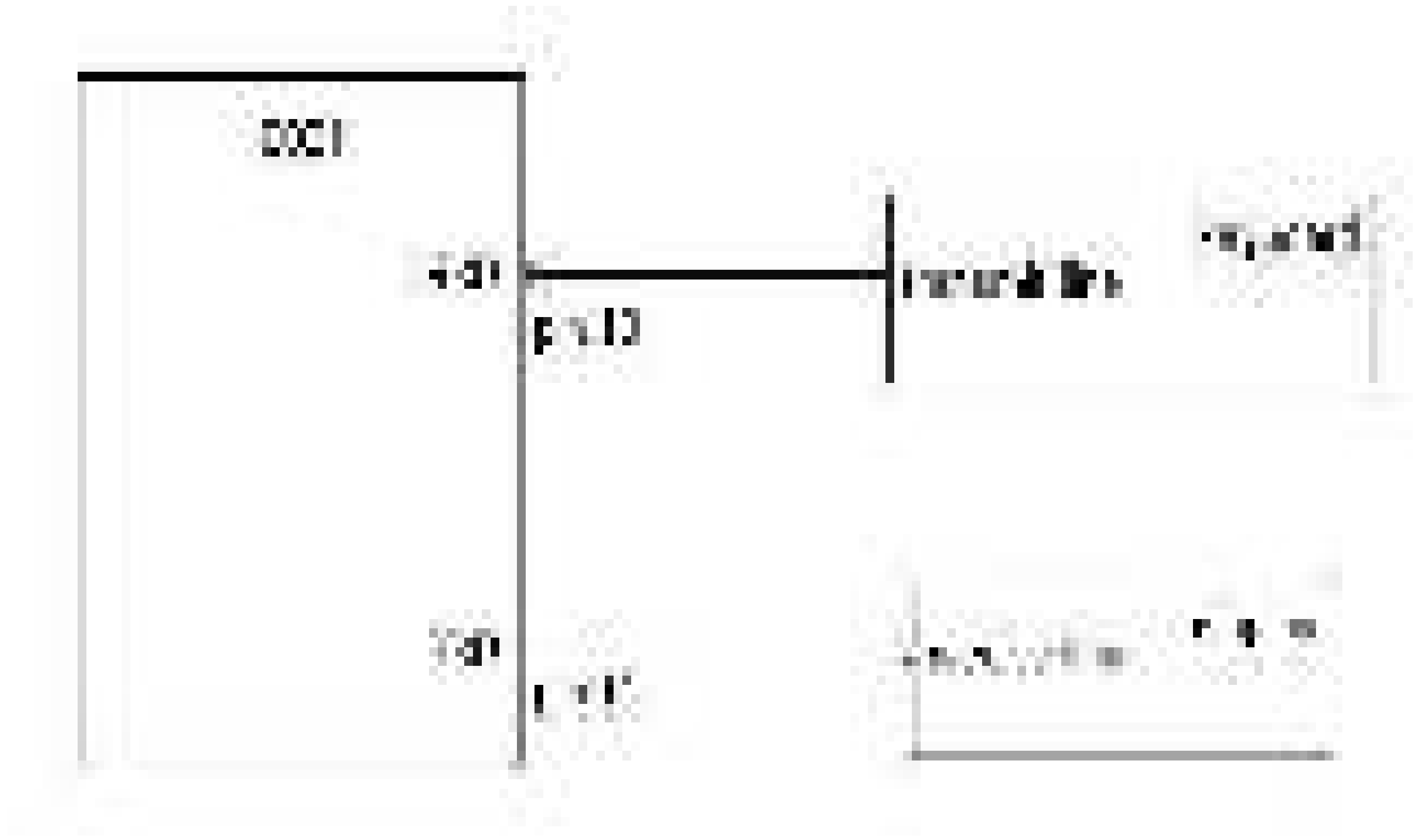
- The 8051 includes an on-chip serial port that can be programmed to operate in one of four different modes and at a range of frequencies.
- In serial communication the data is rate is known as the baud rate.
- The baud rate simply means the number of bits transmitted per second.
- In the serial port modes that allow variable baud rates, this baud rate is set by timer 1.

Diagram of Serial port (FIG A)



- The 8051 serial port is full duplex.
- In other words, it can transmit and receive data at the same time. Shown in the Fig B Diagram of Serial port.
- Location 99H the serial buffer special function register (SBUF).
- SBUF is in fact two distinct registers - the write-only register and the read-only register.
- Transmitted data is sent out from the write-only register while received data is stored in the read-only register.
- There are two separate data lines, one for transmission (TXD) and one for reception (RXD).
- Therefore, the serial port can be transmitting data down the TXD line while it is at the same time receiving data on the RXD line.

Diagram of Serial Communication (FIG B)



- The TXD line is pin 11 of the microcontroller (P3.1) while the RXD line is on pin 10 (P3.0).
- Therefore, external access to the serial port is achieved by connecting to these pins.
- For example, if you wanted to connect a keyboard to the serial port you would connect the transmit line of the keyboard to pin 10 of the 8051.
- If you wanted to connect a display to the serial port you would connect the receive line of the display to pin 11 of the 8051.
- This is detailed in the Fig B Diagram of Serial Communication diagram below.

- **Transmitting and Receiving Data**
- Essentially, the job of the serial port is to change parallel data into serial data for transmission and to change received serial data into parallel data for use within the microcontroller.
  - ❖ Serial transmission is changing parallel data to serial data.
  - ❖ Serial reception is changing serial data into parallel data.
  - ❖ Both are achieved through the use of shift registers.

Thank You

INTEGRATED TECHNICAL EDUCATION CLUSTER
AT ALAMEERIA

E-626-A
Real-Time Embedded Systems (RTES)

Lecture #3
PIC Microcontrollers

Instructor:
Dr. Ahmad El-Banna

SPRING 2015

Agenda

- What's a Microcontroller?
- Types of Microcontrollers
- Features and Internal structure of PIC 16F877A
- Instruction Execution

What is a microcontroller?

- A **microcontroller** (sometimes abbreviated **μC, uC** or **MCU**) is a small computer on a single **integrated circuit** containing a **processor core**, **memory**, and programmable **input/output** peripherals.
- It can only perform simple/specific tasks.
- A microcontroller is often described as a '**computer-on-a-chip**'.

Microcomputer system and Microcontroller based system

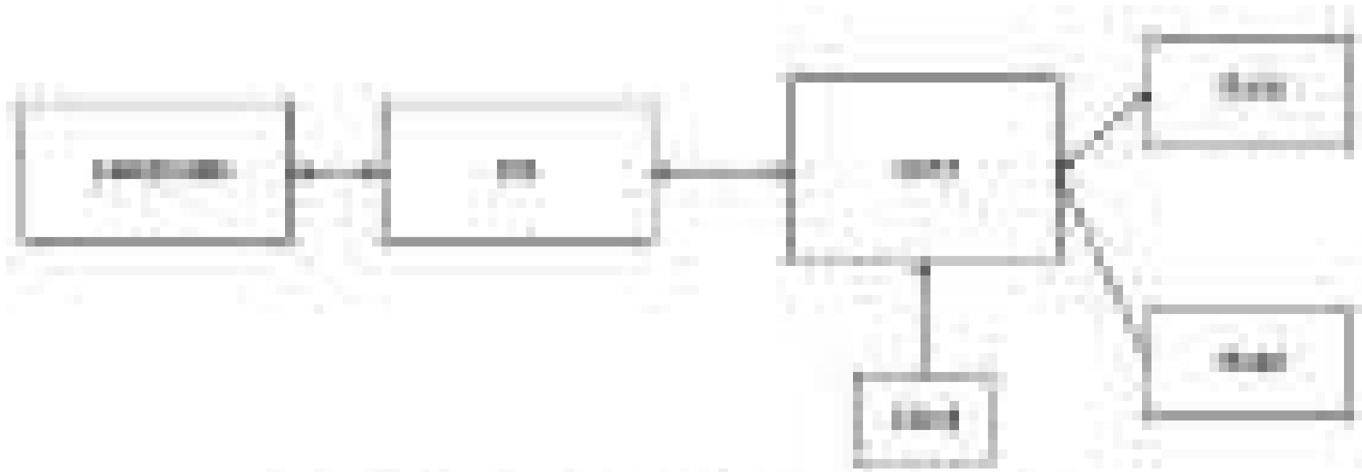


Figure 1: Basic building blocks of a computer

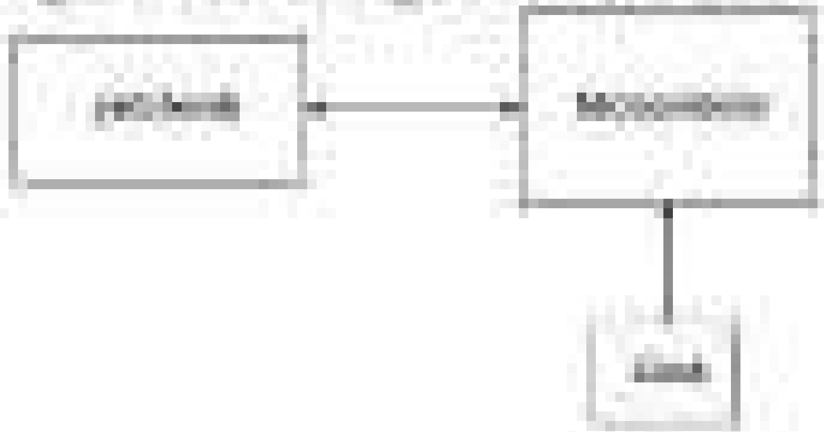
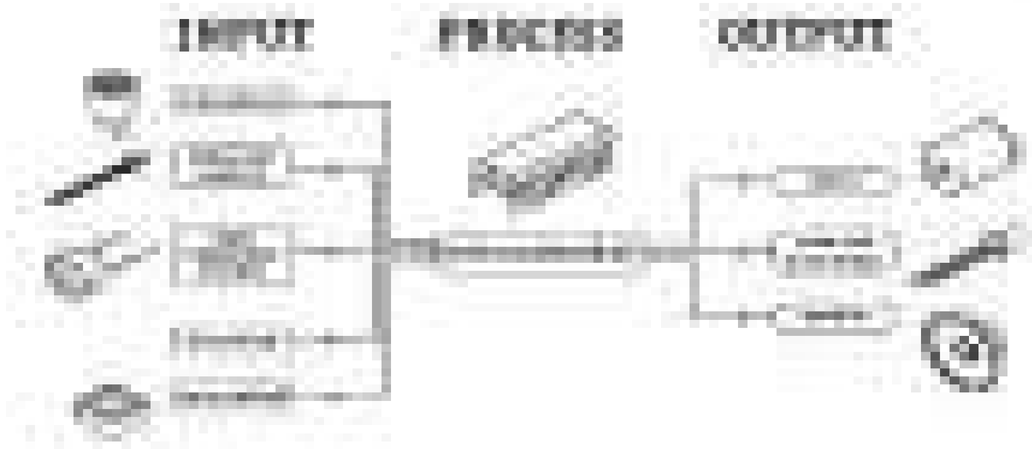


Figure 2: A microcontroller based system

Microcontrollers..

- Microcontrollers are purchased ‘blank’ and then programmed with a specific control program.
- Once programmed the microcontroller is build into a product to make the product more intelligent and easier to use.
- A designer will use a Microcontroller to:
  - Gather input from various sensors
  - Process this input into a set of actions
  - Use the output mechanisms on the microcontroller to do something useful.



Types of Microcontrollers

- **Parallax Propeller**
- **Freescale 68HC11** (8-bit)
- **Intel 8051**
- **Silicon Laboratories** Pipelined 8051 Microcontrollers
- **ARM** processors (from many vendors) using **ARM7** or Cortex-M3 cores are generally microcontrollers
- **STMicroelectronics STM8** (8-bit), **ST10** (16-bit) and **STM32** (32-bit)
- **Atmel AVR** (8-bit), **AVR32** (32-bit), and **AT91SAM** (32-bit)
- **Freescale ColdFire** (32-bit) and **S08** (8-bit)
- **Hitachi H8**, **Hitachi SuperH** (32-bit)
- **Hyperstone** E1/E2 (32-bit, First full integration of RISC and DSP on one processor core [1996])
- **Infineon Microcontroller**: 8, 16, 32 Bit microcontrollers for automotive and industrial applications.

Types of Microcontrollers..

- **MIPS** (32-bit PIC32)
- **NEC V850** (32-bit)
- **Microchip PIC** **(8-bit PIC16, PIC18, 16-bit dsPIC33/PIC24)**
- **PowerPC** ISE
- **PSoC** (Programmable System-on-Chip)
- **Rabbit 2000** (8-bit)
- **Texas Instruments Microcontroller MSP 430** (16-bit), C2000 (32-bit), and Stellaris (32-bit)
- **Toshiba TLCS-870** (8-bit/16-bit)
- **Zilog eZ8** (16-bit), **eZ80** (8-bit)
- etc

Microcontroller Packaging and Appearance

From left to right: PIC 12F508, PIC 16F84A, PIC 16C72, Motorola 68HC05B16, PIC 16F877, Motorola 68000

PIC Microcontrollers

- Peripheral Interface Controller (PIC) was originally designed by General Instruments
- In the late 1970s, GI introduced PIC® 1650 and 1655 – RISC with 30 instructions.
- PIC was sold to Microchip
- Features: low-cost, self-contained, 8-bit, Harvard structure, pipelined, RISC, single accumulator, with fixed reset and interrupt vectors.

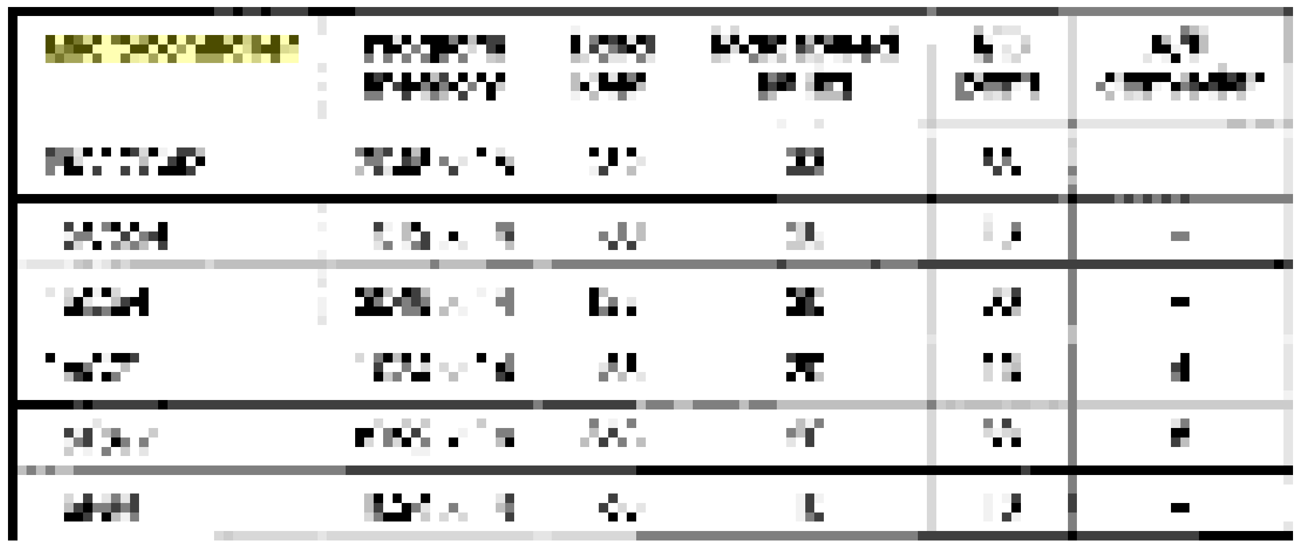
PIC Microcontroller product family

- 8-bit microcontrollers
  - PIC10
  - PIC12
  - PIC14
  - PIC16
  - PIC17
  - PIC18
- 16-bit microcontrollers
  - PIC24F
  - PIC24H
- 32-bit microcontrollers
  - PIC32
- 16-bit digital signal controllers
  - dsPIC30
  - dsPIC33F

PIC Microcontroller product family..

- The **F** in a name generally indicates the PICmicro uses flash memory and can be erased electronically.
- The **C** generally means it can only be erased by exposing the die to ultraviolet light (which is only possible if a windowed package style is used). An exception to this rule is the PIC16C84 which uses EEPROM and is therefore electrically erasable.

[illegible]



[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-

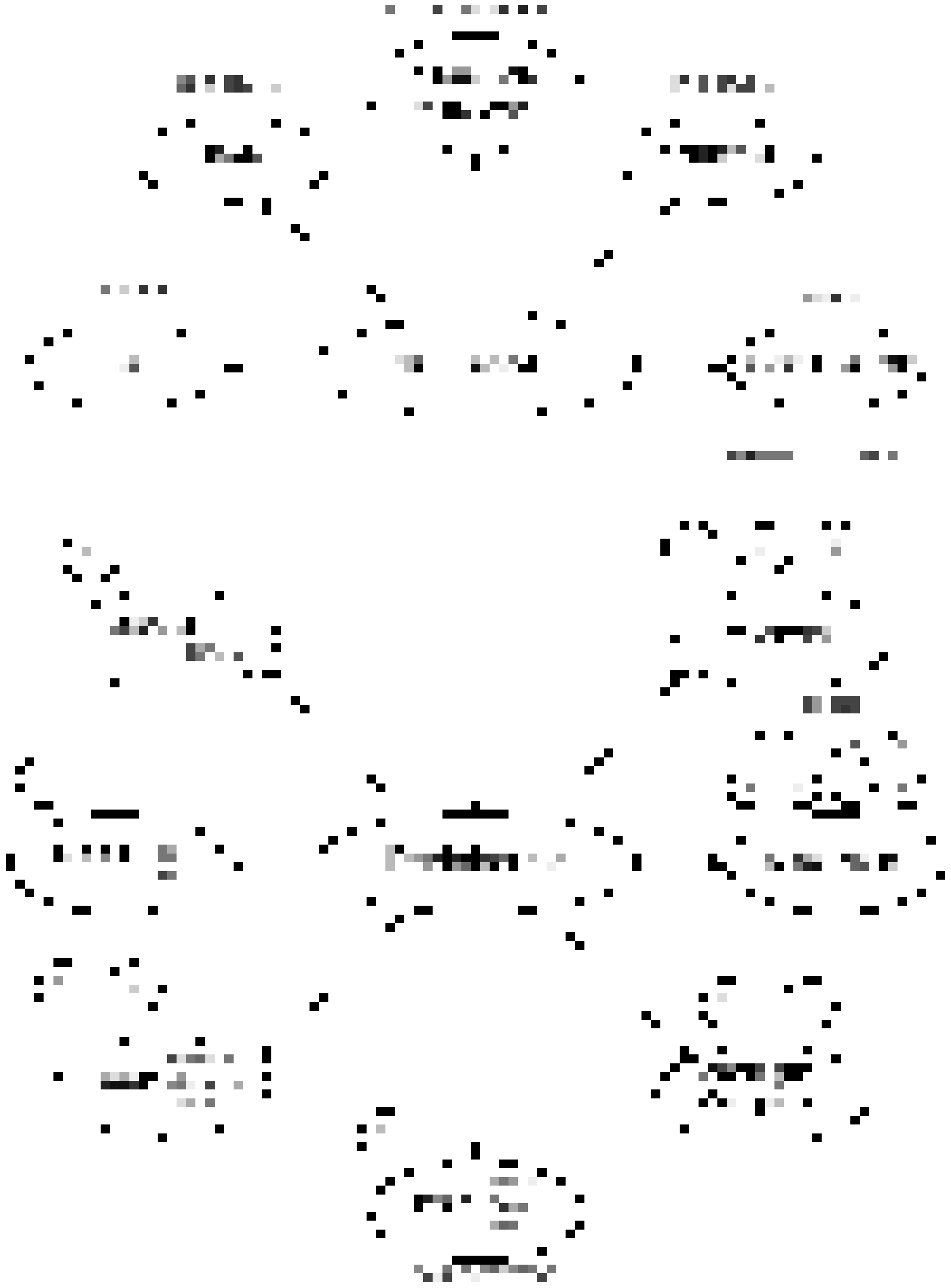
An Example: PIC16F877

- **Why PIC16F877A is very popular?**
- This is because PIC16F877A is very cheap. Apart from that it is also very easy to be assembled. Additional components that you need to make this IC work is just a 5V power supply adapter, a 20MHz crystal oscillator and 2 units of 22pF capacitors.
- **What is the advantages of PIC16F877A?**
- This IC can be reprogrammed and erased up to 10,000 times. Therefore it is very good for new product development phase.
- **What is the disadvantages of PIC16F877A?**
- This IC has no internal oscillator so you will need an external crystal of other clock source.

Features

Key Features	PIC16F877
MAX Operating Frequency	20MHz
FLASH Program Memory (14-bit words)	8K
Data Memory (bytes)	368
EEPROM Data Memory (bytes)	256
I/O Ports	RA0-5 (6) RB0-7 (8) RC0-7 (8) RD0-7 (8) RE0-2 (3)
Timers	3
CCP (Capture/Compare/PWM)	2
Serial Communications	MSSP, USART
Parallel Communications	PSP
10-bit Analog-to-Digital Module	8 Channels
Instruction Set	35 Instructions
Pins (DIP)	40 Pins

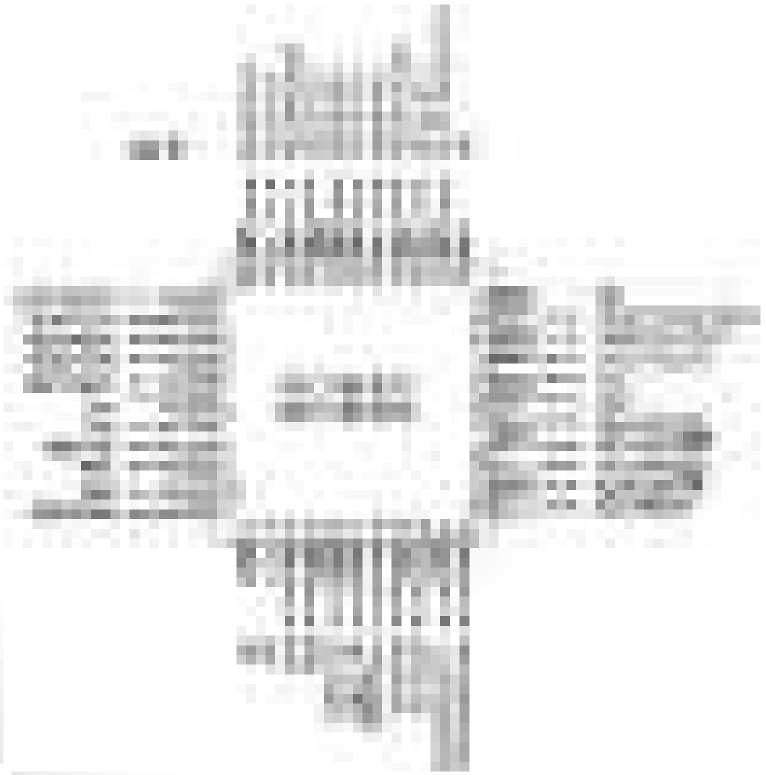
Bubble diagram of PIC16F877

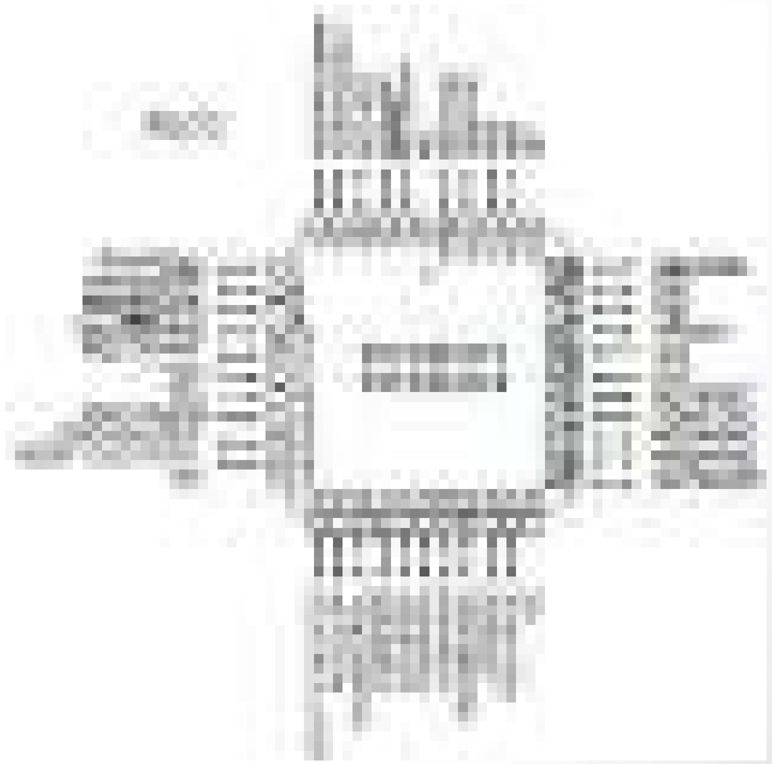


Pin Diagram of PIC16F877

- **Quad Flat Package (QFP)**

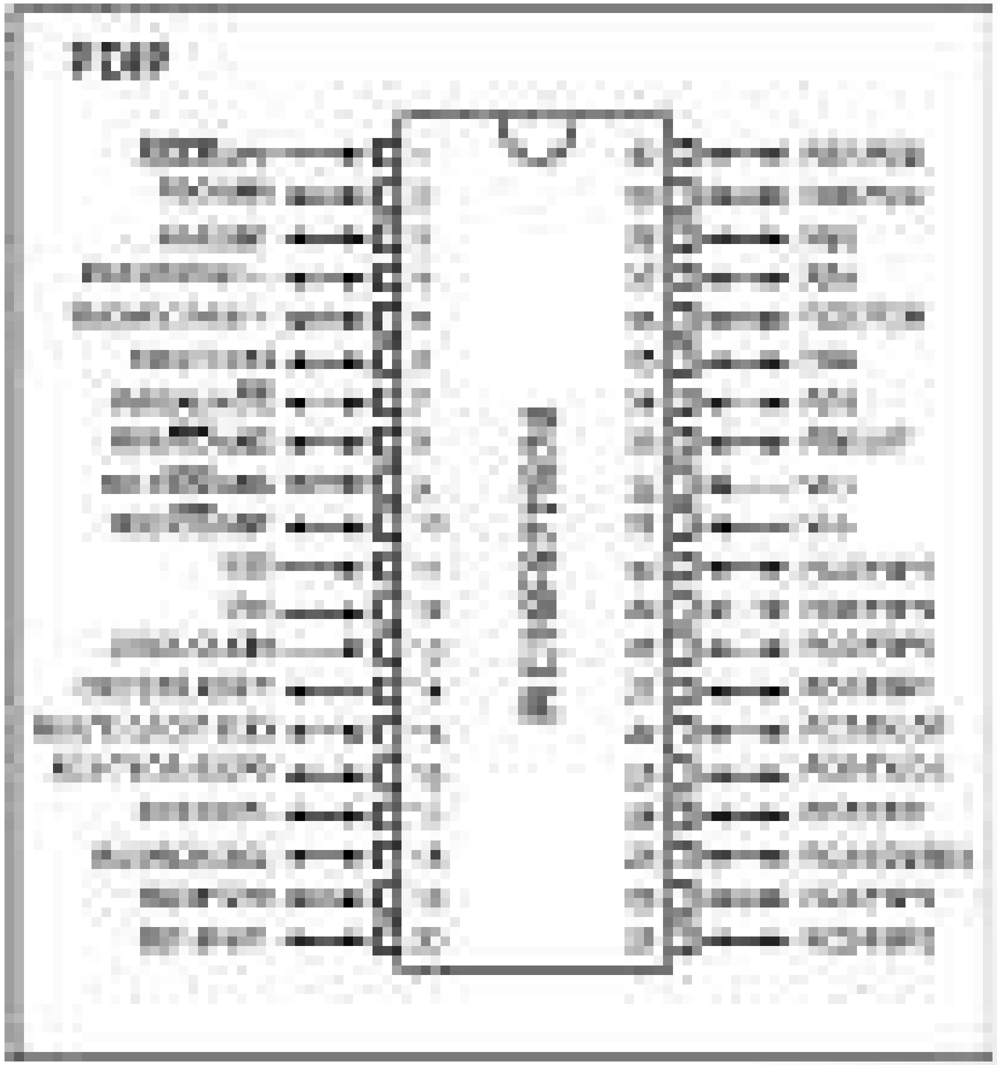
- **Plastic Leaded Chip Carrier Package (PLCC)**



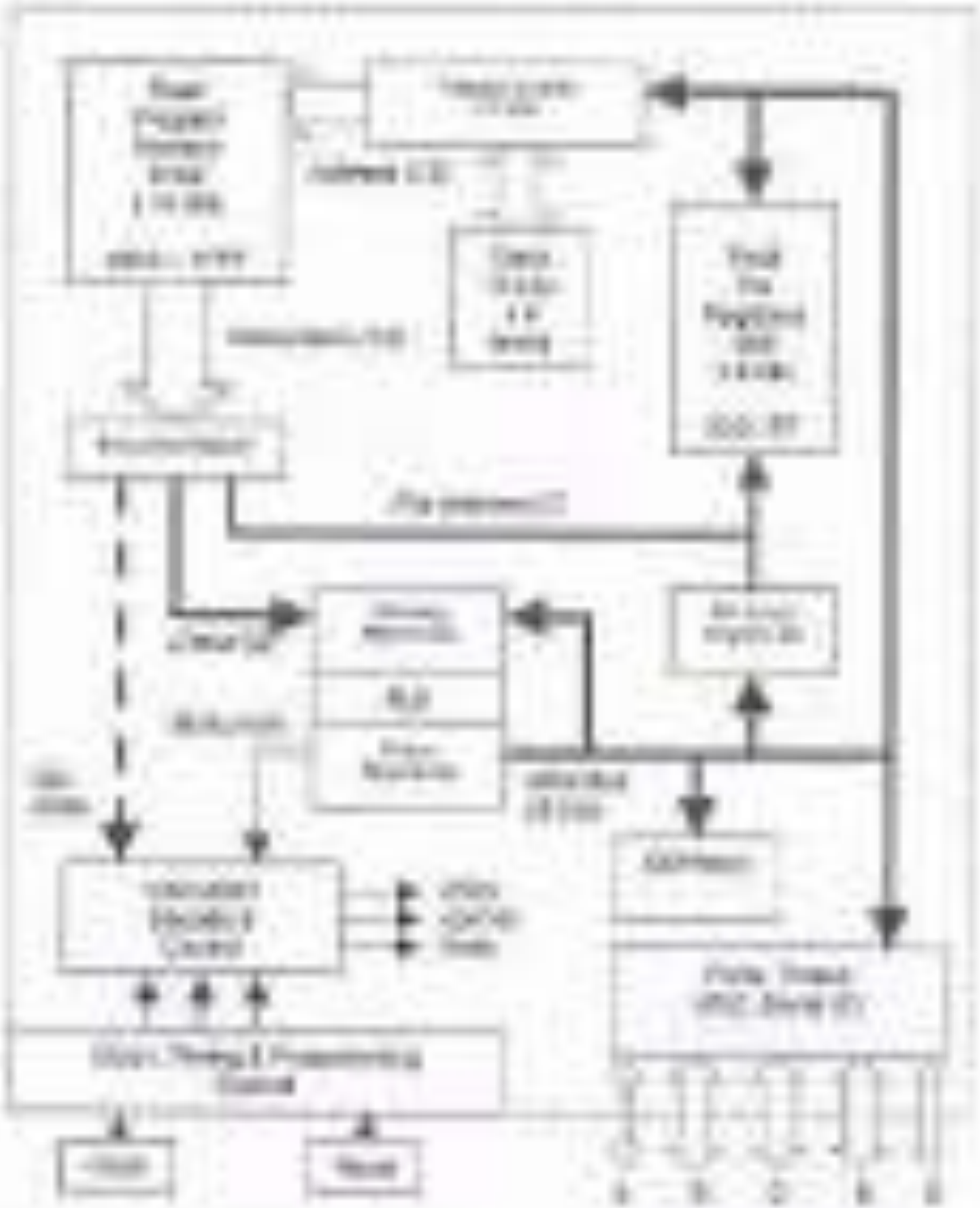


Pin Diagram of PIC16F877..

- **Plastic dual in-line package (DIP)**

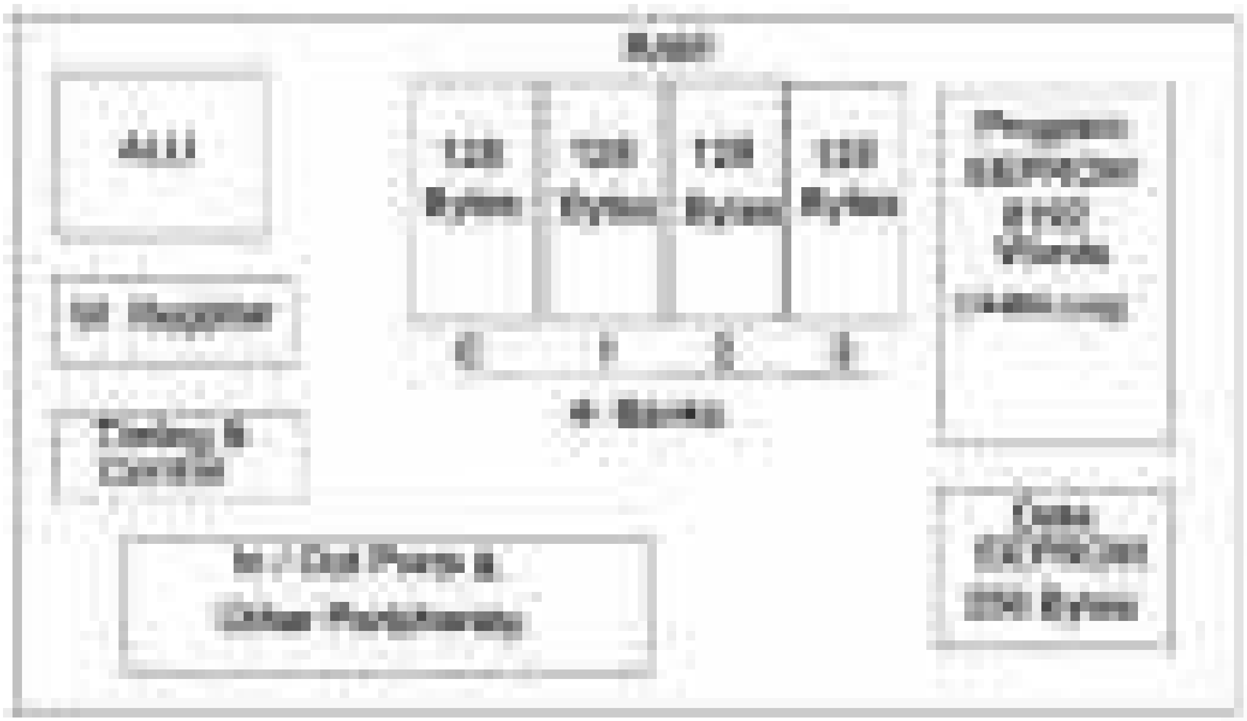


PIC16F877 Architecture



PIC16F877 Internal Block Diagram

- The basic architecture of PIC16F877 consists of Program memory, file registers and RAM, ALU and CPU registers.



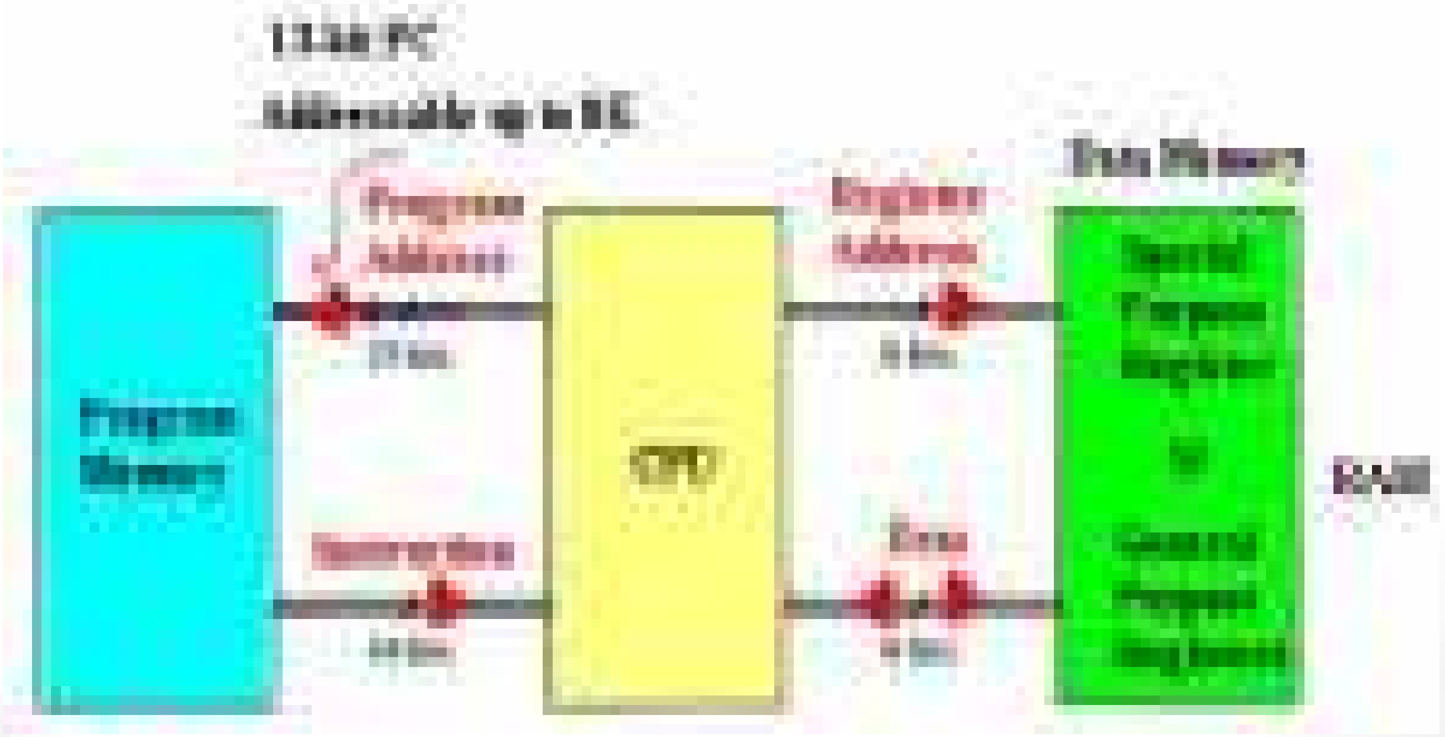
Memory of the PIC16F877

- divided into 3 types of memories:

  1. **Program Memory** – A memory that contains the program (which we had written), after we've burned it. As a reminder, Program Counter executes commands stored in the program memory, one after the other.
  2. **Data Memory** – This is RAM memory type, which contains a special registers like **SFR** (Special Function Register) and **GPR** (General Purpose Register). The variables that we store in the Data Memory during the program are deleted after we turn off the micro. These two memories have separated data buses, which makes the access to each one of them very easy.
  3. **Data EEPROM (Electrically Erasable Programmable Read-Only Memory)** – A memory that allows storing the variables as a result of burning the written program.

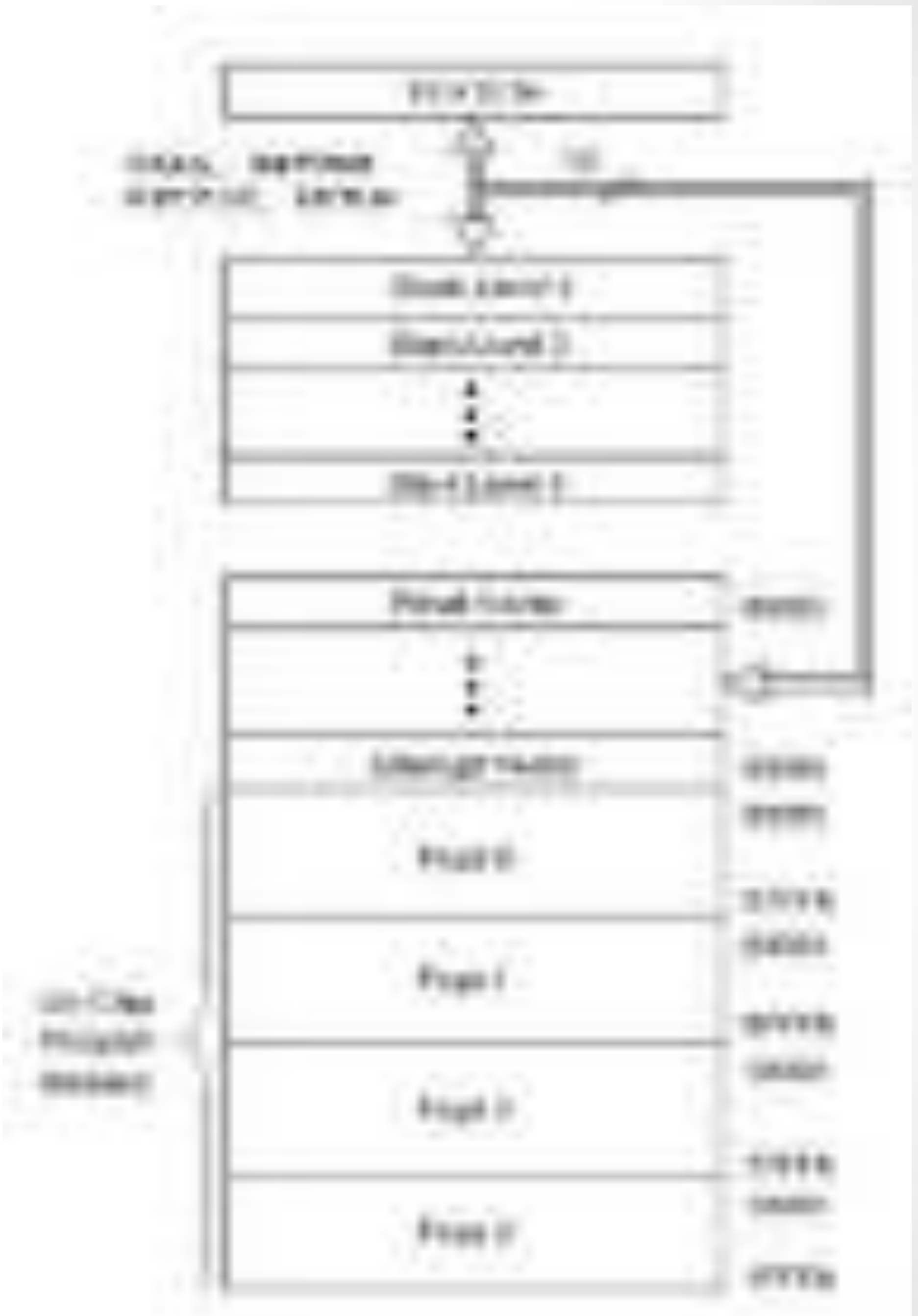
Memory of the PIC16F877..

- Each one of them has a different role. Program Memory and Data Memory two memories that are needed to build a program, and Data EEPROM is used to save data after the microcontroller is turn off.



PIC16F877A Program Memory

- Is Flash Memory
- Used for storing compiled code (user's program)
- Program Memory capacity is 8K x 14 bit → Each location is 14 bits long
  → Every instruction is coded as a 14 bit word
- PC can address up to 8K addresses
- Addresses H'000' and H'004' are treated in a special way



PIC16F877A Data Memory (RAM)

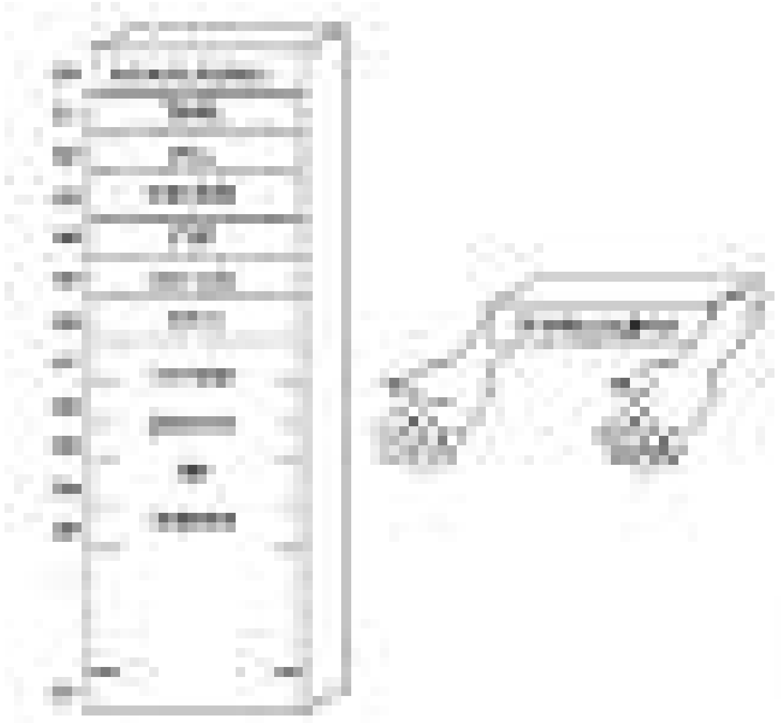
- Memory storage for variables
- Data Memory is also known as Register File and consists of two components.
  - General purpose register file (same as RAM).
  - Special purpose register file (similar to SFR in 8051).
- Addresses range from 0 to 511 and partitioned into 4 banks → each bank extends up to 7Fh (128 bytes).
- The user can only access a RAM byte in a set of 4 banks and only one bank at a time. **The default bank is BANK0.**
- To access a register that is located in another bank, one should access it inside the program. There are special registers which can be accessed from any bank, such as **STATUS** register.

PIC16F877A Registers

- Some CPU Registers:
  - W
  - PC
  - FSR
  - IDF
  - PCL
  - PCLATH
  - STATUS

W Register

- W, the working register, is used by many instructions as the source of an operand. This is similar to accumulator in 8051.
- It may also serve as the destination for the result of the instruction execution. It is an 8-bit register.



Program Counter

- Program Counter (PC) is 13 bit and capable of addressing an 8K word x 14 bit program memory space.
- PC keeps track of the program execution by holding the address of the current instruction.
- It is automatically incremented to the next instruction during the current instruction execution.

- **Program Counter Stack**

- an independent 8-level stack is used for the program counter.
- As the PC is 13-bit, the stack is organized as 8x13bit registers.
- When an interrupt occurs, the PC is pushed onto the stack. When the interrupt is being served, other interrupts remain disabled. Hence, other 7 registers of the stack can be used for subroutine calls within an **interrupt service routine** or within the mainline program.

FSR & INDF

- **FSR Register**

- (File Selection Register, address = 04H, 84H)

  is an 8-bit register used as data memory address pointer. This is used in indirect addressing mode.

- **INDF Register**
- (INDirect through FSR, address = 00H, 80H)

  INDF is not a physical register. Accessing INDF is actually access the location pointed to by FSR in indirect addressing mode.

PCL & PCLATH

- **PCL Register**
- (**P**rogram **C**ounter **L**ow Byte, address =02H, 82H)

  PCL is actually the lower 8-bits of the 13-bit Program Counter. This is a both readable and writable register.

- **PCLATH Register**
- (**P**rogram **C**ounter **LAT**c**H**, address = 0AH, 8AH)

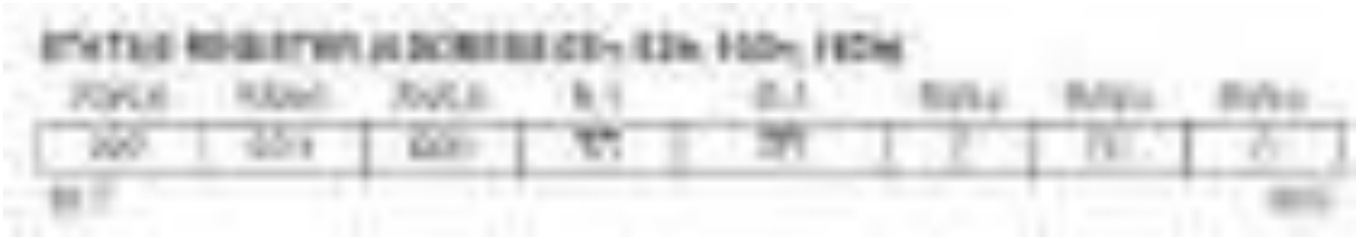
  PCLATH is a 8-bit register which can be used to decide the upper 5-bits of the PC. PCLATH is not the upper 5bits of the PC. PCLATH can be read from or written to without affecting the PC. The upper 3 bits of PCLATH remain zero and they serve no purpose. When PCL is written to, the lower 5bits of PCLATH are automatically loaded to the upper 5bits of the PC.

Memory Map Registers

- In order to start programming and build automated system, there is no need to study all the registers of the memory map, but only a few most important ones:
  - **STATUS register** – changes/moves from/between the banks.
  - **PORT registers** – assigns logic values (“0”/”1”) to the ports
  - **TRIS registers** – data direction register (input/output)

STATUS Register

- Is an 8-bit register that stores the status of the processor.
- In most cases, this register is used to switch between the banks (Register Bank Select), but also has other capabilities.


[illegible]

- **IRP - Register Bank Select bit.**
- **RP1:RP0: - Register Bank Select bits.**
- **TO: Time-out bit**
- **PD: Power-down bit**

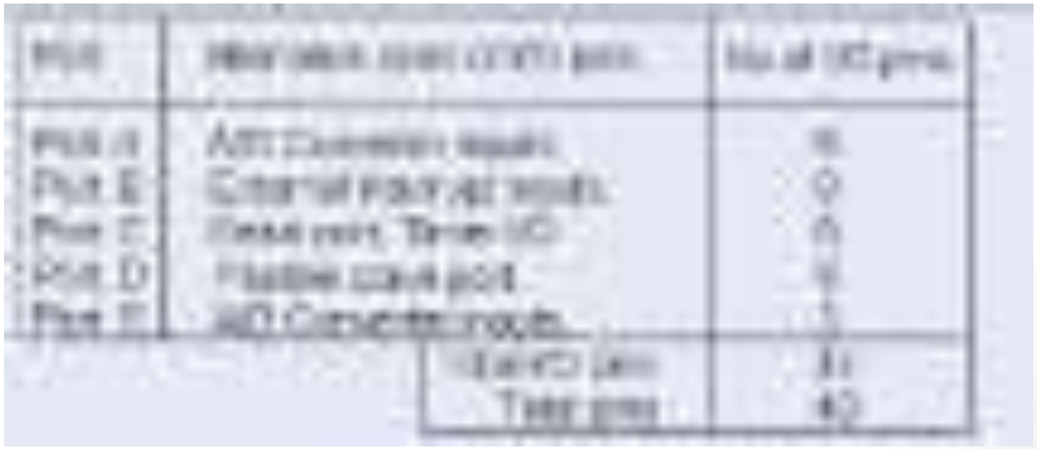
$\}$ **Used in conjunction with PIC's sleep mode**

- **Z: Zero bit**
- **DC: Digit carry/borrow bit**
- **C: Carry/borrow bit**

PIC16F877 Peripheral features

1. I/O Ports:

- PIC16F877 has **5 I/O ports**:
    - **PORT A** has 6 bit wide, Bidirectional
    - **PORT B, C, D** have 8 bit wide, Bidirectional
    - **PORT E** has 3 bit wide, Bidirectional
- In addition, they have the following alternate functions:



Port	[illegible]	No. of I/O pins
Port A	A/D Conversion inputs	6
Port B	External interrupt inputs	8
Port C	Serial port, Timer I/O	8
Port D	Parallel slave port	8
Port E	A/D Converter inputs	3
	I/O pins	33
	Total pins	40

PIC16F877 Peripheral features..

- Each port has **2 control registers**:
  - **TRIS**x sets whether each pin is an input(1) or output(0)
  - **PORT**x sets their output bit levels or contain their input bit levels.
- Pin functionality "overloaded" with other features.
- Most pins have 25mA source/sink thus it can drive LEDs directly.

PIC16F877 Peripheral features…

2. Analog to Digital Converter (ADC)

- Only available in 14bit and 16bit cores
- Fs (sample rate) < 54KHz
- The result is a **10 bit digital** number
- Can generate an interrupt when ADC conversion is done
- The A/D module has **4 registers**:
    - A/D Result High Register (**ADRESH**)
    - A/D Result Low Register (**ADRESL**)
    - A/D Control Register0 (**ADCON0**)
    - A/D Control Register1 (**ADCON1**)
- Multiplexed **8 channel inputs**
    - Must wait **T** $_{\textbf{acq}}$ to change up sampling capacitor.
- Can take a reference voltage different from that of the controller.

PIC16F877 Peripheral features….

3. Timer/counter modules

- Generate interrupts on timer overflow
- Can use external pins as clock in/ clock out (ie. for counting events or using a different Fosc)
- There are **3 Timer/counter modules**:
  - Timer0: 8-bit timer/counter with 8-bit pre-scaler
  - Timer1: 16-bit timer/counter with 8-bit pre-scaler, can be incremented during SLEEP via external crystal/clock
  - Timer2: 8-bit timer/counter with 8-bit period register, pre-scaler and post-scaler.

PIC16F877 Peripheral features…..

4. Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection.

- Asynchronous communication: UART (RS 232 serial)
  - Can do 300bps – 115kbps
  - 8 or 9 bits, parity, start and stop bits, etc.
  - Outputs 5V → needs a RS232 level converter (e.g MAX232)
- Synchronous communication: i.e with clock signal
  - SPI = Serial Peripheral Interface
    - 3 wire: Data in, Data out, Clock
    - Master/Slave (can have multiple masters)
    - Very high speed (1.6Mbps)
    - Full speed simultaneous send and receive (Full duplex)
  - I2C = Inter IC
    - 2 wire: Data and Clock
    - Master/Slave (Single master only)
    - Lots of cheap I2C chips available; typically < 100kbps

PIC16F877 Peripheral features…..

5. Capture, Compare, PWM modules

- Capture is 16-bit, max. resolution is 12.5 ns
- Compare is 16-bit, max. resolution is 200 ns
- PWM max. resolution is 10-bit

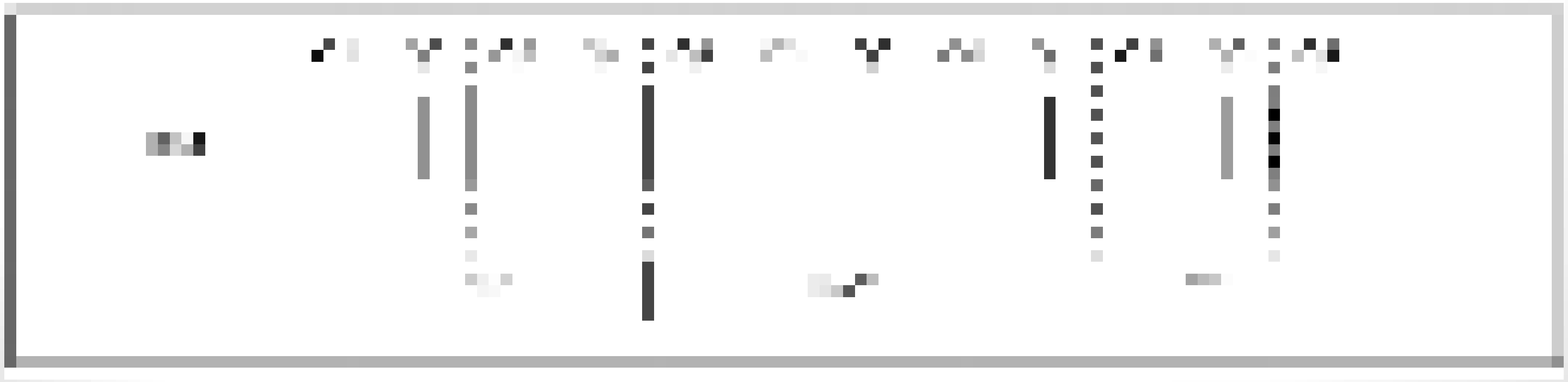
6. Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls

Clock and Instruction Cycles

- Clock from the oscillator enters a microcontroller via OSC1 pin where internal circuit of a microcontroller divides the clock into four even clocks Q1, Q2, Q3 and Q4 which do not overlap.
- These four clocks make up one **instruction cycle** (also called machine cycle) during which one instruction is executed.

Clock and Instruction Cycles..

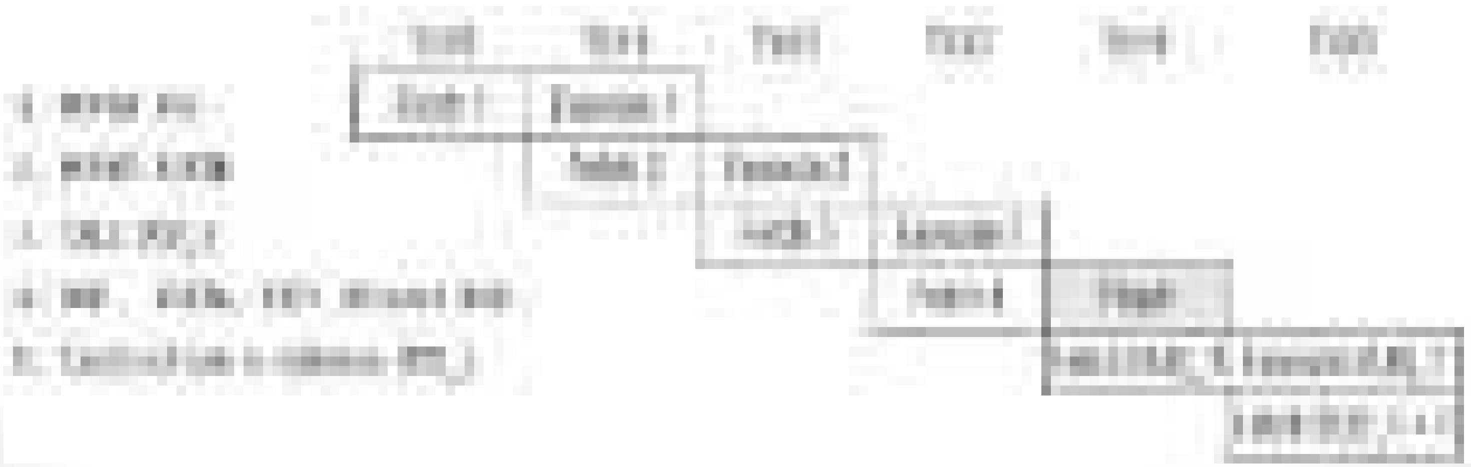
- Execution of instruction starts by calling an instruction that is next in string.
- Instruction is called from program memory on every Q1 and is written in Instruction Register (IR) on Q4.
- Decoding and execution of instruction are done between the next Q1 and Q4 cycles. The following diagram shows the relationship between instruction cycle and clock of the oscillator (OSC1) as well as that of internal clocks Q1 – Q4.
- Program Counter (PC) holds information about the address of the next instruction.



Pipelining in PIC

- There are 35 single word instructions. A two-stage pipeline overlaps fetch and execution of instructions. As a result, all instructions execute in a single cycle except for program branches. These take two cycles since the fetch instruction is “flushed” from the pipeline while the new instruction is being fetched and then executed.
- A typical picture of the pipeline is shown in Figure 3.

Figure3: Instruction Pipeline Flow



- For more details, refer to:
  - Chapter 14, John Catsoulis, **Designing Embedded Hardware**, 2005.
- The lecture is available online at:
  - http://bu.edu.eg/staff/ahmad.elbanna-courses/12134
- For inquires, send to:
  - ahmad.elbanna@feng.bu.edu.eg

Raspberry Pi

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Raspberry Pi, developed by Raspberry Pi Foundation in association with Broadcom, is a series of small single-board computers and perhaps the most inspiring computer available today. From the moment you see the shiny green circuit board of Raspberry Pi, it invites you to tinker with it, play with it, start programming, and create your own software with it. Earlier, the Raspberry Pi was used to teach basic computer science in schools but later, because of its low cost and open design, the model became far more popular than anticipated.

Audience

This tutorial will be useful for people of all ages especially students who want to take their first step in computer science. The reader can be a beginner or an advanced learner.

Prerequisites

The readers must have basic knowledge about Linux and Python programming language. They should also be aware of the basics of electronic circuits.

Copyright & Disclaimer

Table of Contents

1. Raspberry Pi – Introduction

Raspberry Pi, developed by Raspberry Pi Foundation in association with Broadcom, is a series of small single-board computers and perhaps the most inspiring computer available today.

From the moment you see the shiny green circuit board of Raspberry Pi, it invites you to tinker with it, play with it, start programming, and create your own software with it. Earlier, the Raspberry Pi was used to teach basic computer science in schools but later, because of its low cost and open design, the model became far more popular than anticipated.

It is widely used to make gaming devices, fitness gadgets, weather stations, and much more. But apart from that, it is used by thousands of people of all ages who want to take their first step in computer science.

It is one of the best-selling British computers and most of the boards are made in the Sony factory in Pencoed, Wales.

Generations and Models

In 2012, the company launched the Raspberry Pi and the current generations of regular Raspberry Pi boards are **Zero, 1, 2, 3, and 4**.

Generation 1 Raspberry Pi had the following four options:

- Model A
- Model A +
- Model B
- Model B +

Among these models, the **Raspberry Pi B models** are the original credit-card sized format.

On the other hand, the **Raspberry Pi A models** have a smaller and more compact footprint and hence, these models have the reduced connectivity options.

Raspberry Pi Zero models, which come with or without GPIO (general-purpose input output) headers installed, are the most compact of all the Raspberry Pi boards types.

Speed Specifications

The table below gives the speed specifications of various Raspberry Pi models and generations focusing on the version’s release date, form factor and dimensions:

Raspberry Pi Version	**Release Date**	**Form Factor**	**Dimensions (in mm)**
Raspberry Pi 4 Model B	2019-2020	Standard	85.6 x 56.5
Raspberry Pi 3 Model B+	2018	Standard	85.6 x 56.5
Raspberry Pi 3 Model B	2016	Standard	85.6 x 56.5
Raspberry Pi 3 Model A+	2018	Compact	65 x 56.5
Raspberry Pi Zero Wireless with Headers	2017	Mini	65 x 30 x 5
Raspberry Pi Zero Wireless	2016	Mini	65 x 30 x 5
Raspberry Pi Zero	2015	Mini	65 x 30 x 5
Raspberry Pi 2 Model B	2015	Standard	85.6 x 56.5
Raspberry Pi 1 Model B +	2014	Standard	85.6 x 56.5
Raspberry Pi 1 Model B	2012	Standard	85.6 x 56.5
Raspberry Pi 1 Model A+	2014	Compact	65 x 56.5
Raspberry Pi 1 Model A	2013	Standard	85.6 x 56.5

The table below gives the speed specifications of various Raspberry Pi models and generations focusing on the version's weight, General Purpose Input/Output (GPIO), central processing unit (CPU) speed, Cores and Random-access memory (RAM):

Raspberry Pi Version	Weight (in grams)	GPIO	CPU Speed	Cores	RAM
Raspberry Pi 4 Model B	46	40 Pin	1.5 GHz	Quad	1,2,4, or 8 GB
Raspberry Pi 3 Model B+	50	40 Pin	1.4 GHz	Quad	1 GB
Raspberry Pi 3 Model B	40	40 Pin	1.2 GHz	Quad	1 GB
Raspberry Pi 3 Model A+	28	40 Pin	1.4 GHz	Quad	512 MB
Raspberry Pi Zero Wireless with Headers	10	40 Pin	1 GHz	Single	512 MB
Raspberry Pi Zero Wireless	10	40 Pin Unpopulated	1 GHz	Single	512 MB
Raspberry Pi Zero	8	40 Pin Unpopulated	1 GHz	Single	512 MB
Raspberry Pi 2 Model B	42	40 Pin	1.2 GHz	Quad	1 GB
Raspberry Pi 1 Model B +	42	40 Pin	700 MHz	Single	512 MB
Raspberry Pi 1 Model B	38	21 Pin (26 Pin Header)	700 MHz	Single	512 MB
Raspberry Pi 1 Model A+	23	40 Pin	700 MHz	Single	512 MB
Raspberry Pi 1 Model A	30	21 Pin (26 Pin Header)	700 MHz	Single	256 MB

Connectivity Specifications

The table below gives the connectivity specifications of various Raspberry Pi boards focusing on the version's full sized USB ports, other USB and charge methods, power and High-Definition Multimedia Interface (HDMI) ports:

Raspberry Pi Version	Full sized USB Ports	Other USB & Charge Methods	Power	HDMI Ports

Raspberry Pi 4 Model B	2 USB3.0 2 USB2.0	1 USB-C	5.1V at 3A	2 micro-HDMI
Raspberry Pi 3 Model B+	4 USB2.0	1 MIcroUSB	5.1V at 2.5A	HDMI, Composite (TRRS)
Raspberry Pi 3 Model B	4 USB2.0	1 MIcroUSB	5.1V at 2.5A	HDMI, Composite (TRRS)
Raspberry Pi 3 Model A+	1 USB2.0	1 MIcroUSB	5.1V at 3A	HDMI, Composite (TRRS)
Raspberry Pi Zero Wireless with Headers	__	1 MIcroUSB	5.1V at 1.2A	Mini-HDMI, GPIO Composite
Raspberry Pi Zero Wireless	__	1 MIcroUSB	5.1V at 1.2A	Mini-HDMI, GPIO Composite
Raspberry Pi Zero	__	1 MIcroUSB	5.1V at 1.2A	Mini-HDMI, GPIO Composite
Raspberry Pi 2 Model B	4 USB2.0	1 MIcroUSB	5.1V at 1.8A	HDMI, Composite (TRRS)
Raspberry Pi 1 Model B +	4 USB2.0	1 MIcroUSB	5.1V at 1.2A	HDMI, Composite (TRRS)
Raspberry Pi 1 Model B	2 USB2.0	1 MIcroUSB	5.1V at 3A	PAL and NTSC, HDMI or DSI, RCA
Raspberry Pi 1 Model A+	1 USB2.0	1 MicroUSB or GPIO	5.1V at 700mA	HDMI, Composite (TRRS)
Raspberry Pi 1 Model A	1 USB2.0	1 MicroUSB or GPIO	5.1V at 700mA	PAL and NTSC, HDMI or DSI, RCA

The table below gives the connectivity specifications of various Raspberry Pi boards focusing on the version's video out quality, video in, Ethernet, bluetooth, Wi-Fi and external storage:

Raspberry Pi Version	**Video Out Quality**	**Video In**	**Ethernet**	**Bluetooth**	**Wi-Fi**	**External Storage**

Raspberry Pi 4 Model B	4kp60	CSI Camera Connector	Gigabit Ethernet	Bluetooth 5.0	Dual Band-2.4 GHz and 5GHz	MicroSD
Raspberry Pi 3 Model B+	1080p60	CSI Camera Connector	10/100 Mbit/s	Bluetooth 4.2/BLE	Dual Band-2.4 GHz and 5GHz	MicroSD
Raspberry Pi 3 Model B	1080p60	CSI Camera Connector	10/100 Mbit/s	Bluetooth 4.1	2.4 GHz	MicroSD
Raspberry Pi 3 Model A+	1080p60	CSI Camera Connector	____	Bluetooth 4.2/BLE	Dual Band-2.4 GHz and 5GHz	MicroSD
Raspberry Pi Zero Wireless with Headers	1080p60	CSI Camera Connector	____	Bluetooth 4.1	2.4 GHz	MicroSD
Raspberry Pi Zero Wireless	1080p60	CSI Camera Connector	____	Bluetooth 4.1	2.4 GHz	MicroSD
Raspberry Pi Zero	1080p60	CSI Camera Connector	____	____	____	MicroSD
Raspberry Pi 2 Model B	1080p60	CSI Camera Connector	10/100 Mbit/s	____	____	MicroSD
Raspberry Pi 1 Model B +	1080p60	CSI Camera Connector	10/100 Mbit/s	____	____	MicroSD

History

Software developer Eben Upton and Software Engineers Pete Lomas and David Braden formed the Raspberry Pi foundation in 2006. The main aim of this foundation was to devise a computer to inspire children. Hence, in order to reduce the cost, the early prototypes of the Raspberry Pi were based on the 8-bit Atmel ATmega microcontroller.

On February 29th, 2012, the team started taking the orders for Model B and in the same year, they started its production run which consisted of around 10,000 units. These models were manufactured by the founders in China and Taiwan.

On February 4th, 2013, they started taking the orders for lower cost Model A. Similarly, on November 10th, 2014, the team launched for even more low-cost Model A+. The cheapest Raspberry Pi Zero was launched on November 26th, 2015.

The name Raspberry Pi was chosen with "Raspberry" as an ode to tradition of naming early computer companies after fruit. Here, "Pi" is for Python Programming Language.

2. Raspberry Pi — Getting Started

In this modern age when computers are sleek, Raspberry Pi seems alien with tiny codes printed all over its circuit board. That's a big part of Raspberry Pi's appeal. Let us have a look at what we can do with this appealing circuit board.

Uses

Like a desktop computer, you can do almost anything with the Raspberry Pi. You can start and manage programs with its graphical windows desktop. It also has the shell for accepting text commands.

We can use the Raspberry Pi computer for the following:

- Playing games
- Browsing the internet
- Word processing
- Spreadsheets
- Editing photos
- Paying bills online
- Managing your accounts.

The best use of Raspberry Pi is to learn how a computer works. You can also learn how to make electronic projects or programs with it.

It comes with two programming languages, **Scratch** and **Python**. Through GPIO (general-purpose input output) pins, Raspberry Pi can be connected to other circuits, so that you can control the other devices of your choice.

Retailers and Distributors

Some of the global retailers from whom you can buy your Raspberry Pi computers are as follows. You can also refer to their respective websites for details about the Raspberry Pi computers.

- Electronics manufacturing company, **Pimoroni** (www.Pimoroni.com)
- Electronics store, **The Pi Hut** (https://thepihut.com)
- U.S. based electronics company, **Adafruit** (www.adafruit.com)

You can also get it from the following Raspberry Pi's distributors:

- Electronic components supplier, **RS Components** (www.rs-components.com)
- Electronic components distributor, **Element14** (www.element14.com)

Requirements

To use your Raspberry Pi board, you need to buy a few other bits and pieces. Following is the checklist of what else we might need:

Monitor

The Raspberry Pi uses a high-definition multimedia interface (HDMI) connection for video feed, and you can connect your monitor directly with this interface connection, if your monitor has an HDMI socket.

Television

In the similar way, if you have High Definition Television (HD TV), you can also connect it to your Raspberry Pi using an HDMI socket. It will give you a crisper picture.

USB hub

Depending on the model, Raspberry Pi has 1, 2, or 4 Universal Serial Bus (USB) sockets. You should consider using powered USB to connect other devices to your Raspberry Pi at the same time.

Keyboard and Mouse

Raspberry Pi only supports the USB keyboards and mouse. If you are using keyboards and mouse with PS/2 connectors, you need to replace them with Raspberry Pi.

SD or MicroSD card

As we know that the Raspberry Pi does not have a hard drive, so we need to use SD cards or MicroSD cards (depending on the model) for storage.

USB Wi-Fi adapter

If you are going to use model A and A+ then, you need to buy a USB Wi-Fi adapter for connecting to the internet. This should be done because these Raspberry models do not have an Ethernet socket.

External hard drive

If you want to share your collection of music and movies, you need to use an external hard drive with your Raspberry Pi model. You can connect the same by using a powered USB cable.

Raspberry Pi Camera Module

The Raspberry Pi camera module originated at Raspberry Pi foundation. It is an 8MP (megapixel) fixed focus camera that can be used to shoot high-definition video and take still photos. For wildlife photography at night, it provides another version without an infrared filter.

Speakers

The Raspberry Pi has a standard audio out socket. This socket is compatible with headphones and speakers that use a 3.5mm audio jack. We can plug headphones directly to it.

Power supply

For power supply, it uses a Micro USB connector. Hence theoretically, it is compatible with a mobile phone and tablet charger.

Cables

Following are some of the cables, which you need for the connections to the Raspberry Pi computer:

- HDMI cable
- HDMI-to-DVI adapter, if you are using a Digital Visual Interface (DVI) monitor.
- RCA cable, if you want to connect to an older television.
- Audio cable
- Ethernet cable

Compatible and Incompatible Devices

To minimise the cost, the Raspberry Pi models are designed to be used with whatever accessories we have. But, as we know that in practice, not all the devices can be compatible.

You need to check for compatible and incompatible devices as incompatible USB, keyboards and mouse can cause problems.

You can find the list of compatible and incompatible devices at https://elinux.org/RPi_VerifiedPeripherals.

3. Raspberry Pi — Operating System

Before you get started with your Raspberry Pi board, you need to provide with an OS (operating system). **Linux** is the most frequently used OS on the Raspberry Pi.

For using an OS, we need to create a Secure Digital (SD) or MicroSD card with an OS on it. The prerequisite for setting up the SD or MicroSD is a computer having an internet connection and the ability to write to SD or MicroSD cards.

NOOBS Software

NOOBS means **new-out-of-box software** and it is the easiest way to get started with the Raspberry Pi. It is easy to copy NOOBS to your SD or MicroSD card. Once copied, it provides us with a simple menu for installing various operating systems.

There is an option to buy a card with NOOBS already installed on it, but it is always useful to know how to create your own NOOBS cards.

Download NOOBS

Follow the below given steps to download NOOBS:

Step 1: Go to the website www.raspberrypi.org/downloads/noobs

Step 2: Select from the two versions of NOOBS available. **Version 1** is the main version and includes Raspbian. This is the officially supported OS, which you can use even without any network connection.

Another option is to choose the OS from the menu. You can download and install the OS from the menu, if you have a network connection. It is always recommended to download NOOBS for your first OS.

MicroSD card Formatting

Before downloading and installing OS, we first need to format our SD or MicroSD card. We can use an application program, called SD card Formatter, from SD Association. The latest version is SD Memory Card Formatter 5.0.1.

For Windows and Mac, it can be downloaded from the link https://www.sdcard.org/downloads/formatter/.

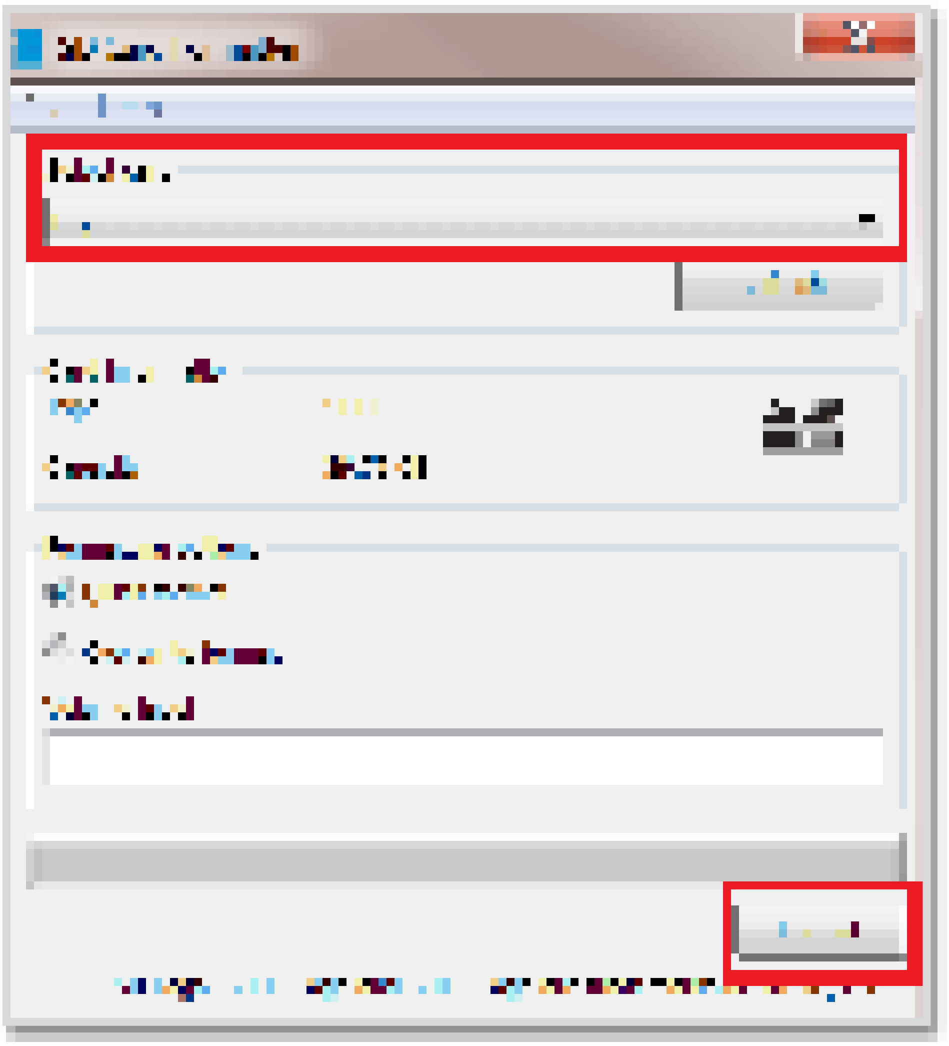
Let us see how we can format the SD card by using windows, Mac OS, and Linux.

Using Windows

Step 1: Download and install the SD formatter application. It will be as follows:

Step 2: Next, we need to select the drive in which we have our SD High Capacity SDHC/SDXC card. Once selected, click on the format button to format it.

The following screen will appear:



Step 3: The program will ask for the confirmation. You need to click **yes** to confirm the format process.

Step 4: Once the format process is completed, your SD card will be formatted completely.

Using Mac OS

The process of formatting is similar as we did in windows. You just need to download and install the Mac version of SD card formatter.

Using Linux

We will be using the **GParted** application program, which is an open source partition manager for Linux.

Use the steps given below to format a SD card in Ubuntu software:

Step 1: Download and install the **GParted** application by using the terminal as follows:

```
sudo apt-get install gparted
```

Step 2: Once installation is completed, you need to insert the SD card. Next, by using Unity dash, launch the **GParted** application.

Step 3: You will get the screen as below, which shows the partitions of the removable disk. But before starting the formatting, we need to unmount the disk by right-clicking on the partition as shown below:

Step 4: After unmounting, we need to right click on it, which will show us the **Format to** option. Now from the list, you can choose whatever type of file system you want on the disk.

After selecting the drive to format, you need to click on the **Tick sign** as shown below:

Step 5: It will show you a couple of warnings and the format procedure will be started.

Install NOOBS to Memory card

Now, you have a formatted card and the .zip file that was downloaded from the Raspberry website. Hence, you can install NOOBS on your card.

On windows PC, you can simply double click the .zip file. It will open the file. Once opened, you can select all the files and copy them to your formatted card.

Similarly, on a Mac OS, you can see the folder that contains all the files by double clicking on the NOOBS .zip file. Now, click on the **Edit menu** and select all. Drag all the files onto your SD card.

In the same way, on Linux we can use the desktop environment to copy the NOOBS .zip files to our SD card.

Flashing a MicroSD card

Some operating systems (OS) may not be available through NOOBS. One of them is the Reduced Instruction Set Computer (RISC) OS.

For creating a card for such an OS, we need to first download the OS as an image file. Once an image file is downloaded, we need to use the process called flashing your card. Later on, the single file can be converted into all the files which we need on our card (SD or MicroSD).

To download the OS images, we can find the links at the website https://www.raspberrypi.org/software/.

Now to flash the card or you can say burning an image to the card, we can use an OS image flasher **Etcher**. It is available for windows, Mac OS and Linux at https://www.balena.io/etcher/.

4. Connecting Raspberry Pi

It is quite easy to connect Raspberry Pi. Let us understand about the same in detail in this chapter.

Ports and Sockets

You should make sure that you have to face your Raspberry Pi in the right way. Most components and sockets, with the help of which you connect it, are sticking out at the top side whereas the back side is relatively flat. The spiky GPIO (general-purpose input output) pins should be at the top left.

Let us have a look at the diagrams below representing the location of connectors and main integrated circuits (ICs) on the Raspberry Pi boards.

The source of the diagrams is https://core-electronics.com.au

Diagram 1

Following is the diagram for **Raspberry Pi Model B:**

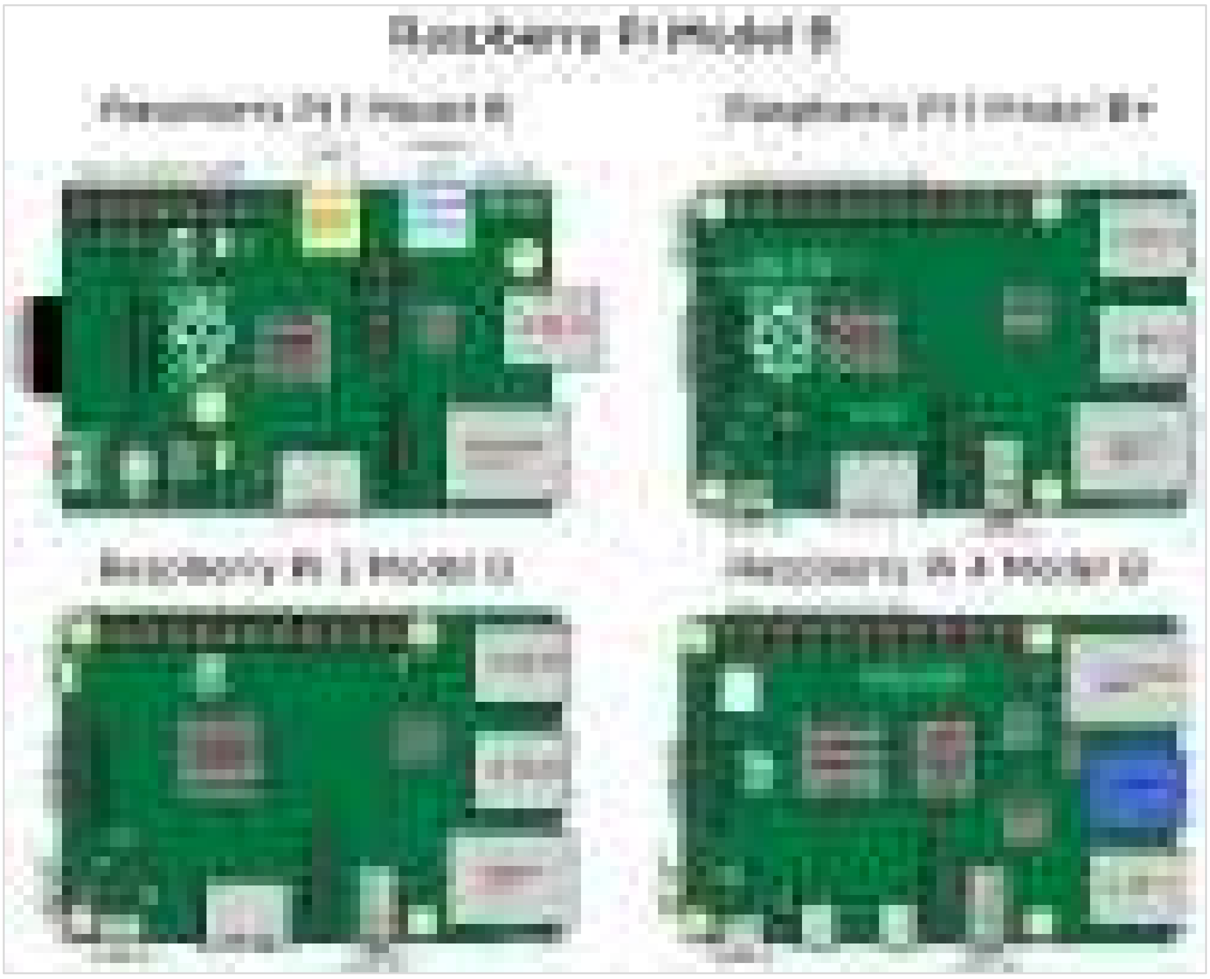


Diagram 2

Following is the diagram for **Raspberry Pi Model A:**

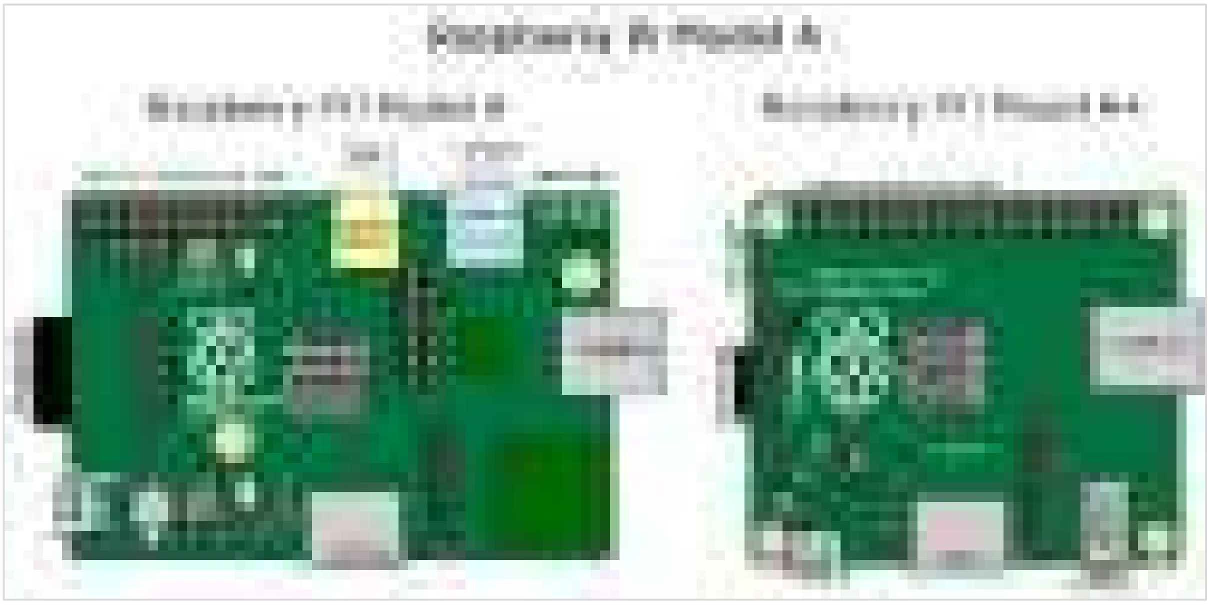
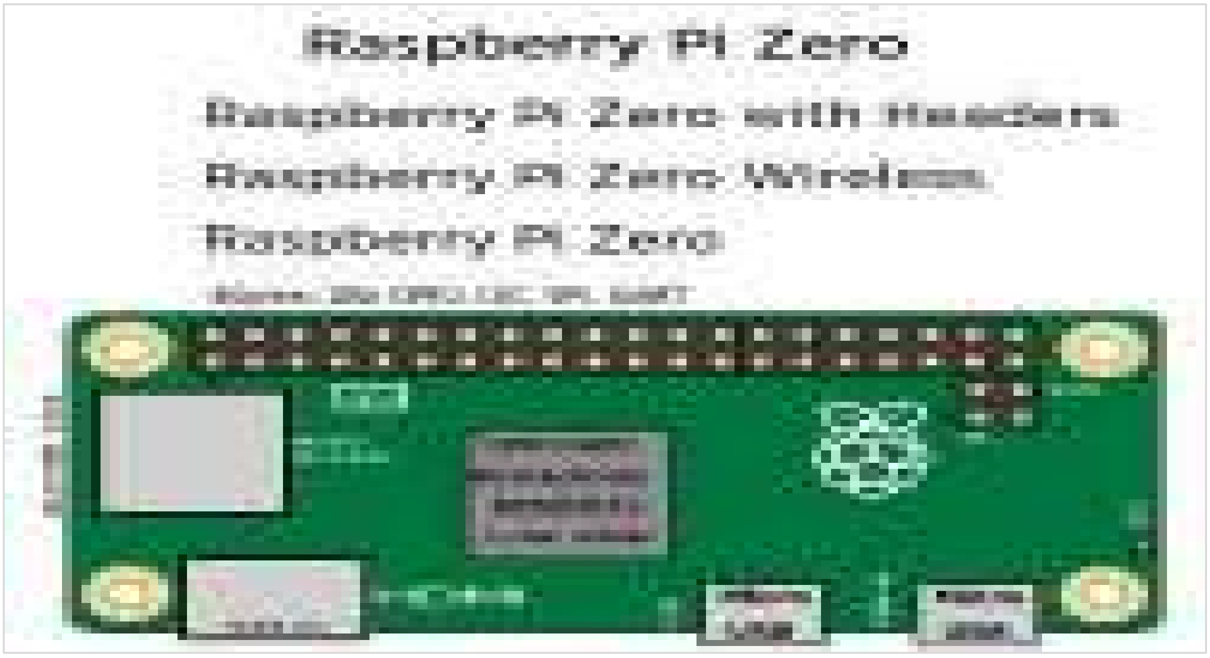


Diagram 3

Following is the diagram for **Raspberry Pi Zero**:



Insert SD or MicroSD card

As we have discussed, you need an SD or MicroSD card with OS to get started with Raspberry Pi. We have also discussed how you can create one, in the previous chapter. Now, it is time to insert that card and get started.

If you are using **model 2, 3, A+, or B+** then, you need to turn your Raspberry Pi circuit board, so that the underside is at your side and you can see that.

You can see, there would be a metal MicroSD card slot on the **left side** of the board. Slide your card into this slot.

On the other hand, if you are using **Model A or Model B**, you need an SD card and you need to flip your Raspberry Pi over. Now, slide the SD card while facing the **label side** above. After that you need to gently press the card home.

And we know that the models **Pi Zero and Zero W** have the MicroSD card slot mounted on the top surface of the board. To insert the card, you need to put the label side facing you.

Camera Module

Camera module, an official module from the Raspberry Pi board, is a small circuit board with a strip of ribbon cable. It plugs directly into the board.

You can see the diagram below:

From the above diagram, you can see that for protection, the lens has a plastic film over it. You need to pull the green plastic tab to remove the film.

On Raspberry Pi Zero

The Raspberry Pi model camera socket uses a different width of cable and you can buy that cable separately. You can also get that cable with the official Raspberry Pi Zero case. You can check the board and the camera have similar sockets for the cable.

To open the connector, you just need to gently press the connector between your finger and thumb. The camera connector is on the right of the Raspberry Pi board.

To connect the cable with the camera, insert the cable with the shiny contacts facing the camera front. And on the Pi Zero board, insert the cable with the shiny contacts facing the flat side of the board i.e., the **bottom side.**

On other Raspberry Pi Models

To connect the camera on other boards, you need to hold the ends in between your finger and thumb. Then, gently lift the board and it will move apart to make a gap. This is the place, where you will insert the cable of the camera.

At the end of the camera's cable, you can see there are silver connectors on one side. Now hold the cable in such a way that this side faces to the left.

Once done, insert the cable into the connector on your Raspberry Pi board. Press it gently and then press the socket back together again and your board is ready with the camera.

Connect Raspberry Pi to Devices

The respective processes to connect your Raspberry Pi board to different devices is explained below in detail. Let us begin by understanding how to connect a display device to your Pi board.

Display device

Depending on the screen type, you have two ways to connect the display device to your Pi board. In these two ways, we are assuming that you are going to use either monitor or television. Apart from these two ways, there is an official Pi touchscreen that connects using the display socket. Let us check how we can connect an HDMI display and television, as explained below.

HDMI or DVI display

The HDMI connector is on the top surface of your Raspberry Pi board. But for the Raspberry Pi Zero model, you need to use an adapter that converts Mini HDMI to an HDMI socket. For connecting, insert one end of the HDMI cable in the board or Pi ZERO connector and the other end into your monitor.

On the other hand, if you are using a DVI display, an adapter should be used.

Television

If the TV you are using is having a HDMI socket, you can use that for optimal results. But if in case, your TV does not have an HDMI socket, you need to use the composite video socket.

On the Raspberry Pi Model-A and Model-B, the composite video socket is placed on the top edge of the board. It is a round, yellow-and-silver sockets.

On other models, Raspberry Pi 3, Pi 2, and Model B+, the same socket as the audio output can be used as a composite video socket. It is placed on the bottom of the board.

One thing you should note is that you will need to use a special **RCA cable** for this socket. Connect one end of the RCA cable to the **audio output socket** and the other end to **Video in** socket of the TV.

If you are using Pi Zero or Zero W boards then, you need to solder your own connector to the board, where it is labelled TV. This should be done because, both these boards do not have composite video socket.

Keyboard and Mouse

On Raspberry Pi Model B+, Model Pi 2, and Model Pi 3 the keyboard and mouse can be directly connected. They should work fine. But for earlier models of Raspberry Pi, you should use an external USB hub to connect keyboard and mouse.

Because with this, the devices will not draw too much power from the Pi board, and we can reduce the risk of heat and other problems caused by devices.

On the other hand, for Raspberry Pi Zero, Model A, and Model A+, we must use a USB hub, since these boards have only one USB socket.

Audio devices

Raspberry Pi's audio socket is a small black or blue box. On Model A and Model B, it is stuck along the top edge of the board. Whereas, on Model B+, Pi 2 and Pi 3, it is stuck along the bottom edge of the board.

If you have connected an HDMI TV, then you do not need to connect a separate audio cable, as the sound is routed through your HDMI cable.

On the other hand, if you have earphones or headphones with a 3.5mm jack, you can directly plug them into the audio socket.

Alternatively, it is recommended to use a suitable cable, as shown in the below figure. The cable has Pi’s 3.5mm jack on the left and stereo input/output plugs that feed into many stereos shown on the right.

Internet router

All the Raspberry Pi models other than Model A, A+, and Zero have an Ethernet socket. You can find the socket on the right edge of the Raspberry board. To connect to the internet, you can use a standard Ethernet cable in this socket.

In case if you are using a router with DHCP (Dynamic Host Configuration Protocol) support, your Raspberry Pi will automatically connect to the internet.

On the other hand, if you have a Wi-Fi adapter then, you can plug into a USB socket of Raspberry Pi and it will be ready to use whenever you turn on your board.

Power

Once you are done with connecting all the necessary and required devices, it is time to connect your Raspberry Pi to power and turn it on. For this, you need to use the Micro USB power socket.

To safeguard your board from damage, you need to provide a steady **5v of power**. Keep in mind that Raspberry Pi board has no on/off switch. It means, whenever you connect it with power, it will start working.

If you want to turn it off, you just need to disconnect it. So, if you want to save your data, you should proceed with caution and should shut down the Raspberry Pi first.

Turn on Raspberry Pi

Connect with the power and turn on your Raspberry Pi board. There will be a rainbow of colors on screen. Afterwards, it will start to run the NOOBS software on the Memory card. You will get a choice of OS to install.

Below are the OS choices in NOOBS:

Raspbian

Raspbian, a version of a Linux distribution called Debian, is the distribution that is recommended by the Raspberry Pi foundation. It has been optimized for the Raspberry Pi board.

Most of the Raspberry Pi users start with Raspbian and it includes:

- Graphical Desktop software.
- Web browser.
- Development and programming tools like Scratch, Python etc.

It has two versions, one with the PIXEL desktop and other is termed as Raspbian Lite, with a more minimal installation.

LibreELEC and OSMC

Both are the versions of Kodi media center. They are mainly used for playing music and video.

RISC OS

It is an alternative to Linux OS, which most of the people use on the Raspberry Pi. It has a GUI (Graphical User Interface). In 1987, it was created by Acorn Computers and now-a-days, it is maintained and managed by RISC OS open Limited.

Data Partition

If you use the Data Partition option, it will give you an option to sort the data. The sorted data can be accessed by various Linux distributions.

Lakka

It is a retro gaming system that includes emulators for a range of vintage home computers such as Commodore 64 and Amiga, Amstrad CPC, ZX Spectrum and various Atari machines.

It also includes emulators for a range of game consoles such as Nintendo machines and Sony PlayStation. Although the Bomberman clones and 2048 games are included but, if you want to use Lakka, you need to get the games separately.

Plug your USB with games files and you will be ready to get games into Lakka.

Recalbox

It is another game system. It also includes emulators for Super Nintendo Entertainment System (SNES), Nintendo Entertainment System (NES), Game Boy Advance, PC Engine, and Sega Master System. The shareware version of a famous game called Doom is also included in the Recalbox game system.

Screenly OSE

As the name implies, it is a digital signage system. It enables the users to use a Raspberry Pi with a connected HD screen as a digital sign. Here, OSE refers to Open Source Edition.

It enables the following to be displayed on the screen:

- Videos
- Images
- Web pages

Screenly OSE is also suitable for displaying the advertisements and information in public areas like shops, schools, offices, shopping malls, railway stations, etc.

Windows 10 IoT Core

As the name implies, it is the version of Windows which is designed to support the IoT (Internet of things) devices. It is actually different from the windows desktop experience we are familiar with.

Once installed, it will give us the following two versions:

- **RTM version:** It is the release to manufacturing (RTM) version. It is recommended to use because it is a stable version as compared to the Pre-release version.
- **Pre-release version:** Another is pre-release version, which is less stable as compared to RTM version.

TLXOS

This is ThinLinX's thin client software. It is a trial version and enables the Raspberry Pi to work as a virtual desktop. By using ThinLinX, we can also manage one or more Raspberries centrally.

5. Raspberry Pi — Configuration

In this chapter, we will learn about configuring the Raspberry Pi. Let us begin by understanding how to configure Raspberry Pi board in Raspbian.

Raspbian configuration

For configuring Raspberry Pi in Raspbian, we are using Raspbian with PIXEL desktop. It is one of the best ways to get Raspbian started with the Raspberry Pi. Once we finish booting, we will be in the PIXEL desktop environment.

Now to open the menu, you need to click the button that has the Raspberry Pi logo on it. This button will be in the top left. After clicking the button, choose **Raspberry Pi configuration** from the preferences.

Configuration tool

Following is the configuration tool in PIXEL desktop:

By default, the configuration tool opens to its system tab which has the following options:

- **Change Password:** The default password is **raspberry.** You can change it by clicking the change password button.

- **Change the hostname:** The default name is **raspberry pi**. You can also change it to the name, which you want to use on the network.
- **Boot:** You can choose from the two options and control whether Raspberry Pi boots into the desktop or CLI i.e., command line interface.
- **Auto Login:** With the help of this option, you can set whether the user should automatically log in or not.
- **Network at Boot:** By choosing this option, you can set whether the pi user is automatically logged in or not.
- **Splash screen:** You can enable or disable it. On enabling, it will display the graphical splash screen that shows when Raspberry Pi is booting.
- **Resolution:** With the help of this option, you can configure the resolution of your screen.
- **Underscan:** There are two options, enable or disable. It is used to change the size of the displayed screen image to optimally fill the screen. If you see a black border around the screen, you should disable the underscan. Whereas, you should enable the underscan, if your desktop does not fit your screen.

There are three other tabs namely Interfaces, Performance, and Localization. The job of interface tab is to enable or disable various connection options on your Raspberry Pi.

You can enable the Pi camera from the interface tab. You can also set up a secure connection between computers by using SSH (short for Secure Shell) option.

If you want to remote access your Pi with a graphical interface then, you can enable RealVNC software from this tab. SPI, I2C, Serial, 1-wire, and Remote GPIO are some other interfaces you can use.

There is another tab called Performance, which will give you access to the options for overclocking and changing the GPU memory.

The localization tab, as the name implies, enable us to set:

- The character set used in our language.
- Our time zone.
- The keyboard setup as per our choice.
- Our Wi-Fi country.

Configure Wi-Fi

You can check at the top right, there would be icons for Bluetooth and Wi-Fi. The fan-shaped icon is on the Wi-Fi. To configure your Wi-Fi, you need to click on that icon. Once clicked, it will open a menu showing the available networks. It also shows the option to turn off your Wi-Fi.

Among those available networks, you need to select a network. After selecting, it will prompt for entering the Wi-Fi password i.e., the Pre Shared Key.

If you see a red cross on the icon, it means your connection has been failed or dropped. To test whether your Wi-Fi is working correctly, open a web browser and visit a web page.

Configure Bluetooth Devices

We can use wireless Bluetooth devices such as keyboard and/or mouse with Pi 3 and Pi zero W because these models are Bluetooth-enabled. In PIXEL desktop, you can set up your Bluetooth devices easily.

Following are the steps to configure the Bluetooth devices:

- First, make your device discoverable for pairing.
- Now, you need to click on the Bluetooth menu at the top right of the screen. It is aligned to the Wi-Fi button.
- Now, choose the Add Device option.
- The Raspberry will start searching for the devices and when it finds your device, click it and click the pair button.

Data Partition Setup

As we know that data partition is that area on your memory card (SD or MicroSD) which can be shared by various distributions. One of the best examples of use of a data partition is transferring the files between distributions.

The data partition has the **label** data.

You can use this labeled data to make a directory point to it as follows:

Step 1: First, you need to boot the Raspberry Pi into Raspbian.

Step 2: Now, click the Terminal icon to get to the command line.

Step 3: Next, type the command **mkdir shared**. It will create a directory named **shared**.

Step 4: Write the command **sudo mount -L data shared**. This command will point the directory to the shared partition.

Step 5: Write the command **sudo chown $USER: shared**. It will set the permission for writing in this shared folder.

Step 6: Now, to go to this shared folder, you need to type the command **cd shared**.

Once all the files are created in this shared folder, they will be available to all the distributions that have the permission to access the data partition.

6. Raspberry Pi — Working with Linux

This chapter enlightens about the functioning of Raspberry Pi with Linux.

PIXEL Desktop Environment

PIXEL (Pi Improved Xwindows Environment, Lightweight) is a visual desktop environment which is a part of the recommended Raspbian Linux distribution. It is the quickest way to get started with Raspberry Pi and by default, it appears when our Raspberry Pi computer finishes starting up.

Some of the characteristics of PIXEL are as follows:

- It is based on LXDE (Lightweight X11 Desktop Environment) open-source desktop.
- Raspberry Pi foundation has redesigned LXDE and converted it into a PIXEL desktop environment.
- The PIXEL desktop environment works in a similar way to Mac OS and Windows OS.
- We can manage and find files by using mouse and icons.
- Using this desktop environment, it’s really intuitive to navigate.

Navigate Desktop Environment

The image below is of the PIXEL desktop environment. You can see a taskbar (a strip along the top of the screen), which is usually visible in every program we will be using.

The Application Menu

For most of the programs, which we would like to run under PIXEL desktop or any other desktop environment, we need to use the application menu. You can get it by clicking the Raspberry Pi icon at the top left side of the desktop screen.

You will see the image as follows:

Submenu Programs

You will get the submenu program on the right, after moving the cursor over the categories of the programs. It will show the programs in that particular category.

You need to click on that category to start with that. If you want to add that category icon to the desktop, just right-click that program on the menu.

Following are the wealth of programs under submenu program:

Claws Mail

It is in the internet part of the Application with the help of which you can send or receive messages on your Raspberry Pi computer.

Debian Reference

As we have discussed earlier, the Raspbian version of Linux is the Pi-specific version of the Debian distribution. This icon will guide us how to use Linux on your Raspberry Pi computer.

This is a reference document, which is stored on your SD card and to find this, you need to go through the help section of the Application menu.

To get started with this, first, you need to click the icon and then, click the multi-files link (it is an HTML link) which is at the top of the screen.

LibreOffice

This is the most popular suite of productivity applications. It mainly includes word processing, spreadsheets, and presentations. You can get it from the office section of the Application menu.

Mathematica

Mathematica, under the programming section of the Application menu, is based on the Wolfram programming language. It is used for scientific and technical computing.

Minecraft Pi

We know about the world-building game called Minecraft. Similarly, Minecraft Pi is the Raspberry version of that. You can find it under the Game section of the Application program, and you can also program it by using the Python programming language.

Python 2 and Python 3

Raspberry Pi provides us the Python programming language, which can be found under Programming in the Application menu. We can also use the **Thonny IDE** (integrated development environment) which provides the Pi users, an alternative way of creating Python programs.

Python games

Raspberry Pi has games such as Reversi, Four in a Row, a sliding puzzle game as well as a snake game. These all are built in Python programming language and can be found in the Game section of the Application menu.

Scratch

Raspberry Pi foundation provides us a simple programming language called Scratch, which is approachable for the peoples of all ages. You can use it to create games and animations. It can also be used to manage electronic projects. You can find it under the Programming section of the Application menu.

Sense HAT emulator

As the name implies, it has some built-in sensors that can be used for creating experiments and other projects. It is an add-on for the Raspberry Pi users, which can be found under the Programming section of the Application menu.

Shutdown

Shutdown, a top-level option in the Application menu, can be used for switching off your Raspberry Pi, before you remove the power. With this, we will also get the options to log out as well as restart your Pi computer.

Sonic Pi

It is another programming language provided by Raspberry Pi foundation which is mainly used for creating music. You can also find it under the Programming section of the Application menu.

Terminal

Terminal is a window that let us issue the instructions from a command line without leaving your PIXEL desktop environment. There are two ways through which you can reach the terminal window. One is to get it in the Accessories part of the Programs menu and other is to use the button on the taskbar.

Wolfram

Wolfram is a programming language provided by Raspberry Pi foundation. It aims to incorporate knowledge, so that the programmers can get results quickly. You can get more information about this at www.wolfram.com/language. It is under the Programming section of the Application menu.

Running programs

Even after installing, some of the programs won't appear on the Application menu. You can use the **Run option** to run those programs.

Follow the below given steps:

Step 1: First, we need to open the Application menu. For this, click the icon at top left of the desktop.

Step 2: Now, we need to select the Run option from this menu.

Step 3: Run option will give you a dialog box. You can type the name of the program, which you want to open, and then press Enter.

Close and Rearrange Programs

The controls for closing and rearranging the programs on the PIXEL desktop environment is like the ones in MS Windows. These controls enable us to close as well as resize (minimize and maximize) the programs.

You can find these controls in the top right. They are explained below:

- **X button:** It is used for closing the programs/applications.
- **Maximize button:** As the name implies, this button will enlarge a particular application. Once used, the application will fill the screen.

- **Minimize button:** As the name implies, this button will reduce a particular application. It will hide the program from view but does not stop it from executing/running. We can return to the program by clicking the name of the program on the taskbar.

7. Raspberry Pi — PIXEL Desktop Environment

Let us learn about the PIXEL desktop environment in Raspberry Pi. First, we will understand what a task manager is.

Task Manager

Sometimes, it may happen that your Raspberry Pi computer does not seem to be responding. But, there is nothing to worry about. This happens when the computer might be quite busy.

The diagram below shows the task manager.

In the top right side, you can see the CPU usage monitor, which will tell you how heavily your Pi's processor is being used. Moreover, in the top left side of this taskbar, we have three buttons, which are collectively called as the closing and resizing window.

There are two options to open the task manager, which are as follows:

- Go to the Accessories folder on the Application menu.
- Use the shortcut key, which is by holding down **Ctrl** and **ALT** keys and then pressing the **Delete** key.

If any of the program is responding and you want to terminate it, just right-click in the task list. The menu will appear, and you can choose the Term from it. This option will give a chance to shut down the program safely.

On the other hand, we can also use Kill, but, this option will terminate the program immediately with loss of data.

File Manager

It is easy to manage your files in a PIXEL desktop environment rather than using the command line. Refer the screen given below for the file manager:

With the help of file management, we can browse, copy, rename or delete the files on our Raspberry Pi or other connected storage devices.

You have two options to start the file manager, which are as follows:

- Click the button on the left of the desktop.
- Go to system tools under the Application menu.

Navigate File Manager

There is an icon bar, having useful shortcuts, under the file manager’s menu bar.

Let us navigate the file manager and understand the icons that come under it.

Add Tab

Suppose, if you want to work in two folders at the same time. For example, copy files from one folder to another. Then, you need to quickly switch between those two folders. Tabs become handy for this purpose.

It enables us to have two different folders open at the same time, so that we can simply click them to switch between them. You can close the tab by clicking the cross (X) icon on the tab.

Previous folder

Previous folder button, as the name implies, takes you back to the last folder, which we have accessed on that tab. It works a bit like a web browser's back button.

Next folder

Next folder button, as the name implies, takes us to a folder, where we have visited after the folder on which we have been working. We will end up where we started, if we will first click the Previous folder button and then the Next folder button.

Folder History

The Folder history button, as the name implies, will open a menu having the folders we have visited.

Up a level

There can be parent and child folders in your Raspberry Pi desktop. For example, the Desktop folder is inside the Pi folder. Hence, the Pi folder will be the parent folder and Desktop folder will be the child folder. The up a level button will take you to the parent folder.

Home

The home button, as the name implies, takes us back to the Pi folder.

Path

Path, as you have seen in the web browser's URL bar, is the text description if it is the location of the folder, we are working with. It also includes the list of the folders, which are above it.

Cut, copy, move files and folders

The file manager in the PIXEL desktop environment makes it easy to move your files and folders from one place to another. It also makes it easy to cut, copy, and paste your files and folders.

You just need to right-click a file or folder of your choice and a menu will appear. This menu has the following options:

- Renaming the file.
- Moving the file to the wastebasket.
- Cut or copy the files.

Now, if you want to cut the file or folder, right-click on that and choose option **cut.** After that, right-click an empty space, where you want to paste that folder. From the menu that appears, select **paste** and your file or folder will be pasted at that empty space.

Likewise, if you want to copy the file or folder, you need to choose option **copy** from the right-click menu and then, paste wherever you want. It will create a duplicate file or folder.

Multiple files and folders

If you want to select more than one file at a time, there are following methods to do it:

First method

You need to hold the CTRL key and then, click on every file you want to select.

Second method

For selecting a group of consecutive icons, you first need to click the first icon, press the SHIFT key, and then click the last icon.

Third method

In this method, you need to click the mouse on the background of the file manager. Now while holding the button, you need to loop all the files which you want to select.

Moving the files

Now for moving these files, you have the following methods:

- Once you have selected the files, you can drag these files into a different folder.
- Otherwise, you can choose the option of cut or copy the whole group of files by right-clicking one of the selected files.

Keyboard Shortcuts

Like MS windows, you can also use some shortcuts in PIXEL as follows:

- **Ctrl+A:** To select all the files and folders.
- **Ctrl+C:** To copy the files and folders.
- **Ctrl+V:** To paste the files and folders.
- **Ctrl+X:** To cut the files and folders.

Organize files in folders

To easily manage your files, you can organize them in a folder. It is easy to create a new folder.

Follow the below given steps:

- First, select and go to the location, typically your **pi** folder, where you want to create a new folder.
- Now, right-click a blank space in the **File Manager** and click on the option **Create New** from menu.
- Now another menu will appear, and you need to click **Folder** from that menu. You will now be prompted to enter a name. Enter the name you want to give to the new folder and click **OK** to confirm.

The other method to create a New Folder is to click the File menu at the top left of the File Manager and find **Create New**. With these methods, you can also create empty files.

Delete files and folders

If you want to delete a single file or folder, you can right-click that in the File Manager. For the menu, you need to choose the option **Move to Wastebasket**.

On the other hand, if you want to delete more than one file or folder, you can select all of them as we did before, and choose the option **Move to Wastebasket** right-click menu.

You can also use the keyboard **Delete button** to send the selected files to the wastebasket.

Sorting the files

You can sort your files in Raspberry Pi by name, size, file type, modification time, etc. For this, you again need to right-click the empty space in the right pane of the file manager. A menu will appear and you need to select the option to change how the files are sorted.

You can also change how your files are displayed in the file manager. For this, you need to use the View menu on the Menu bar at the top of the File Manager.

The View menu will give us the following four ways to display our files and folders:

Icon view

It is the default option used by the File manager. It strikes a good balance between the size of each icon and number of files, we can see at one time.

Thumbnail view

Another view option is thumbnail view, which is mostly used in a folder of images. It enlarges the preview.

Compact view

As the name implies, the compact view lists the files and folders in columns and this is done with a small icon and filename. It helps us to view as many files as possible at a time.

Detailed view

As the name implies, this view reveals detailed information like short description, size, last modification date, etc., about the file.

Now, let us continue learning about the other important aspects with regards to the PIXEL desktop environment in Raspberry Pi.

Browsing the Web

Raspberry Pi gives us the choice of four browsers for browsing the web, namely, Chromium, Dillo, Netsurf and Epiphany. You can type the name of the browsers into the run option on the Application menu and it will appear.

Chromium Browser

It is the recommended browser. Apart from the Run option, you can also access it by clicking the Web browser button (the Globe icon) in the top left of the screen.

The layout of Chromium browser is quite similar to other browsers. It has a thin toolbar and provides a maximum screen to the page which you are accessing. It provides the user with the facility for ad-blocker to strip advertising. You can also change the settings for the same.

Dillo Browser

This browser is fast. Hence, it is a good choice for those users who have a slow internet connection and have problems related to accessing mainly the text information. It does not support Javascript and cannot handle the sophisticated layout instruction.

This is the reason that the web pages look different than the ones intended on it. This browser provides the users with an option to switch off the images from the Tool menu to speed up the downloads of complex pages.

Netsurf

This web browser is capable of handling more sophisticated layouts than Dillo web browser. But like Dillo, netsurf also lacks support for Javascript. Hence, the websites (including Facebook) that require Javascript won't work on Netsurf.

Epiphany

It supports Javascript and was the recommended browser before Chromium. Epiphany browser is optimized for the Raspberry Pi but, might be noticeably slower than what we are used to.

Claws Mail

Raspberry Pi provides us an open-source email program called Claws Mail. It is preinstalled and you can find it in the internet category of the Application menu.

Following are some prerequisites, if you want to use email on Raspberry Pi:

- For sending emails, you need to know the details of the server. You can find this information on the website of your email provider.
- Your email user ID and password. This should be the same as you use when logging on with webmail.

Sending and Receiving Emails

Follow the below given steps to use Claws mail for sending and receiving emails:

- First, you need to add an account from the configuration wizard of Claws mail. Apart from adding a new account, you can also edit the account settings, delete an account by using the configuration menu.
- Once you are done with configuration, go to top left and click the Get mail button. It will show your mail folder in the left and messages on the right at the top.

- To read the messages, you can use two methods. One is to use the message preview pane at the bottom right and other is to double-click on a message to open in its own window.
- For composing a new message, replying, and forwarding a message, there is a menu bar across the top of Claws mail.

Image Viewer

If you want to look at your Digital images and work with them in Raspberry Pi, PIXEL provides us with the Image Viewer. You can find it in among the accessories on the Application Menu.

Toolbar buttons

You will see a toolbar underneath the picture opened in the Image Viewer.

Following are the buttons on that toolbar:

Previous

As the name implies, with this button, you will go to the previous photo in that folder. Any unsaved changes would be lost.

Next

As the name implies, with this button, you will go to the next photo in that folder. Any unsaved changes would be lost.

Start Slide show

This button will begin a slide show of all the photos in that folder. Predefined interval between two photos is 5 seconds. Image Viewer gives us an option to change it in preferences. Keyboard shortcut for starting slide show is key W.

Zoom Out

This button will reduce the magnification of the image. Keyboard shortcut for Zoom Out is the Minus (-) key.

Zoom In

This button will increase the magnification of the image. Keyboard shortcut for Zoom In is the Plus sign (+) key.

Fit Image to Window

It will shrink a large image to make it fit in the Image Viewer window. Its keyboard shortcut is key F.

Go to Original Size

This button will reset all the zooming by showing an image at its original size. Its keyboard shortcut is key G.

Full Screen

As the name implies, this button will expand an image to fill the monitor. With the use of this button, you will lose the Image Viewer controls.

Rotate Left

It will rotate the image 90 degrees counterclockwise. Keyboard shortcut for Rotate Left is key L.

Rotate Right

It will rotate the image 90 degrees clockwise. Keyboard shortcut for Rotate Right is key R.

Flip Horizontally

This button will mirror the image horizontally. Keyboard shortcut for Flip Horizontally is key H.

Flip Vertically

This button will mirror the image vertically i.e., turns an image upside down. Keyboard shortcut for Flip Vertically is key V.

Open File

It will open a new image file. You can also open an image from a folder in File Manager by using the Drag-and-Drop option on Image Viewer.

Save File

It will save an image with changes which you have done. It will replace the original file. The keyboard shortcut key is S.

Save File as

It will save an image, with the changes which you have done, with a new filename. It will not replace the original file.

Delete

It will delete an image from the storage device. If you use this button, an image will be deleted permanently and won’t be recovered.

Preferences

This button holds the settings, which you can change for the Image Viewer. It permits you to customize the settings as per your needs.

Exit Image Viewer

As the name implies, it will close the Image Viewer application. We can also use the close button (X) which is in the top right.

Text Editor

PIXEL has a simple text editor called Leafpad. You can find it by clicking the Text Editor in the Accessories part of the Application Menu. Leafpad text editor is good for writing and word processing but not ideal for creating the print-ready documents.

Leafpad

The menu on Leafpad consists of the following buttons:

File Menu

You can use this menu to start the new documents as well as to open, save, and print the files. It has a quit option, which we can use to close the text editor.

Edit Menu

The Edit menu gives you the tools to do the following tasks:

- Undoing your work.
- Redoing your work.
- Cutting the work.
- Copying your work.
- Pasting the work.
- Deleting the work.
- Selecting all the text.

It uses the same shortcuts as MS Windows, as mentioned below:

- Ctrl+C: To copy the work.
- Ctrl+V: To paste the work.

- Ctrl+X: To cut the work.
- Ctrl+A: To select all the text.

Search Menu

This menu gives us the following options:

- To find a particular word or phrase.
- Jump to a particular line in a document.
- Replace a chosen word or phrase with an alternative.

Option Menu

This menu gives us the following options:

- To change the font.
- Switch on word wrap.
- Switch on line numbers.

Customize your Desktop

You can change the look and feel of your desktop and make it easier to use by doing some changes. The options to customize your desktop are under the Preferences section of the Application menu.

With these options we can do the following:

- Change the picture used as a backdrop i.e., wallpaper.
- Change desktop color, if not using wallpaper.
- Change the color of icon descriptions i.e., the text color.

Install New Applications

Although, we can use the command line to discover and install new software, but, there is also a friendly menu in the PIXEL desktop environment. For that menu, we need to go to the Preferences option and click on **ADD/REMOVE** software.

This menu has the following parts that helps us to find and install new applications:

Search box

The search box is at the top left. Here, you can enter the name of a program you are looking for and it will show you the options.

Main pane

It shows us the packages. The already installed packages will be checked and would be in bold. Tick the box titled **decide it,** if you want to install that package.

After choosing your software, you need to click the Ok button to install and remove the applications. It will prompt for entering the password.

Back up the data

To back up your data, you can use File manager to copy them to a USB key or MicroSD card. Raspberry Pi provides us an application called the SD Card Copier application for copying data.

You can also use the shell commands, which we will discuss further.

8. Raspberry Pi — Linux Shell

The Shell, called Bash in Raspberry Pi, is the text-based way of issuing instructions to your Pi board. In this chapter, let us learn about the Linux shell in Raspberry Pi. First, we will understand how to open a shell window.

Open Shell Window

You can open a shell window by using one of the two following ways:

- There is a Terminal icon, having a >_ prompt, on the top of the screen. Click on it and you will get a shell window.
- Another way is to use the Accessories section of the Application menu. You can find the Terminal there.

Both of the above approaches will open a shell window on the desktop.

Understanding the Prompt

The prompt looks like as follows:

```
pi@raspberrypi ~ $
```

It contains lots of information. Let us see the various bits:

pi

It represents the name of that user who logged in.

raspberrypi

It represents the hostname of the machine i.e.; the name other computers use to identify when connecting to it.

The tilde symbol (~)

The tilde symbol tells the user which directory they are looking at. The presence of this horizontal wiggly line is known as home directory and the presence of this symbol shows that we are working in that directory.

The dollar sign ($)

It represents the presence of the ordinary user and not all-powerful superuser. A # symbol means a superuser.

List Files and Directories

When you start the shell window, you start in your home directory.

To see the folders and files in your home directory, you need to issue a command which is as follows:

```
pi@raspberrypi ~ $ ls
```

Output

The output is as follows:

```
Desktop Downloads Pictures python_games Videos

Documents Music Public Templates
```

You can see the files and folders after issuing the **ls command**.

As we know that Linux is case sensitive hence the commands LS, Ls, ls and lS are all different.

Change the directory

You can see the above output, it's all blue which means these are all directories. We can go to these directories and check which files they contain. The command to change the directory is **cd**. You need to use the cd command along with the name of the directory which you want to see.

The example for changing the directory in Raspberry Pi is given below:

```
pi@raspberrypi ~ $ cd Pictures
```

Find information about files

The command to find the information about a particular file is **file.** You need to put the name of the file after the command to check the information about that file.

Check the below example for finding information about the files in Raspberry Pi:

```
pi@raspberrypi ~ /Pictures $ file leekha.png aarav.png

leekha.png: PNG image data, 50 x 85, 8-bit/color RGBA, noninterlaced

aarav.png: PNG image data, 100 x 150, 8-bit/color RGBA, noninterlaced
```

We can also use the file command on directories. It will give us some information about the directories as well:

```
pi@raspberrypi ~ $ file Pictures Desktop

Pictures: directory

Desktop: directory
```

Parent Directory

Earlier, we have used the cd command to change into a directory that is inside the current working directory. But sometimes, we need to go to the parent directory i.e. into the directory which is above the current working directory.

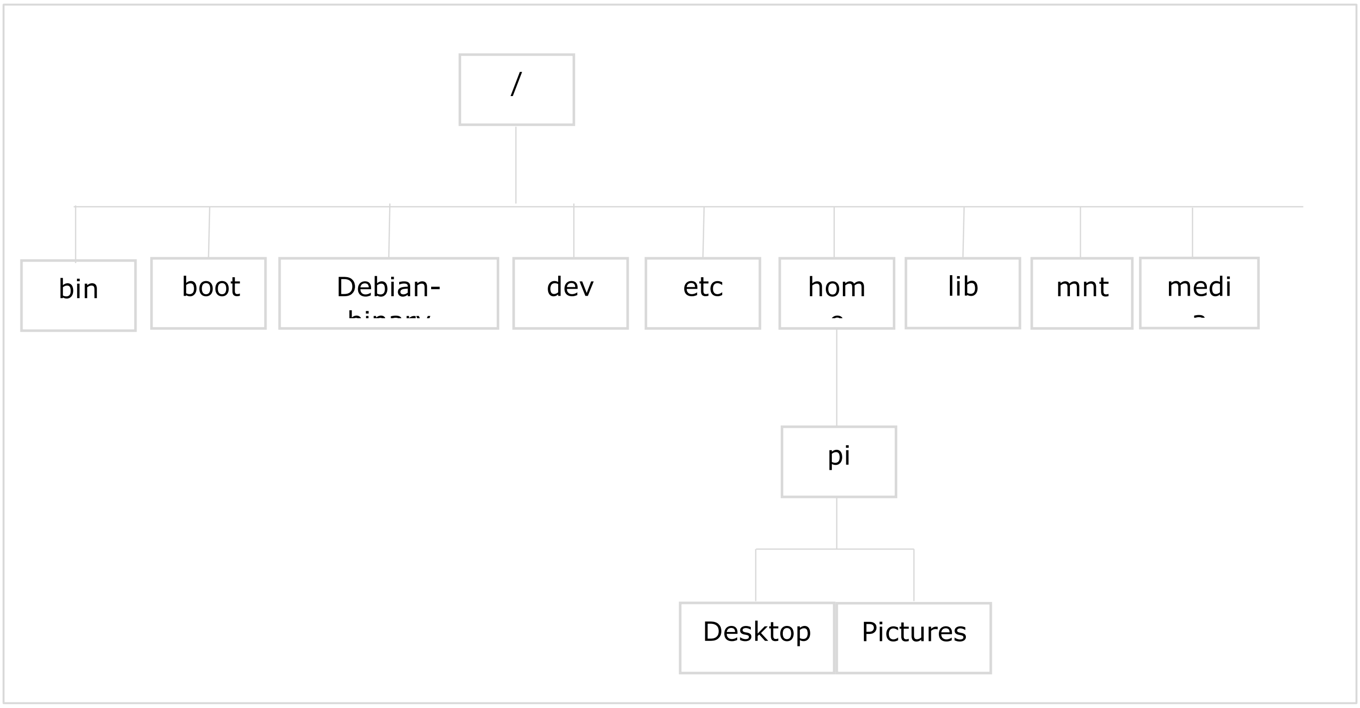
The command for this is cd..(cd with two dots), as given below:

```
pi@raspberrypi ~ /Pictures $ cd..
pi@raspberrypi ~ $
```

The tilde symbol represents your home directory.

Directory Tree

The following diagram shows the part of the directory tree on your Raspberry Pi computer:



The directories and their uses are as follows:

bin

Bin, short for binaries, contains some small programs that behave like commands in the shell. For example, ls and mkdir.

boot

This directory contains the heart of the OS i.e. the Linux kernel. It also contains the configuration files containing the technical settings for Raspberry Pi computer.

dev

This directory contains a list of devices. For example, the devices like disks and network connections.

etc

This directory is used for various configuration files. These configuration files apply to all the users on the computer.

home

This is the directory where a user can store or write files by default.

lib

The directory contains various libraries that are used by different OS programs.

lost+found

This directory is used, if the file system gets corrupted and recovers partially.

media

You connect a removable storage device such as a USB key and it is automatically recognized. All the details will be stored in the media directory.

mnt

mnt stands for mount and will store all the details of the removable storage devices that we mount ourselves.

root

It is reserved for the use of the root users and we don't have the permission to change into this directory as an ordinary user.

Relative and Absolute Paths

The shell enables the Raspberry Pi users to go straight to the location by specifying a path.

We have the following two types of paths:

Relative path

It is a bit like giving the directions to the directory from where the user is now.

Absolute path

On the other hand, an absolute path is like a street address. This path is exactly the same wherever the user is. These paths are measured from the root. Hence, they start with a slash (/).

For example, we know the absolute path to the pi directory is /home/pi.

Now, go straight forward to this directory by using the following command:

```
cd /home/pi
```

If you want to go to the root, you can use the following command:

```
cd /
```

Advanced Listing Commands

We can use the listing command (ls) to look inside any directory outside the current working directory as follows:

```
pi@raspberrypi ~ $ ls /boot
```

There are several advanced options, which we can use with the ls command.

These options are given in the following table:

Option	Description
-1	This option is 1 not l and it outputs the results in a single column instead of a row.
-a	The ls command with this option will display all the files. All the files will also include hidden files.
-F	This option will add a symbol besides a filename. It will do this to indicate the file type. If you use this option, you will notice a / after directories names and a * after executable files.
-h	This option is short for human-readable. It expresses file sizes by using kilobytes, megabytes, and gigabytes.
-l	This option will display the result in the long format. It shows the information about the permissions of files, their last modification date, their size.
-m	This option will list the result as a list separated by commas.
-R	This option is the recursive option. It will also list files and directories in the current working directories, open the subdirectories(if any) and list their results too.
-r	It is the reverse option and will display the result in reverse order.
-S	This option will sort the result by their size.
-t	This option will sort the result as per the date and time they were last modified.
-X	This option will sort your result as per the file extension.

Furthermore, we will learn about the other important aspects related to Linux Shell in Raspberry Pi.

Long Listing Format

Long format is one of the most useful formats of the **ls** command, because it provides us the additional information on the file.

You can use the ls command with the long listing option as follows:

```
pi@raspberrypi ~ $ ls -l
total 65
-rw-r--r-- 1 pi pi  256 Feb 18 22:45 Leekha.txt
drwxr-xr-x 2 pi pi 4096 Jan 25 17:45 Desktop
```

```
drwxr-xr-x 5 pi pi 4096 Jan 25 17:50 Documents

drwxr-xr-x 2 pi pi 4096 Jan 25 17:52 Downloads

drwxr-xr-x 2 pi pi 4096 Jan 25 17:53 Music

drwxr-xr-x 2 pi pi 4096 Jan 25 17:45 Pictures

drwxr-xr-x 2 pi pi 4096 Jan 25 17:45 Public

drwxr-xr-x 2 pi pi 4096 Jan 25 17:54 Templates

drwxr-xr-x 2 pi pi 4096 Jan 25 17:54 Videos
```

From the above output, it is very easy to understand that each line relates to one file or directory having its name on the right and the date and time, when it was last modified next to that.

The number 256, 4096 represents the size of the file. You can see some files and directories are having the same size.

The remaining part of this output shows the permissions i.e. who is allowed to use the file and what the user is allowed to do with that file or directory.

Permissions

The permissions on a file are divided in the following three categories:

Owner

It is the person who created the file. This permission consists of the things the file owner can do.

Group

These are the people who belong to a group that has the permission to use the file. This permission consists of the things which the group owners can do.

World

These are known as the world permissions i.e. the things that everyone can do with that file or directory.

In Raspberry Pi, we have two main types of files. One is regular files which have a **hyphen (-)** and others are directories having a **d**.

Types of Permissions

Now let us understand the different types of permissions the owner, group and world have respectively:

- **Read permission:** This permission gives the user ability to open and look at the contents of a file or to list a directory.
- **Write permission:** This permission gives the user the ability to change the content of a file. It allows the user to create or delete the files in a directory.

- **Execute permission:** This permission gives the user an ability to treat a file as a program and run it. It also gives permission to enter a directory using the **cd** command.

Less Command

The ls command deluges with the information that you cannot even notice sometimes, because it flies past our eyes faster than we understand or see it. To avoid this or solve this problem, we can use a command called **less.**

This command will take our listing and enable us to page through it and that is one screen at a time. To use this command, we need to use a | (pipe character) after the listing (ls) command.

The example of less command in Raspberry Pi is given below:

```
ls -RXF | less
```

The less command can also be used to view the content of a text file.

For this, we need to provide the filename as an argument, as given below:

```
less /boot/config.txt
```

Speed up the use of Shell

Here we will be learning few tricks to speed up the use of the shell:

- If you want to retype a command, then you can save retyping it because, shell **keeps the record of history** i.e. the commands you entered previously.
- In case if you want to reuse your last command, you just need to use **two exclamation marks** and press enter.
- You can also bring back the previous commands in order by tapping the **up arrow**.
- Similarly, you can also move through your history of commands in another direction by tapping the **down arrow**.
- The shell also guesses what the user wants to type and it also automatically completes it for us.

Create File with Redirection

Redirecting files means, you can send the results from a command to a file instead of sending the results to the screen. For this, we need to use a > (greater-than) sign along with the file name, which we would like to send the output to.

The example of creating file by using redirection in Raspberry Pi is given below:

```
ls > ~/gaurav.txt
```

There are other commands too and with their help, we can display the content online. These commands are explained below:

echo command

The echo command, as the name implies, will display on screen whatever we write after it. The best use of this command is to solve mathematical problems. You need to put the expression between two pairs of brackets and put a dollar sign in front.

The example of echo command is given below:

```
echo $((5*5))
```

date command

The date command, as the name implies, will display on screen the current date and time.

cal command

The cal command (cal stands for calculator) will display the current month's calendar with today highlighted. With the help of option **-y**, you can see the whole year calendar.

Create and Remove Directories

Here, we will understand how to create and remove directories in Raspberry Pi. Let us begin by learning about creating directories.

Create Directories

The command to create a directory under your home directory is **mkdir.**

In the below example, we will be creating a directory named **AI_Python**:

```
mkdir AI_Python
```

You can also use one command to create several directories as follows:

```
pi@raspberrypi ~ $ mkdir AI_Python Machine_Learning Tutorialspoint
pi@raspberrypi ~ $ ls
Downloads AI_Python Machine_Learning Tutorialspoint Desktop Pictures Documents
Public
```

Remove directories

If you want to remove an empty directory, you can use the command **rmdir** as follows:

```
pi@raspberrypi ~ $ rmdir AI_Python
```

On the other hand, if you want to remove non-empty directories, you need to use the command **rm -R** as follows:

```
pi@raspberrypi ~ $ rm -R Machine_Learning
```

Delete Files

We can use the rm command to delete a file.

The syntax for deleting a file would be as follows:

```
rm options filename
```

In an example given below, we will be deleting a text file named **leekha.txt**:

```
pi@raspberrypi ~ $ rm leekha.txt
```

Like mkdir, the rm command will not tell us what it is doing.

To know its function, we need to use the verbose(-v) option as follows:

```
pi@raspberrypi ~ $ rm -v leekha.txt
removed 'leekha.txt'
```

We can also delete more than one file at a time as follows:

```
pi@raspberrypi ~ $ rm -v leekha.txt gaurav.txt aarav.txt
removed 'leekha.txt'
removed 'gaurav.txt'
removed 'aarav.txt'
```

Raspberry Pi Wildcards

A directory contains a lot of files with the similar filenames and if you want to delete a group of such files, you don't need to repeat the command by typing out each filename. In shell, wildcards will do this job for us.

Following table provides us a quick reference to the wildcards, which we can use in Raspberry Pi:

Wildcard	Meaning	Example	Description
?	It means any single character.	pic?.jpg	The example means that the files start with a pic and have exactly one character after it before extension starts.
*	It means any number of characters.	*pic*	The example means that any files that have the word pic in their filename.
[…]	This wildcard will match any one of the characters in brackets.	[gla]*	The example means that all the files that start with the letter g, l or a.

[^…]	This wildcard will match any single character that is not between the brackets.	[^gla]*	The example means that any files that do not start with the letter g, l or a.
[a-z]	This wildcard will match any single character in the range specified.	[x-z]*.png	The example means that any files that start with a letter x, y or z and end with the .png extension.
[0-9]	This wildcard will match any single character in the range specified.	Pic[1-5]*.png	The example means that it will match pic1.png, pic2.png, pic3.png, pic4.png, and pic5.png.

The below given example will remove all the files starting with letters **lee**,

```
rm -vi lee*
```

Copy Files

Copying files is one of the fundamentals things we would like to do.

The command for this is **cp,** which can be used as follows:

```
cp [options] copy_from copy_to
```

Here, we need to replace **copy_from** with the file you want to copy and **copy_to** for where you want to copy it.

Example

Let us see an example of using the command to copy the respective file.

Suppose, if you want to copy the file leekha.txt from the /desktop directory to the home directory, you can use the cp command as follows:

```
cp /Desktop/leekha.txt ~
```

We can also specify a path to an existing folder to send the file to as follows:

```
cp /Desktop/leekha.txt ~/doc/
```

Move Files

Rather than making a copy of the file, if you want to move it from one place to another then, you can use the **mv** command as follows:

```
mv ~/Desktop/leekha.txt ~/Documents
```

The above command will move the file named leekha.txt from Desktop directory to the Documents directory. Both of these directories are in the home directory.

Reboot Raspberry Pi

With the help of following command, we can reboot our Raspberry Pi without disconnecting and reconnecting the power:

```
sudo reboot
```

Shutdown Raspberry Pi

With the help of following command, we can safely turn off our Raspberry Pi:

```
sudo halt
```

9. Raspberry Pi — Managing Software

We have discussed the simple menu based ADD/REMOVE software tab under Preferences for installing software. It is one of the easiest ways to manage and install software on your Raspberry Pi.

But here, we will be discussing how we can use the command line to install software in Raspberry Pi.

To install software, we require the authority of the root user or superuser but, sometimes that leaves our Raspberry Pi computer files vulnerable, including to any malicious software that might get in.

We can use **sudo** instead of a root account. Putting **sudo** before a command will indicate that one wants to carry it with the authority of the root user.

Update Cache Memory

If you want to install software on your Raspberry Pi, you first need to update the cache memory. It is the list of the packages, which the package manager knows about.

Use the following command to update the cache memory:

```
sudo apt-get update
```

Find the Software

To find the package name or software, we need to use the package manager cache. In Linux terminology it is the apt, cache.

It contains an index of all the packages available for install. It collects information of the software package and, also used to search for the available packages which are ready for installation on Raspberry Pi.

With the help of following command, we can search the required software:

```
sudo apt-cache search pkgname
```

Suppose if you want to search the games packages, you can use the command as follows:

```
sudo apt-cache search game | less
```

The list might be long. Hence, we have used **less.**

And suppose, if you want to find the package name for a particular game say chess, you can give the title in the command as follows:

```
sudo apt-cache search chess
```

This command will search for all the packages with name **chess.**

Install Software

Once you finish the searching, you can now install the software. For searching, you used **apt-cache**. However, for installing, you need to use **apt-get** command.

The command will download the specific package from the internet and install it. It will also install other dependencies.

For example, if we want to install the chess game say **3dchess**, then the command will be as follows:

```
sudo apt-get install 3dchess
```

Run the Software

Following are the two ways to run a particular program in Raspberry Pi:

From command line

You can run some programs directly from the command line. You need to type the name of the program as follows:

```
3dchess
```

It will directly run the program.

From Application menu

Another way is to use the Application menu. After installing, you can find the application in the Application menu.

In Raspberry Pi, most of the end-user applications require the X-server. It means they need the Desktop environment to run them.

Upgrade Software

You can use the package manager to keep your software.

Following is the command with the help of which we can update all our software:

```
sudo apt-get upgrade
```

On the other hand, if you want to update just one application, you can do it by issuing its install command again.

For example, we have installed the chess game above, now enter the same again:

```
sudo apt-get install 3dchess
```

The above command will prompt the apt to check for any updates of the package and install them. If it finds no updates, it will tell us that we are running the latest version of the software.

Remove Software

You can also use the package manager to remove software from your Raspberry Pi computer.

You can use the following command to remove the software:

```
sudo apt-get remove 3dchess
```

The above command will remove the 3dchess package, but it will leave some traces of the application. These traces might include user files and any files containing settings.

You can also completely remove the application by using the following command:

```
sudo apt-get purge 3dchess
```

What software is installed?

You can use the following command to find out what software is installed on your Raspberry Pi computer:

```
dpkg --list
```

With the help of following command, you can search for a specific package:

```
dpkg –-status packagename
```

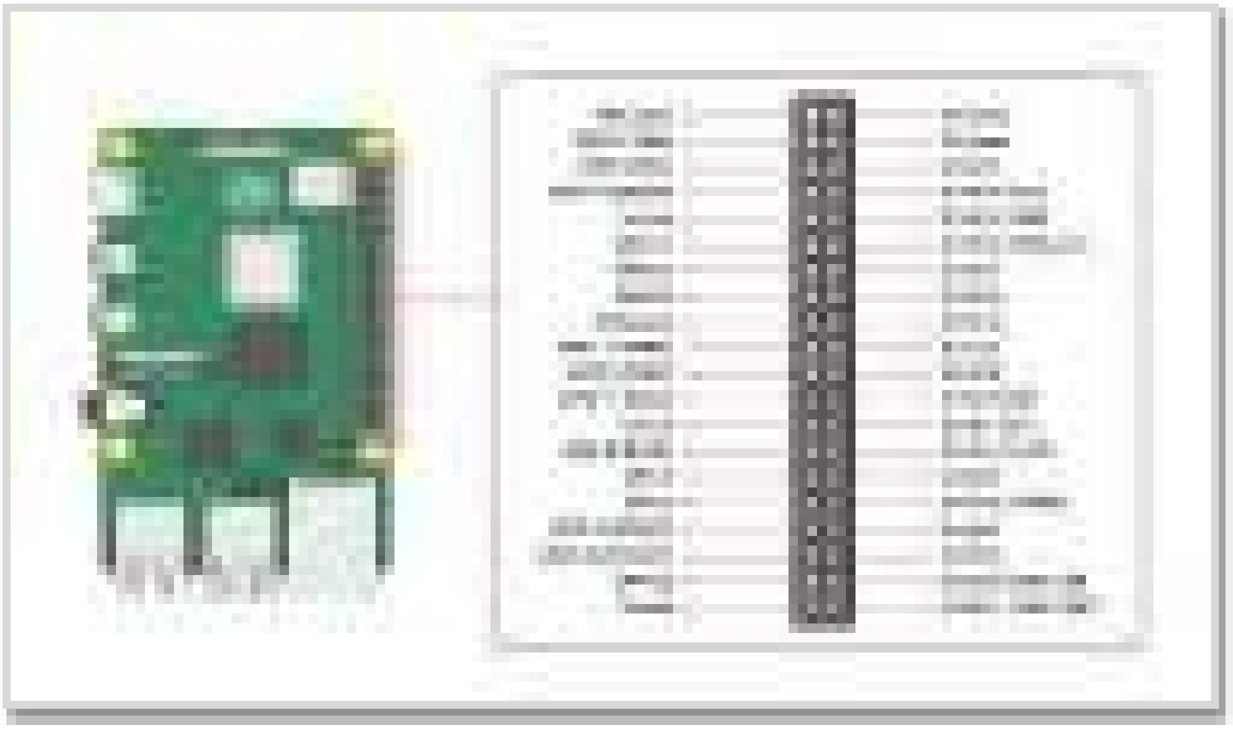
10. Raspberry Pi — GPIO Connector

Here, we will learn about the GPIO (general-purpose input output) connector in Raspberry Pi.

GPIO Pinout

One of the powerful features of the Raspberry Pi is the row of GPIO (general-purpose input output) pins and the GPIO Pinout is an interactive reference to these GPIO pins.

Following diagram shows a 40-pin GPIO header, which is found on all the current Raspberry Pi boards:



The source of the diagram is www.raspberrypi.org

Voltages

From the above diagram, we can see that there are two 5V pins and two 3V3 pins on the board. It also has several ground pins (0V). All these pins are unconfigurable.

Outputs

A GPIO pin can be designated as an output pin. The pin set as output pin can be set to 3V3(high) or 0V(low).

Inputs

A GPIO pin can be designated as an input pin. The pin set as input pin can be read as 3V3(high) or 0V(low). You can use internal pull-up or pull-down resistors.

You can see in the above diagram, GPIO2 and GPIO3 pins have fixed pull-up resistors but for the other pins, you can configure it in software.

Alternative Functions

GPIO pins can be used with a variety of alternative functions. Among them, some are available on all pins and others on specific pins.

PWM: Pulse-width modulation

Software PWM are available on all the pins whereas Hardware PWM are available on GPIO12, GPIO13, GPIO18, and GPIO19.

SPI: Serial Peripheral Interface

The SPI are available on the following:

SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)

SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)

I2C: Inter-integrated Circuit

The I2C are available on the following:

Data: (GPIO2); Clock (GPIO3)

EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)

Serial

The serial function is available at the following:

TX(GPIO14)

RX(GPIO15)

Connect GPIO to Raspberry Pi

Following are some simple rules to reduce the risk of damaging your Raspberry Pi board, while using the GPIO connector:

- Do not try to put more than 3.3V on any GPIO pin.
- Do not try to draw more than 3mA per output. Although, you can draw more but to increase the life of your Pi Board, you should restrict upto 3mA.
- You should not poke the GPIO connector with a screwdriver when the Raspberry Pi board is powered up.
- 5V power is enough for your Raspberry Pi. Don’t try to provide more power than that.
- You should not try to draw more than a total of 50mA from the 3.3V supply pins.

Output of GPIO pins

To set up the output of GPIO pins and to read the input values by using Python, you need to install **RPi.GPIO** python library.

Install RPi.GPIO python library

To install RPi.GPIO python library, type the following commands on terminal window of your Raspberry Pi:

```
sudo apt-get install python-dev

sudo apt-get install python-rpi.gpio
```

Almost all the latest versions of distributions have RPi.GPIO already installed. In that case, the above commands will update it to the latest version.

I2C Device

Let us check how we can make I2C work with Raspberry Pi.

Case 1: Using Adafruit Occidentalis 0.2 or later

In case, if you are using Adafruit Occidentalis, you don't need to do anything. Because, this distribution is preconfigured with I2C support.

Case 2: Using Raspbian

In case if you are using Raspbian, you need to do the following configuration changes:

First, edit the file **/etc/modules** by using the following command:

```
sudo nano /etc/modules
```

Now, we need to add the following lines to the end of this file:

```
i2c-bcm2708

i2c-dev
```

Next, we need to edit the file named **/etc/modprobe.d/raspi-blacklist.conf** and comment out the following line by adding a #:

```
blacklist i2c-bcm2708

#blacklist i2c-bcm2708
```

Once done, install the Python I2C library by using the following command:

```
sudo apt-get install python-smbus
```

Now, reboot your Raspberry Pi and it will be ready for I2C.

Find I2C Address

There is an I2C device attached to Raspberry Pi computer and you want to know its address.

For this, we need to install i2c-tools as follows:

```
sudo apt-get install i2c-tools
```

Once done, attach your I2C device to your Raspberry Pi board and run the following command:

```
sudo i2cdetect -y 1
```

Here, we need to take care about the following two things:

- First, if you have newer distributions then, it is quite possible that it has **i2c-tools** already installed.
- Second, if you have an older version 1 board in use, change 1 to 0 in the above code line.

Serial Peripheral Interface (SPI)

Let us check how we can use SPI (serial peripheral interface) bus with Raspberry Pi.

Case 1: Using Adafruit Occidentalis 0.2 or later

In case if you are using Adafruit Occidentalis, you don't need to do anything because this distribution is preconfigured with SPI support.

Case 2: Using Raspbian

In case if you are using Raspbian, you need to do the following configuration changes:

First, edit the file **/etc/modules** by using the following command:

```
sudo nano /etc/modules
```

Now, we need to add the following lines to the end of this file:

```
spidev
```

Next, we need to edit the file named **/etc/modprobe.d/raspi-blacklist.conf** and comment out the following line by adding a **#**:

```
blacklist spi-bcm2708

#blacklist spi-bcm2708
```

Once done, install the Python library to carry out communication from a Python program by using the following command:

```
cd ~
```

```
sudo apt-get install python-dev

git clone git://github.com/doceme/py-spidev

cd py-spidev/

sudo python setup.py install
```

Now, reboot your Raspberry Pi and it will be ready for SPI.

Serial Port

Suppose if you want to use the serial port i.e. Rx and Tx pins on your Raspberry Pi board but, it is used by Linux OS as a console connection. To disable this, we need to comment out a line in a file named **/etc/inittab**.

Open this file by using the following code line:

```
sudo nano /etc/inittab
```

Now, find the following line by scrolling down at the end of this file:

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Now, we need to use a **#** to comment out this line:

```
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Now, save this file and reboot your Raspberry Pi.

Access serial port from Python

We can also use the serial port i.e. Rx and Tx pins on Raspberry Pi board by using Python. For this, we need to install **PySerial** library as follows:

```
sudo apt-get install python-serial
```

It is recommended to disable (as we did above) the Raspberry Pi's serial console before using **PySerial**.

Test the serial port

Once you start using the serial ports, you may want to send and receive the serial commands from a Terminal session. For this, we need to install **Minicom** as follows:

```
sudo apt-get install minicom
```

It is recommended to disable (as we did above) the Raspberry Pi's serial console before using **Minicom**.

As now Minicom is installed, we can start a serial communication session with a serial device connected to the RXD and TXD pins of the GPIO connector by using the following command:

```
minicom -b 9600 -o -D /dev/ttyAMA0
```

Here, in the above command, after -b is the baud rate and after -D is the serial port. We should use the same baud rate as set on the service we are communicating with.

11. Raspberry Pi — Add-on Boards

There are ready-made boards with all sorts of components built on. Many companies have produced such boards. Such boards come with sample code to show us how to use them.

Types of ready-made boards

We have two types of ready-made boards, Raspberry Pi, which are as follows:

- The boards which are designed for making it easy to get access to the GPIO pins.
- The boards with components that have already soldered up.

Styles of Ready-made Boards

The ready-made boards come in following three styles:

Separate boards

These kinds of ready-made boards connect to the GPIO pins via a ribbon cable or your own wires.

Shield or plates

These kinds of ready-made boards plug into all the GPIO pins and cover most of the area of the Raspberry Pi board.

HATs (Hardware Attached on Top)

These kinds of ready-made boards are like the shields or plates but contain an additional identification. Sometimes, it may contain a software so that the Raspberry Pi can read what they are on start-up and install some software and prepare GPIO pins automatically.

Various Boards

Since the introduction of the Raspberry Pi computer, new boards are constantly being developed and produced.

Some of the boards are as follows:

The Sense HAT

The Sense HAT, specifically designed for the Astro-Pi mission, allows Raspberry Pi to sense the world around it. Two ruggedized versions of Sense HAT were flown on the International Space Station from December 2015. The codes for both the versions were written by school children.

Following are some of the characteristics of the Sense HAT board:

- It has a 8X8 RGB LED matrix.
- It has a 5-button joystick.

- It also has sensors to measure acceleration, temperature, pressure, humidity, and magnetism.
- It also has sensors to measure the gyroscope.
- It has an extensive Python library associated with it. This Python library allows for easy access to this board.

The comprehensive coverage of using the Sense HAT can be found at https://www.raspberrypi.org/learning/getting-started-with-the-sense-hat/.

The Skywriter HAT

The Skywriter HAT is an electric near-field 3D proximity interface, which senses things floating in the air over it.

Some of the characteristics of the skywriter HAT are as follows:

- You can make gestures with your hands because it can detect the motion of your hand with X, Y and Z positions.
- It can detect gestures such as flicks right, left, up, and down.
- It can easily detect the circular motion of the finger.
- It can also detect the taps directly to its surface.
- The range of the skywriter HAT has a range of about 5 cms.
- It can also be mounted behind any non-conducting surface.

The source of the above image is www.magpi.raspberrypi.org

The Xtrinsic Sense Board

The Xtrinsic Sense Board, made in partnership with the component distributors and Raspberry Pi co-manufacturer Farnell, is a low-cost sensor board. It is somewhat like the Sensor HAT, but without LEDs.

Some of the characteristics of the Xtrinsic Sense Board are as follows:

- It contains a high-precision pressure sensor. The range of this sensor is 50 to 110 kPa.
- It also contains a 3-axis digital accelerometer and a 3-D magnetometer.

The source of the above image is www.farnell.com

12. Raspberry Pi — Third-party Software Package

Previously, we have discussed how we can download and install the software on our Raspberry Pi. It is one of the best things about Raspberry Pi.

Here, we will be discussing some of the software packages in Raspberry Pi.

Penguins Puzzle

This is a 3D puzzle game in which you need to safely escort a penguin to exit without letting it fall. It has a total of 50 levels. To move around, you can use the cursor keys. Shortcut key for zoom-out is Z and for reset is R.

Penguin Puzzle is preinstalled with Raspbian but in case, if you want to install or update it, you can find it on ADD/REMOVE software menu.

You can also use the following shell command to install/update it:

```
sudo apt-get install penguinspuzzle
```

Once installed, you can start playing this game by typing **penguinspuzzle** on shell.

FocusWriter

FocusWriter, as the name implies, is a word processing software designed for distraction-free work. While working in FocusWriter, you will only see your writing on screen. To get the menus for changing the setting and saving your file, you need to move the mouse to the top of the screen.

One of the best things about this word processor is that you can set your daily goals for the number of words written per day or time spent for writing. To check your progress or counting number of words, you need to move the mouse to the bottom of the screen.

To install or update it, you can search it on the ADD/REMOVE software menu.

You can also use the following shell command to install/update it:

```
sudo apt-get install focuswriter
```

Once installed, to get started with FocusEriter, you need to program's entry in the office category of the Application menu.

Mathematica

Mathematica, pre-installed on Raspbian, is a symbolic package or Computer Algebra System (CAS). In Mathematica, you can do anything with numbers, complex multidimensional graphics, and music.

As it is preinstalled, to get started with it, you need to click the Mathematica icon under the Programming category of Application menu.

You can expand equations as well as plot graphs with Mathematica.

RealVNC

RealVNC, a remote access server and viewer software was included in Raspbian on 28th September 2016. With the help of RealVNC's new capture technology, you can directly render content. It can also be used to view non-X11 applications and to control them remotely.

Steam Link

Steam Link which can be implemented as both hardware and software solutions, supports the streaming of the Steam content from a PC to a mobile device or other monitor.

In 2015, it was originally released as a hardware device but on 13 December 2018, its developer Valve released the official Steam Link game streaming client for Raspberry Pi microcomputer (Raspberry Pi3 and Pi 3 B+).

XInvaders 3D

XInvaders 3D is a game like classic Arcade Cabinates. Similar to another classic game Asteroids, this game uses the line graphics to put a fresh spin on Space Invaders.

The three-dimensional rendering makes the aliens move progressively closer to you. To line up your shots, you need to move in four directions. The cursor keys are used to move in four directions, and you can use a spacebar to fire the shots.

To install or update it, you can search it on the ADD/REMOVE software menu.

You can also use the following shell command to install/update it:

```
sudo apt-get install xinv3d
```

Once installed, to get started with XInvaders 3D, you need to click the icon on the taskbar to go into the Terminal and then enter xinv3d.

Tux Paint

Tux Paint is a simple drawing program for kids. The tools in Tux Paint help children to quickly create art on Raspberry Pi computer. It also enables freehand drawing and the placement of shapes.

The magic tool of Tux Paint can be used to create effects such as brick walls, flowers, rainbows, waves, and various other creative image distortions. It also has a stamp tool, which is used to stamp clip art onto the screen. The stamp tool includes animals, penguins, hats, food as well as musical instruments.

The name of Tux Paint is a tribute to the penguin Tux, the official mascot of the Linux kernel.

To install or update it, you can search it on the ADD/REMOVE software menu.

You can also use the following shell command to install/update it:

```
sudo apt-get install tuxpaint
```

Once installed, to get started with Tux Paint, you need to click the icon on the Education category of your application menu.

Sense HAT Emulator

In the previous chapter, we have discussed the Sense HAT board. Raspbian OS provides us with an emulator to use Sense HAT. You can get it in the Programming section of the Application menu.

As the name suggests, the Sense HAT emulator enables us to simulate the I/O of the Sense HAT, so that we can test how our programs work. It also provides us several sample programs to get started with.

Check the diagram of Sense HAT emulator below:

The source of the above image is www.raspberrypi.org

Brain Party

Brain Party, a series of fun minigames, is designed to tune up your brain between programming sessions. To get your "brain weight" score, you need to complete five randomly selected tests. The puzzles in Brain Party game will challenge your memory, logical skills, as well as observational skills.

To install or update it, you can search it on the ADD/REMOVE software menu.

You can also use the following shell command to install/update it:

```
sudo apt-get install brainparty
```

Once installed, to get started with Brain Party, you can find it under Games of the Application menu. You can also get it by typing **brainparty** in the command line.

Grisbi

Grispi is a free application with the help of which you can keep track of your regular and one-off payments. It is mainly used to manage your home accounts on your Raspberry Pi computer. The format in which most of the banks enable us to download bank statements, can be easily used in Grisbi.

To install it or update it, you can search it on the ADD/REMOVE software menu.

You can also use the following shell command to install/update it:

```
sudo apt-get install grisbi
```

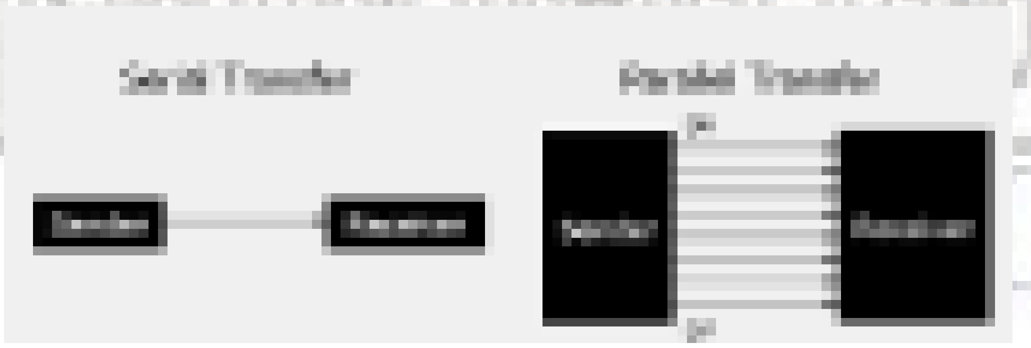
Once installed, to get started with Brain Party, you can find it in the Office Category of the Application menu.

8051 Serial Communication

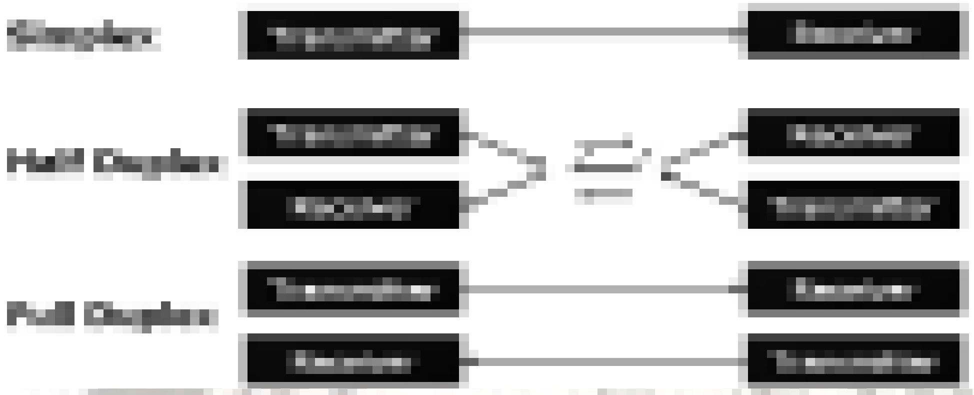
When a microcontroller communicates with the outside world, it provides the data in byte-sized chunks. In some cases, such as printers, the information is simply taken from the 8-bit data bus and presented to the 8-bit data bus of the printer. This can work only if the cable is not too long, since long cables diminish and even distort signals. Furthermore, an 8-bit data path is expensive. For these reasons, serial communication is used for transferring data between two systems located at distances of hundreds of feet to millions of miles apart. Fig(1) shows serial versus parallel data transfers. The fact that serial communication uses a single data line instead of the 8-bit data line of parallel communication. For serial data communication to work, the byte of data must be converted to serial bits using a parallel-in-serial-out shift register, then it can be transmitted over a single data line. At the receiving end there must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte.

When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation. However, for long-distance data transfers, serial data communication requires a modem to modulate (convert to 0s and 1s to audio tones) and demodulate (converting from audio tones to 0s and 1s).

Serial data communication uses two methods, asynchronous and synchronous. The synchronous method transfers a block of data(characters) at a time, while the asynchronous method transfers a single byte at a time. There are special IC's made by many manufacturers for serial data communications. These chips are commonly referred to as UART (universal asynchronous receiver transmitter) and USART (universal synchronous asynchronous receiver transmitter).



Fig(1)



In data transmission if the data can be transmitted and received, it is a duplex transmission. This is in contrast to simplex transmission such as with printers, in which the computer only sends data. Duplex transmissions can be half o full duplex, depending on whether or not the data transfer can be simultaneous. If data is transmitted one way at a time, it is referred to as half duplex. If the data can go both ways at the same time, it is full duplex. Full duplex requires two wire conductors for the data lines, one for transmission and one for reception, in order to transfer and receive data simultaneously.

Asynchronous Serial Communication.

The data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s, it is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a protocol, on how the data is packed, how many bits constitute a character, and when the data begins and ends.

Asynchronous serial data communication is widely used for character oriented transmissions. In this method, each character is placed between start and stop bits. This is called framing. In data framing for asynchronous communications, the data such as ASCII characters, are packed between a start bit and stop bit. The start bit is always one bit, but the stop bit can be one or two bits. The start bit is always a 0 (low) and the stop bit is 1(high). Look at below figure in which the ASCII character “A” (8-bit binary 0100 0001) is framed between the start bit and a stop bit. Notice that the LSB is sent out first.

Figure(2)

Data Transfer Rate

The rate of data transfer in serial data communication is stated in bps (bits per second). Another widely used terminology for bps is baud rate. The baud rate is the modem terminology and s defined as the number of signal changes per second.

RS 232 standards

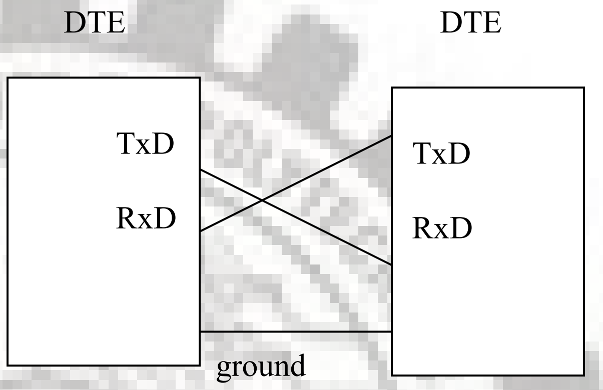
RS 232 is an interfacing standard and is the most widely used serial I/O interfacing standard. Since the standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible. For this reason, to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic levels to RS232 voltage levels, and vice versa. MAX232 IC's are commonly referred to as line drivers.

RS232 pins

RS232 cable commonly referred to as the DB-25 connector has 25 pins. Since not all the pins are used in PC cables, IBM introduced the DB-9 version of the serial I/O standard, which uses 9 pins only, as shown in Table 1. The DB-9 pins are shown in Fig(3).
Current terminology classifies data communication equipment as DTE (data terminal equipment) or DCE (data communication equipment). DTE refers to terminals and computers that send and receive data, while DCE refers t communication equipment, such as modems, that responsible

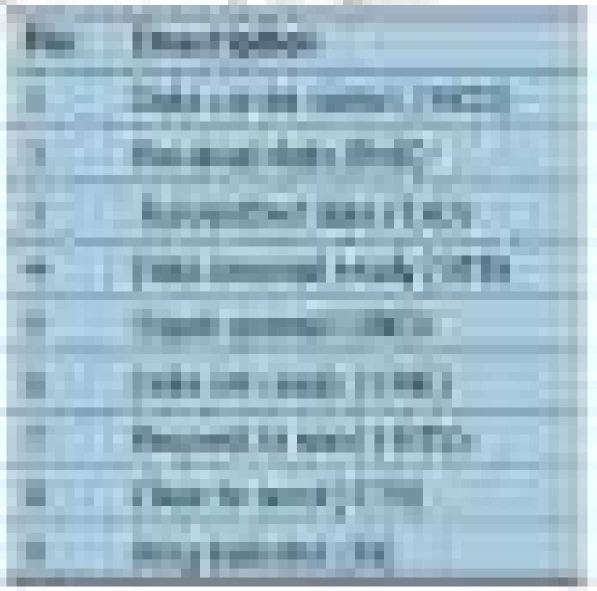
for transferring the data. The simplest connection between a PC and microcontroller requires a minimum of three pins, TxD, RxD and ground as shown in fig(4).

Fig (3)



Fig(4)

RS232 handshaking signals



Pin	Description
1	Data carrier detect (DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (DSR)
7	Request to send (RTS)
8	Clear to send (CTS)
9	Ring indicator (RI)

To ensure fast and reliable data transmission between two devices, the data transfer must be coordinated. Many of the pins of the RS232 connecter are used for handshaking signals. Their descriptions are given below.

DTR(Data Terminal Ready): When a terminal is turned on, after going through a self-test, it sends out signal DTR to indicate that it is ready for communication.

DSR (Data Set Ready): When DCE (modem) is turned on and has gone through the self-test, it asserts DSR to indicate that it is ready to communicate.

RTS (Request To Send): When the DTE device (such as PC) has a byte to transmit, it asserts RTS to signal the modem that it has a byte of data to transmit. RTS is an active-low output from the DTE and an input to the modem.

CTS (Clear To Send): In response to RTS, when the modem has room for storing the data it is to receive, it sends out signal CTS to the DTE to indicate that it can receive the data now.

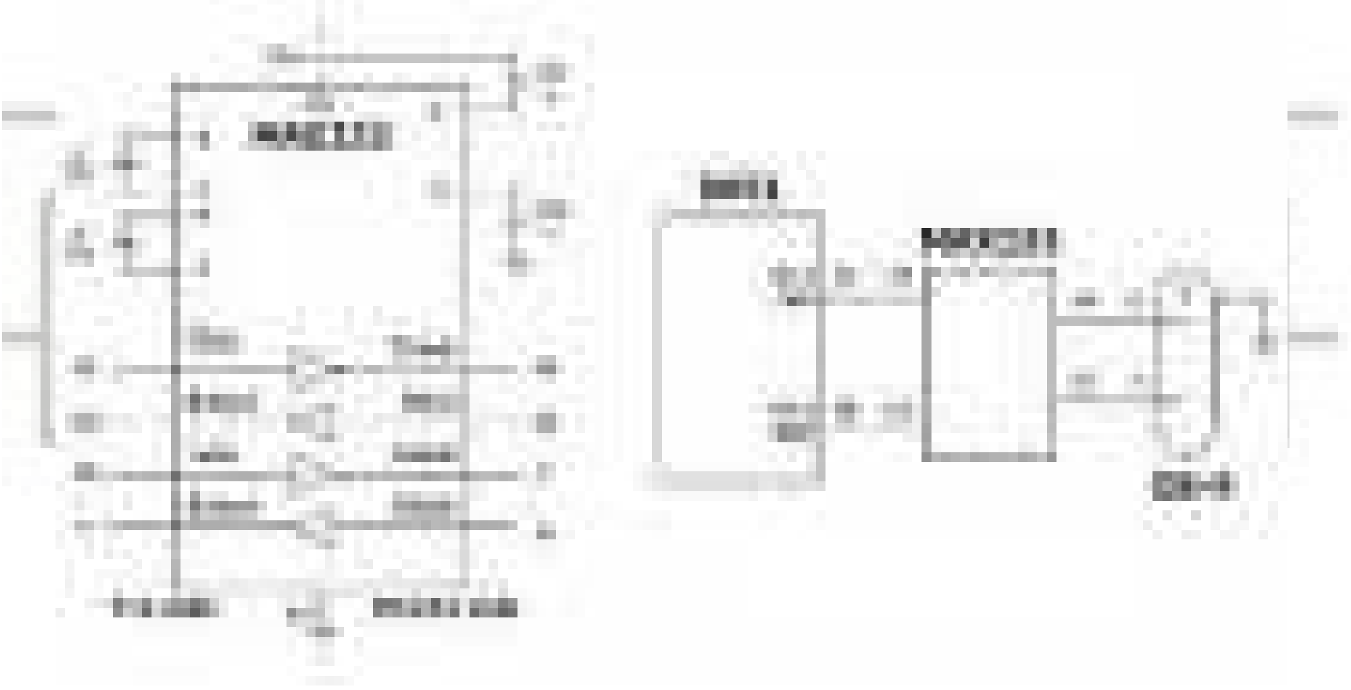
DCD (Data Carrier Detect): The modem asserts signal DCD to inform the DTE that a valid carrier has been detected and that contact between it and the other modem is established. Therefore , DCD is an output from the modem and an input to the PC.

RI (Ring Indicator): It indicates that the telephone is ringing. However if the PC is in charge of answering the phone, this signal can be used.

MAX232

Since the RS232 is not compatible with today's microprocessors and microcontrollers, we need a voltage converter(line driver) to convert the RS232's signals to TTL voltage levels that will be acceptable to the 8051's TxD and RxD pins. One example of such converter is MAX232. It converts from RS232 voltage levels to TTL voltage levels and vice versa. One advantage of the MAX232 is that it uses a +5 V power source which is the same as the source voltage for the 8051.

The MAX232 has two sets of line drivers for transferring and receiving data as shown in fig(5). The line drivers used for TxD are called T1 and T2, while the line drivers for RxD are designated as R1 and R2. MAX232 requires four capacitors ranging from 1 to 22 µF. The most widely used value for these capacitors is 22 µF.



Fig(5)

8051 SERIAL PORT PROGRAMMING

Baud rate in the 8051

The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is programmable. This is done with the help of Timer 1. The relationship between the crystal frequency and the baud rate in the 8051 is shown below.

The 8051 divides the crystal frequency by 12 to get the machine cycle frequency. In the case XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 KHz (11.0592MHz/12 = 921.6 KHz). The 8051's serial communication UART circuitry divides the machine cycle frequency of 921.6kHz by 32 once more before it is used by Timer 1 to set the baud rate. Therefore, 921.6 kHz divided by 32 gives 28,800 Hz.

SBUF REGISTER

SBUF is an 8-bit register used solely for serial communication in the 8051. For a byte of data to be transferred via the TxD line, it must be placed in the SBUF register. Similarly, SBUF holds the byte of data when it is received by the RxD line. SBUF can be accessed like any other register in the 8051.

SCON (Serial Control) REGISTER

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

Programming the 8051 to transfer data serially

In programming the 8051 to transfer character bytes serially, the following steps must be taken.

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 to set the baud rate.
2. The TH1 is loaded with one of the values in Table (2) to set the baud rate for serial data transfer.
3. The SCON register is loaded wit the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
4. TR1 is set to 1 to start Timer 1.

5. TI is cleared by the “CLR TI” instruction .
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag bit is monitored with the use of the instruction “JNB TI, xx” to see if the character has been transferred completely.
8. To transfer the next character, go to Step 5.

[illegible]

Programming the 8051 to receive data serially

In programming the 8051 to transfer character bytes serially, the following steps must be taken.

1. The TMOD register is loaded with the value 20H, indicating the use of Timer 1 in mode 2 to set the baud rate.
2. TH1 is loaded with one of the values in Table(2) to set the baud rate.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where 8-bit data is framed with start and stop bits and receive enable is turned on.
4. TR1 is set to 1 to start Timer 1.
5. RI is cleared with the “CLR RI” instruction.
6. The RI flag bit is monitored with the use of the instruction “JNB RI, xx” to see if an entire character has been received yet.
7. When RI is raised, SBUF has the byte. Its contents are moved into safe place.
8. To receive the next character, go to Step 5.

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

Solution:

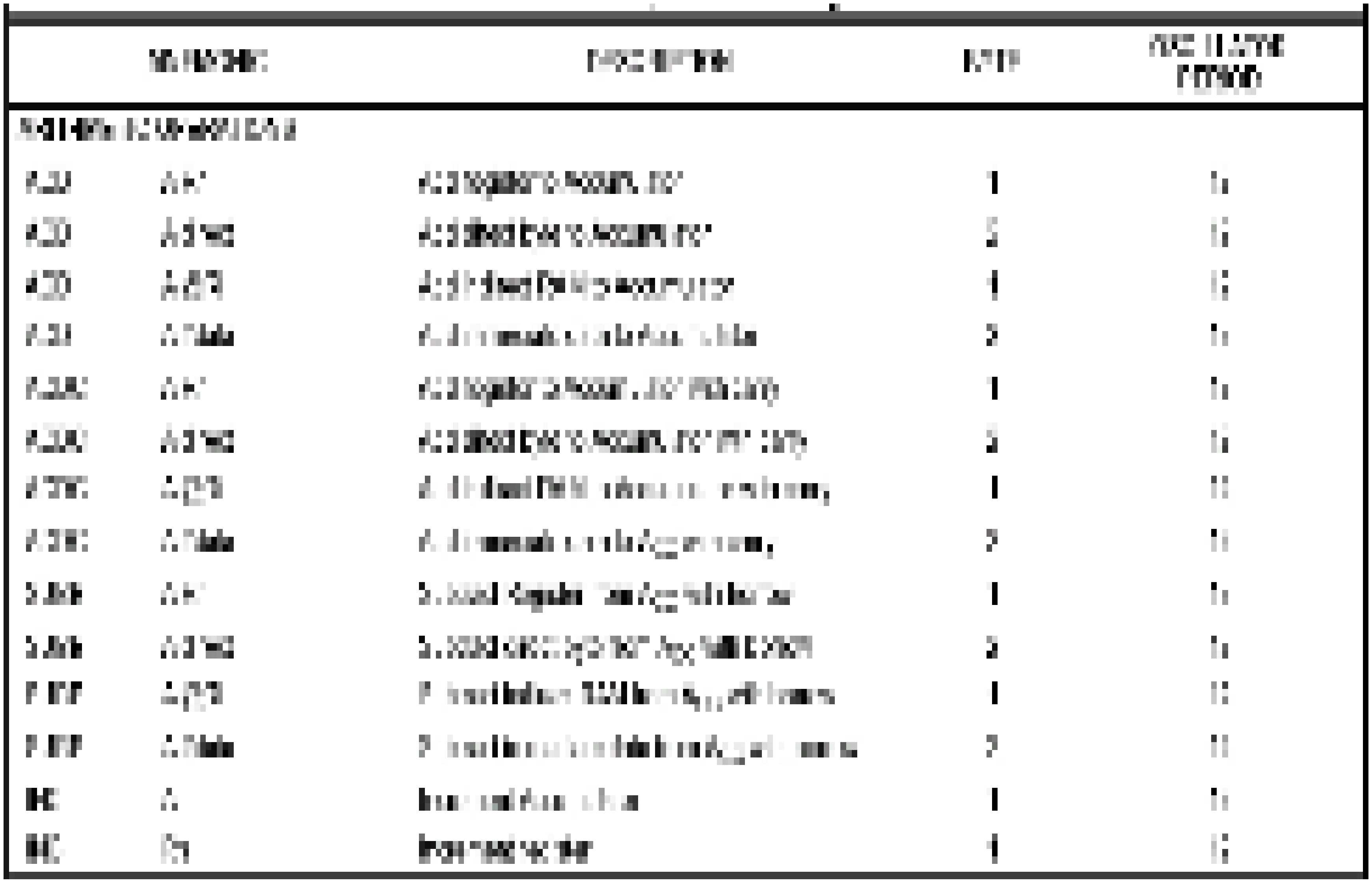
```
        MOV  TMOD,#20H    ;timer 1,mode 2(auto reload)
        MOV  TH1,#-6      ;4800 baud rate
        MOV  SCON,#50H    ;8-bit, 1 stop, REN enabled
        SETB TR1          ;start timer 1
HERE:   JNB  RI,HERE      ;wait for char to come in
        MOV  A,SBUF       ;saving incoming byte in A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

8051 Programming

- The 8051 may be programmed using a low-level or a high-level programming language.

- Low-Level Programming
  - Assembly language programming writes statements that the microcontroller directly executes
  - Advantages
    - 8051 assemblers are free
    - Produces the fastest and most compact code
  - Disadvantages
    - Difficult to learn (8051 assembler has 111 instructions)
    - Slow to program
    - Not portable to other microcontrollers

Assembly Language Instruction Set



MNEMONIC		DESCRIPTION	BYTE	OSCILLATOR PERIOD
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with carry	1	12
ADDC	A,#data	Add immediate data to Acc with carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12

Source Philips 80C51 Family Programmer's Guide and Instruction Set

8051 Programming

- High-Level Programming
  - Uses a general purpose programming language such as C
  - Advantages
    - Easier to learn
    - Faster to program
    - More portable than assembly language
  - Disadvantages
    - Code may not be as compact or as fast as assembly language
    - Good quality compilers are expensive

8051 Programming Examples

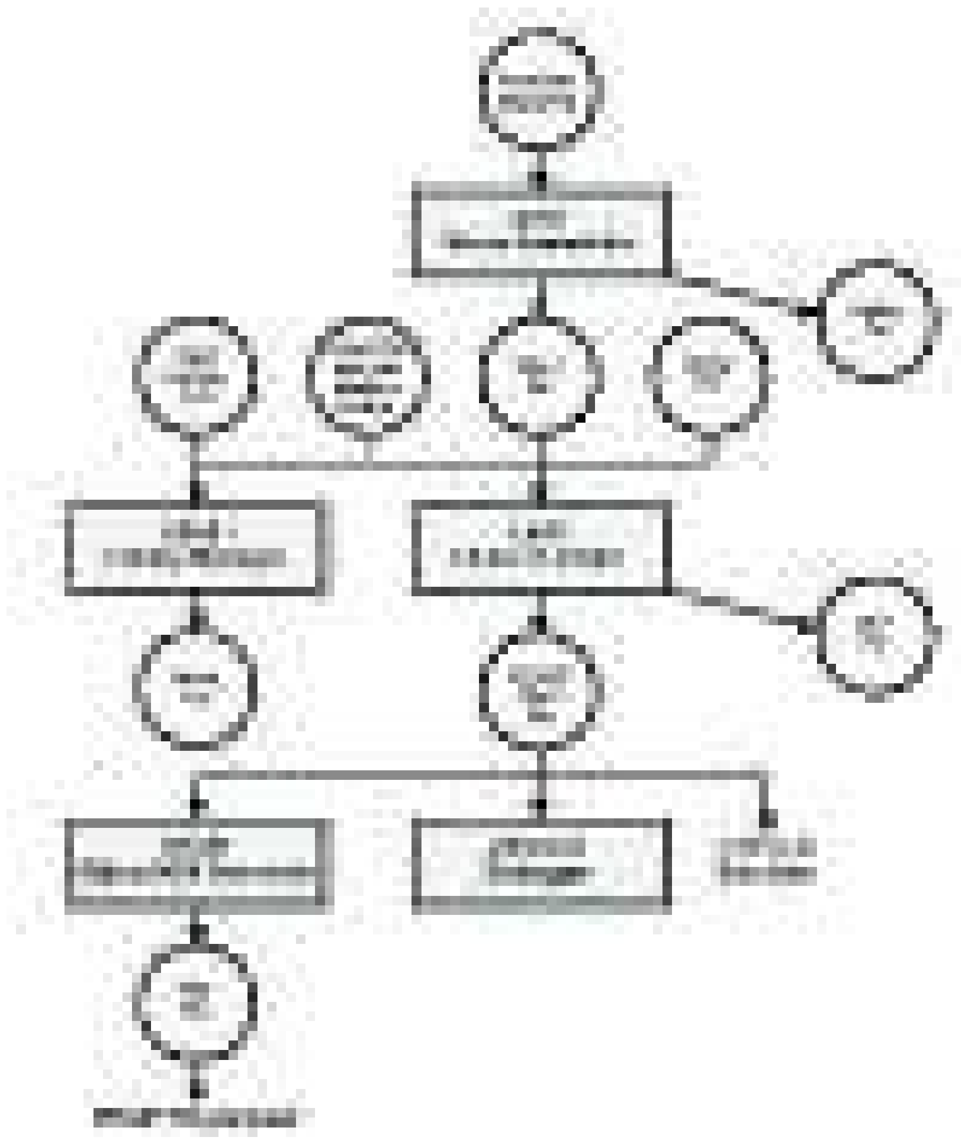
- C program example to add 2 numbers

```
void main()
{
        unsigned char x=5,y=6,z;
        z = x + y;
}
```

- Same code written using assembly language

```
MOV    A,#05H
ADD    A,#06H
MOV    R0,A                ;result stored in R0
```

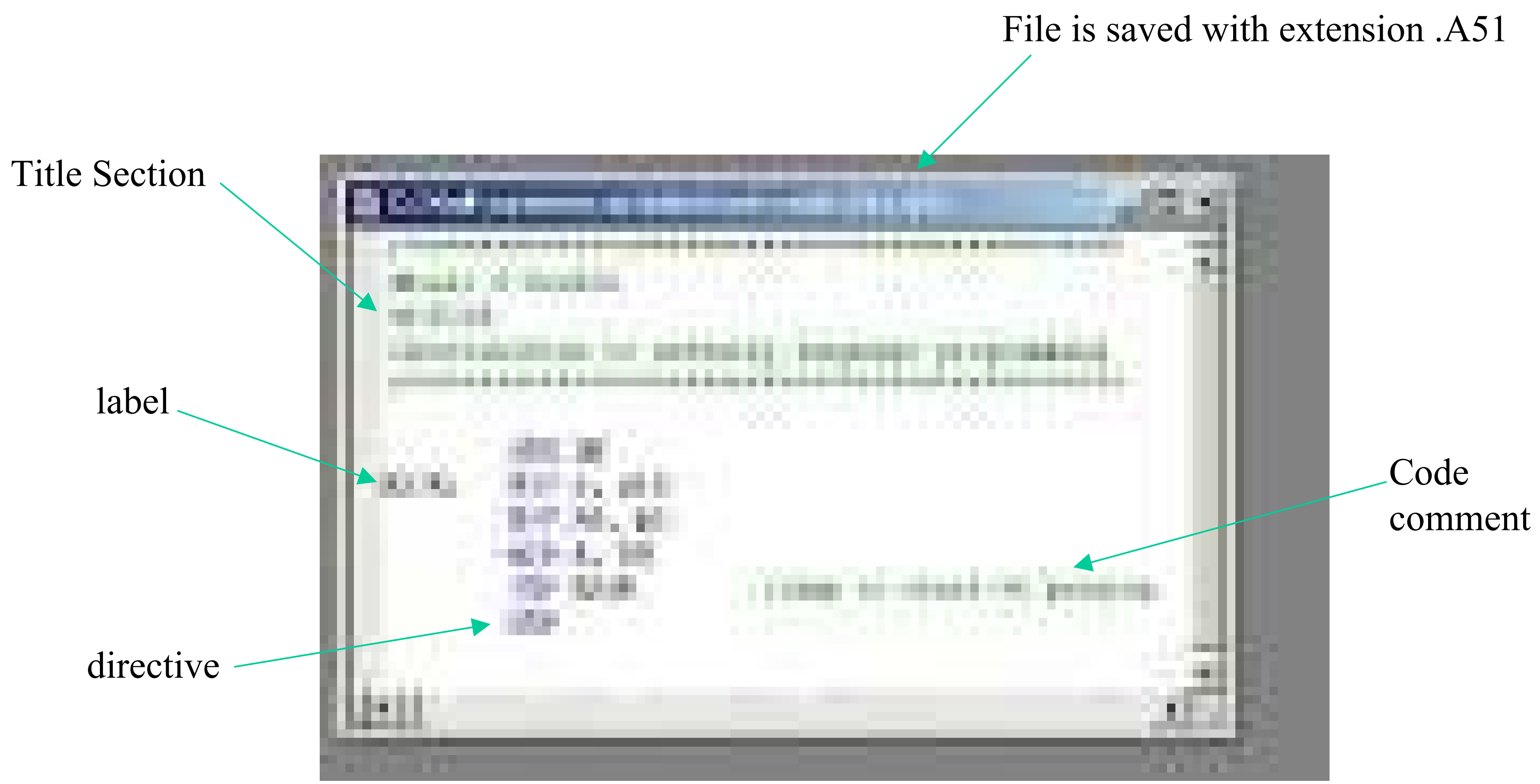
Assembly Language Development Cycle



Rules/Syntax

- All code is normally typed in upper case
- All comments are typed in lower case
  - All comments must be preceded with a semicolon
- All symbols and labels must begin with a letter
- All labels must be followed by a colon
  - Labels must be the first field in a line of assembler code
- The last line of any program must be the END directive

Assembly Programme Example



[illegible]

Listing File Produced by Assembler

8051 Assembly Language

- An assembler program is made up of 3 elements
  - Instructions
  - Assembler Directives
    - Instructions used by the assembler to generate an object file
    - The instructions are not translated to machine code
    - e.g. ORG, END
  - Assembler Controls
    - Define the mode of the assembler
    - e.g. produce a listing file

8051 Instruction Set

The 8051 instruction set can be divided into 5 subgroups: -

- Data Transfer
  - MOV instructions used to transfer data internal and external to the 8051
- Arithmetic
  - Add, subtract, multiply, divide
- Logical
  - AND, OR, XOR, NOT and rotate operations
- Boolean variable manipulation
  - Operations on bit variables
- Program Branching
  - Conditional and unconditional jump instructions

8051 Instruction Set

8051 assembly code contains the following fields:-

<label:> MNEMONIC <DESTINATION>, <SOURCE> <;comment>

- The label and comment fields are optional.
- The mnemonic is the assembler instruction e.g. MOV, ADD
- The destination and source fields are optional
  - It is important to remember that the destination comes first

- The 8051 uses 4 addressing modes: -
  - Immediate Addressing
  - Register Addressing
  - Direct Addressing
  - Register Indirect Addressing

Immediate Addressing

- In immediate addressing the data source is always a number and is specified by a '#'.
  - The number specified is copied into the destination

```
MOV A, #10          ;moves number 10 into Accumulator
MOV R0, #0AH        ;moves number 10 into R0
```

- Assembler Number Representation
  - Default numbering system is decimal
  - Hexadecimal numbers must be followed by the letter H and must begin with a number i.e. the number FA Hex is written as 0FAH.
  - Binary numbers must be followed by the letter B.

  - The following instructions have the same effect

    MOV R0, #255

    MOV R0, #0FFH

    MOV R0, #11111111B

Register Addressing

- Internal registers A, R0 to R7 and DPTR may be used as the source or the destination.

```
MOV A, R0    ;copies contents of R0 to A
```

- Note: - Data may not be copied from Rn to Rn
  - MOV R0, R1 will generate an assembler error
- The source remains unchanged.

Direct Addressing

- Direct Addressing is used in instructions that affect internal data memory locations or the SFR's.
  - The internal data memory address range is 0 to 127 (0 to 7FH)

```
MOV A, 20H          ;copies contents of address 20H into the Accumulator

MOV 30H, 40H        ;copies contents of address 40H to address 30H

MOV P1, A           ;move the contents of the Accumulator to Port 1
```

Indirect Addressing

- The most powerful addressing mode.
- A register is used to store the address of the destination or source of data
  - Similar to the use of pointers in high level languages
  - The @ symbol is used before the register to specify indirect addressing
  - SFRs may not be indirectly addressed
  - Internal data memory may be directly or indirectly addressed

```
MOV R0, #20H        ;Load R0 with the number 20H
MOV @R0, #55H       ;Move 55H to the address contained in R0 (20H)
                    ;R0 acts as a pointer to address 20H
MOV A, @R0          ;Copy the contents of address 20H to the Accumulator
```

- Only registers R0 and R1 may be used for moving data to/from internal data memory when using indirect addressing
- Registers R0, R1 and DPTR may be used when indirectly addressing external memory (more later)

Addressing Modes Exercise

- What are the contents of registers A, R0, R7 and memory locations 30H and 31H after the following code runs: -

```
MOV A, #5
MOV R7, #40H
MOV R0, #30H
MOV 31H, #14H
MOV @RO, A
INC R0
MOV R7, @R0
```

- How long does the code take to execute if the 8051 is operating off a 12MHz crystal?

Some Useful Directives

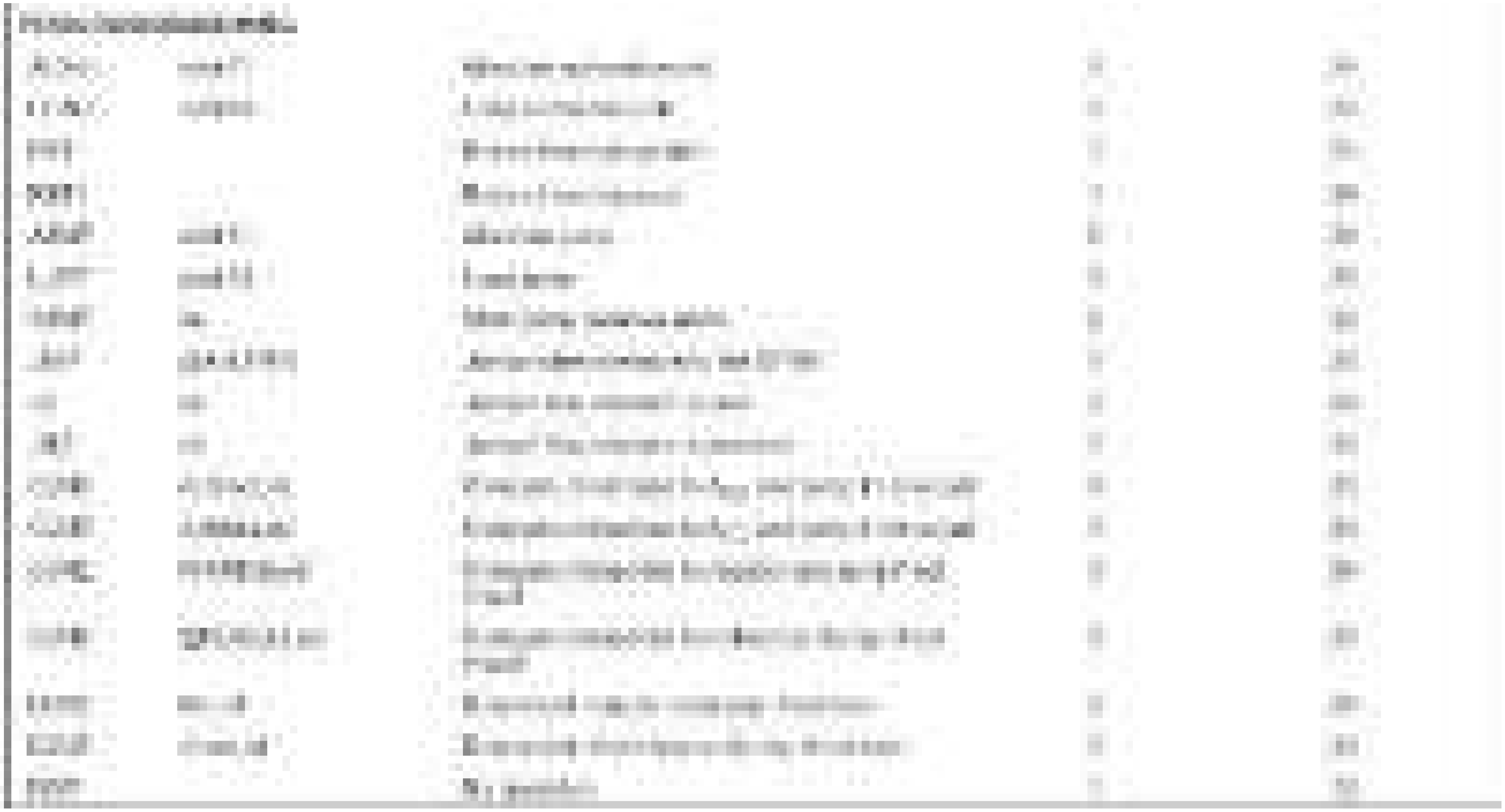
- END
  - Last line of code. Assembler will not compile after this line
- ORG
  - Origin directive. Sets the location counter address for the following instructions
- EQU
  - Equate directive. Used to equate a name with an address or a data value. Useful for constant assignments.
- DATA
  - Used to assign a symbol name to an address in internal data memory

    ```
    AVERAGE DATA 30H
    MOV AVERAGE, A
    ```
- BIT
  - Used to assign a symbol name to a bit in bit-addressable data memory

Programme Sequencing

- Normal program execution is sequential
  - The PC is loaded with the address of instruction N+1 while instruction N is being executed
- The program branching instructions allow the programmer to alter the program execution sequence
  - These instructions allow the address contained in the PC to be changed
  - Program branching is used for jumps, function calls and interrupt service routines.

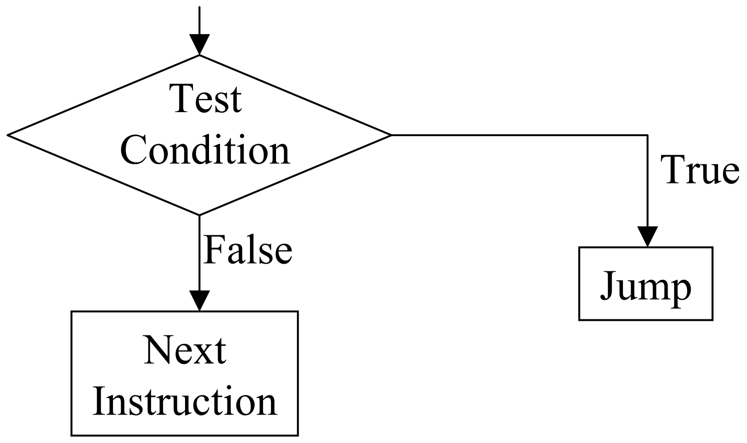
Program Branching Instructions



PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump Indirect Relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

Jump Instructions

- The 8051 has 2 types of JUMP instructions
- Unconditional Jump
  - This instruction type will load the PC with a new address and will automatically jump to the instruction at that address
- Conditional Jump
  - This instruction type will only jump if a certain condition is true
  - Similar to an “if” statement in C.



Unconditional Jumps

The 8051 has 3 unconditional jump instructions with a different range: -

- SJMP (Short Jump)
  - Allows a jump of –128 to +127 bytes relative to the current PC value
  - Instruction is 2 bytes long
- AJMP (Absolute Jump)
  - Allows a jump with the same 2KByte page that the PC is currently located in
  - Instruction is 2 bytes long
- LJMP (Long Jump)
  - Allows a jump anywhere within the 64KByte program memory range of the 8051
- If unsure which of the 3 instructions to use, simply use the JMP instruction and let the assembler decide which instruction to use.

Conditional Jumps

- The 8051 can test conditions at the bit and byte level
- Bit Conditional Jump Instructions
  - These instructions will jump if a bit is in a certain state
  - e.g. JC label ;jump to label if carry bit is set
  - JNC label2 ;jump to label2 if the carry bit is clear
  - These instructions are commonly used for arithmetic instructions and for the testing of flags
- Byte Conditional Jump Instructions
  - DJNZ – Decrement and Jump if Not Zero
  - CJNE – Compare and Jump if Not Equal

DJNZ Instruction

- Decrement and Jump if Not Zero
  - DJNZ Rn, label
  - DJNZ direct address, label

- DJNZ is used to execute a block of code N times
  - Similar to a for or while loop in C
  - Very useful for generating delays

```
        MOV R0, #10        ;R0 = loop counter
LOOP:   DJNZ R0, LOOP      ;DJNZ instruction executed 10 times
        MOV A, R1
```

DJNZ for Generating Delays

```
        MOV R0, #10        ;R0 = loop counter
LOOP:   DJNZ R0, LOOP      ;DJNZ instruction executed 10 times
        MOV A, R1
```

- The DJNZ instruction takes 2 machine cycles to execute (24 clocks)
- If the 8051 is operating from a 12MHz crystal, the loop execution time is
  (10 * 24)/12000000 = 20usec

- The maximum delay for a single loop occurs when the loop counter is initialised to 0
  - This will cause 256 loop iterations
  - Delay = (256 * 24)/12000000 = 512usec

- How do we generate delays longer than 512usec?

DJNZ for Generating Delays

- Longer delays may be generated by using nested DJNZ instructions

```
        MOV R0, #0        ;12 clocks
        MOV R1, #200      ;12 clocks
LOOP:   DJNZ R0, LOOP     ;256 * 24 clocks
        DJNZ R1, LOOP     ;executes inner loop + DJNZ 200 times
```

- *Execution time is (12 + 12 + 200((256*24) + 24))/12000000 = 0.102802 sec*

- *Rewrite the code to generate a delay of 0.1usec accurate to 10usec*

DJNZ Exercise

```
        MOV R0, #0
        MOV R1, #0
        MOV R2, #10
LOOP:   DJNZ R0, LOOP
        DJNZ R1, LOOP
        DJNZ R2, LOOP
```

1. *How long does the above code take to execute if the 8051 is operating off a 12MHz crystal?*
2. *Repeat part 1 for a 16MHz crystal*
3. *Rewrite the code to generate a delay of 1 second accurate to 10usec (assume a 12MHz crystal)*

CJNE Instruction

- Compare and Jump if Not Equal to Zero

  CJNE destination, source, label

- The destination and source bytes are compared and a jump takes place if they are not equal.
  - The carry flag is set if the destination byte is less than the source byte
- Often used to validate characters received via the serial port
- May be used for delays but code is not as efficient as DJNZ

```
        MOV R0, #10
LOOP:   CJNE R0, #0, LOOP1
        JMP DONE
LOOP1:  DEC R0
        JMP LOOP
DONE:
```

Subroutines

- A subroutine is a block of code that can be used many times in the execution of a larger program (similar to functions in higher level languages)
- Subroutines allow the program to branch to a section of code and to remember where it branched from.
- When the subroutine is complete program execution will continue from the line of code following the subroutine call.

- Subroutines have the following advantages: -
  - Code savings
    - The same subroutine may be called over and over again
  - Program structuring
    - A large program may be divided into a number of small subroutines
    - This makes the program easer to maintain and debug

Subroutine Example

```
        ORG 0000H
MAIN:   ……….
        ……….
        CALL SUB1        ;subroutine call (store PC on stack)
        CALL SUB2
        ……….
        JMP MAIN

SUB1:   ……….             ;label defines start of subroutine
        ……….
        RET              ;return to line after CALL

SUB2:   ……….
        ……….
        RET
        END
```

Subroutines

- A subroutine is always executed with the CALL instruction
  - When the CALL instruction is executed the PC register contains the address of the next instruction to be executed (this is known as the return address)
  - The PC is saved onto the stack low byte first
  - The PC is then loaded with the address of the subroutine
  - The subroutine is then executed

- The last line of a subroutine is always the RET instruction
  - The RET instruction will cause the return address to be popped off the stack and loaded into the PC
  - The instruction at the return address is then executed

Subroutine Parameter Passing

1. Place the parameter into an address in internal data memory
   Most popular method used

2. Push the parameter onto the stack
   This method is limited by the size of the stack

3. Place the parameter into external data memory
   Used when internal data memory has been used up.
   Slower execution speed than when using internal data memory

Subroutine Call With No Parameter Passing

```
;code to output a waveform on P1.0 that is high for 5 seconds and low for 2.5 seconds
          ORG 0
MAIN:     CALL ON
          CALL OFF
          JMP MAIN

ON:       SETB P1.0
          MOV R0, #40
          MOV R1, #0
          MOV R2, #0
DELAY1:   DJNZ R2, DELAY1
          DJNZ R1, DELAY1
          DJNZ R0, DELAY1
          RET

OFF:      CLR P1.0
          MOV R0, #20
          MOV R1, #0
          MOV R2, #0
DELAY2:   DJNZ R2, DELAY2
          DJNZ R1, DELAY2
          DJNZ R0, DELAY2
          RET

          END
```

Subroutine Call With Parameter Passing

```
;code to output a waveform on P1.0 that is high for 5 seconds and low for 2.5 seconds
        ORG 0
MAIN:   CALL ON
        CALL OFF
        JMP MAIN

ON:     SETB P1.0
        MOV R0, #40
        CALL DELAY
        RET

OFF:    CLR P1.0
        MOV R0, #20
        CALL DELAY
        RET

DELAY:  MOV R1, #0
        MOV R2, #0
LOOP:   DJNZ R2, LOOP
        DJNZ R1, LOOP
        DJNZ R0, LOOP
        RET

        END
```

8051 Arithmetic Operations

- All arithmetic operations are carried out in the ALU
- The 8051 has 4 arithmetic flags that are stored in the PSW register

1. C Carry Flag
   Set if there is a carry out after addition or a borrow after subtraction.
   Used for unsigned arithmetic.

2. AC Auxiliary Carry Flag
   Set if there is a carry out of bit 3 during addition or a borrow during subtraction.
   Used for BCD arithmetic.

3. OV Overflow Flag
   Set if there is a carry from bit 6 XOR bit 7
   Used for signed arithmetic

4. P Parity Flag
   Contains the parity of the accumulator. 1 if odd, 0 if even. Useful for some serial port operations.

Increment/Decrement Instructions

- INC Source
  - Adds 1 to the source
- DEC Source
  - Subtract 1 from the source
- Source may be a register or a direct or indirect address
  - INC A
  - DEC R1
  - INC 30H
  - DEC @R0
- No flags are affected by the INC and DEC instructions

Multiply/Divide Instructions

- MUL AB
  - Note no comma between source and destination
  - Multiplies the A register by the B register. The low order byte of the result is placed in A, the high order byte in B.
  - The OV flag is set if the result exceeds 255

- DIV AB
  - Divides A register by B register.
  - The integer part of the quotient is placed in A, the integer part of the remainder is placed in B.
  - OV flag is set if a divide by 0 is attempted

Addition

ADD A, SOURCE ;A = A + Source

ADDC A, SOURCE ;A = A + Source + C

- The accumulator is always used to store the result
- All addressing modes may be used.

- Affect on flags
  - Carry bit C is set if there is a carry out of bit 7, cleared otherwise.
    - Used for unsigned addition
  - AC flag set if there is a carry from bit 3, cleared otherwise
    - Used for BCD addition
  - OV flag is set if C7 XOR C6, cleared otherwise
    - Used for signed addition

Unsigned Addition

- The carry bit C should always be tested after an addition to see if the result has exceeded 255 (FFH).
  - *JC Label ;jump if carry bit is set*
  - *JNC Label ;jump if carry bit is clear*

Examples: -

```
25          00011001
47          00101111
--------------------
72          01001000     No carry (result <=255)

65            01000001
208           11010000
----------------------
273         1 00010001   Carry (result > 255)
```

Unsigned Addition

- Write a program to add the contents of internal data memory locations 30H and 31H If a carry occurs, set the pin P1.0

```
        ERROR BIT P1.0
MAIN:   CLR ERROR
LOOP:   MOV A, 30H
        ADD A, 31H
        JNC MAIN            ;no carry
        SETB ERROR          ;carry, set error pin
        JMP LOOP
        END
```

Signed Addition

- For signed arithmetic bit 7 of the number is used as a sign bit.
  - 1 for a negative number and 0 for a positive number
  - Number range is restricted to –128 to +127
- The OV flag should always be tested after adding 2 signed numbers
  - The OV flag will only change when adding numbers of the same sign yields a result outside of the range –128 to +127

Examples: -

+25	00011001	
-45	11010011	
-20	11101100	;OV = 0, take no action

+120	01111000	
+48	00110000	
+168	10101000	;OV = 1, result is –88? Need to adjust result

Signed Addition

```
+120      01111000
+48       00110000
--------------------
+168      10101000          ;OV = 1, result is –88? Need to adjust result
```

- How do we adjust the result to get the correct value (+168)?
  - Remember that the result range is –128 to +127
  - If there is an overflow this range has been exceeded
  - Invert bit 7 to get the correct polarity for the result
  - The result then needs to be adjusted by +/-128 depending on whether we are adding positive or negative numbers

- For the above example, inverting the result bit 7 yields 00101000 (+40)
  - Because of the overflow the real result is 40 +128 = 168.

Adding 2-Byte Numbers

- Write a program to add 2 integers.
  - Integer 1 is stored at addresses 30H and 31H (low byte at address 30H)
  - Integer 2 is stored at addresses 32H and 33H (low byte at address 32H)
  - The result should be stored at addresses 34H to 36H (low byte at address 34H)

- Hint
  - Use the ADDC instruction instead of ADD

BCD Addition

- The 8051 performs addition in pure binary – this may lead to errors when performing BCD addition

Example

```
49 BCD    01001001 BCD
38  BCD   00111000 BCD
----------------------
87 BCD    10000001 (81BCD)
```

- The result must be adjusted to yield the correct BCD result
  - DA A (decimal adjust instruction)
  - The carry flag is set if the adjusted number exceeds 99 BCD

```
MOV A, #9
ADD A, #11      ;A = 1AH (expecting 20H if these are BCD numbers)
DA A            ;A = 20H
```

Subtraction

- *SUBB A, SOURCE*
  - Subtracts source and carry flags from A
  - Result placed in A

- For unsigned subtraction the carry flag C is set if there is a borrow needed for bit 7

- For signed subtraction the OV flag is set if the subtraction of a negative number from a positive number yields a negative result or if the subtraction of a positive number from a negative number yields a positive result.

8051 Logical Operations

- All 4 addressing modes may be used for the 8051 logical instructions

- AND
  - *ANL A,SOURCE*
  - May be used to selectively clear bits in the destination operand
  - e.g. *ANL A, #11111100B ;will clear lower 2 bits of A register*

- OR
  - *ORL A, SOURCE*
  - May be used to selectively set bits in the destination operand
  - *ORL A, #00000001B ;will set bit 0 of A register*

- XOR
  - *XRL A, SOURCE*
  - May be used to selectively complement bits in the destination operand
  - *XRL P1, #00001111B ;will complement lower 4 bits of port 1*

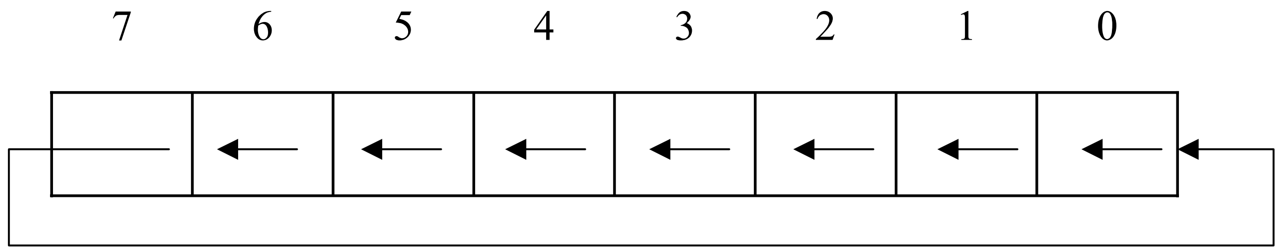
8051 Logical Operations

- Complement
  - *CPL A*
  - Complements each bit of the A register

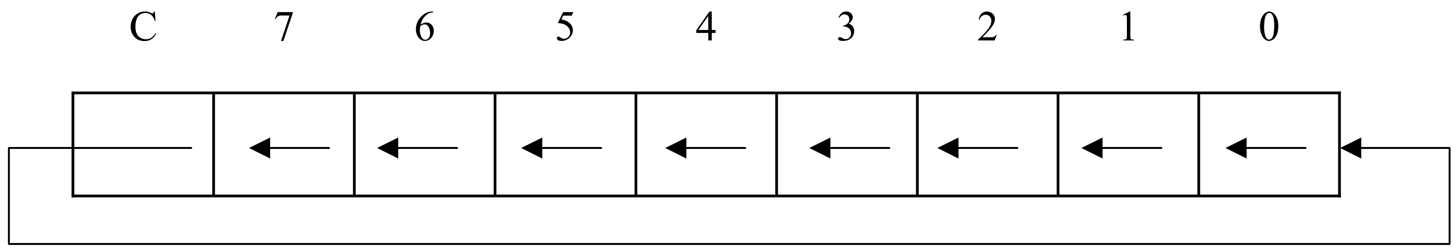
- Clear
  - *CLR A*
  - Clears each bit of the A register

Rotate Operations

- All rotate operations are carried out on the accumulator
- *RL A*
  - Rotate accumulator left, MSB becomes LSB



- *RLC A*
  - Rotate accumulator left, MSB becomes carry, carry becomes LSB



- Similar operations for rotate right: - RR A, RRC A

Bit Level Logical Operations

- These instructions allow a single bit to be altered without affecting the entire byte
  - Can be used to set/clear a bit in bit-addressable memory
  - Can be used to set/clear an I/O pin

- *SETB BIT ;sets bit high*
- *CLR BIT ;clears bit low*

- Example

```
SETB P1.0       ;P1.0 = '1'
CLR P1.0        ;P1.0 = '0'

or

LED BIT P1.0    ;use BIT directive to name pin
SETB LED
```

Bit-level Boolean Operations

[illegible]

Look-Up Tables

- A look-up table is a table of constants stored in program memory
  - Look-up tables can be used to speed up arithmetic operations
  - The look-up table may be accessed using the DPTR or PC as a pointer to the start of the table. The A register is used as an index to the table.

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

  - The look-up table is defined using the DB directive

```
ORG 200H
DB 1,2,4,9
```

  - This code will create a look-up table at address 200H. The value 1 will be stored at address 200H, 2 at address 201H etc
  - Ensure that the look-up table does not overlap with the address space used by code.

Look-up Table Example

- Write a program to read an 8-bit temperature in Celsius from Port 1 and to output the Farenheight temperature equivalent onto Port 2
  - F = ((C * 9)/5) + 32

- This temperature conversion could be coded in 2 ways: -
  - Use arithmetic operations to work out the formula
    - This would involve a multiply and a divide operation which are the 2 instructions with the longest execution time
    - The code would also have to deal with the 2-byte result of the multiply

  - Use a look-up table that stores the Farenheight equivalent of all possible Celsius readings
    - This would require more program memory – 1 byte for each temperature
    - The code is much simpler to implement

Temperature Conversion Program

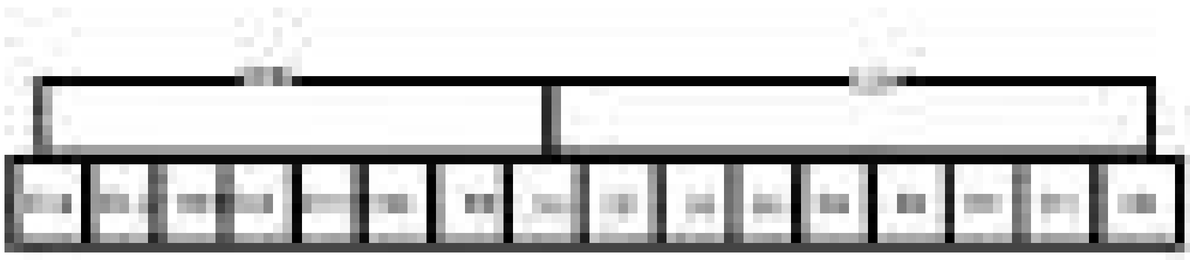
```
        TABLE EQU 100H
        ORG 0
MAIN:   MOV DPTR, #TABLE
LOOP:   MOV A, P1
        MOVC A, @A+DPTR
        MOV P2, A
        JMP LOOP

        ;conversion look-up table for 0 to 40 degrees Celsius
        ORG TABLE
        DB 32,34,36,37,39,41,43,45,46,48,50,52,54,55,57,59,61,63,64,66
        DB 70,72,72,75,77,79,81,82,84,86,88,90,91,93,95,97,99,100,102,104
        END

;What happens if a value outside of the range 0 to 40 is read from Port 1?
;How do you deal with this scenario in code?
```

8051 TIMERS AND COUNTERS

In this article, we focus on Timers/Counters of the 8051 micro controller. The 8051 has two counters/timers which can be used either as timer to generate a time delay or as counter to count events happening outside the microcontroller.

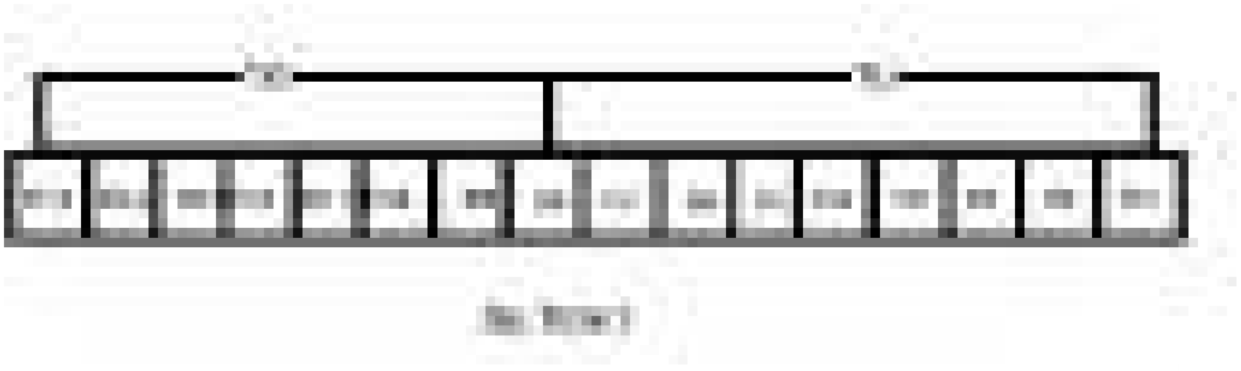
The 8051 has two timers: timer0 and timer1. They can be used either as timers or as counters. Both timers are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit is accessed as two separate registers of low byte and high byte. First we shall discuss about Timer0 registers.

Timer0 registers is a 16 bits register and accessed as low byte and high byte. The low byte is referred as a TL0 and the high byte is referred as TH0. These registers can be accessed like any other registers.

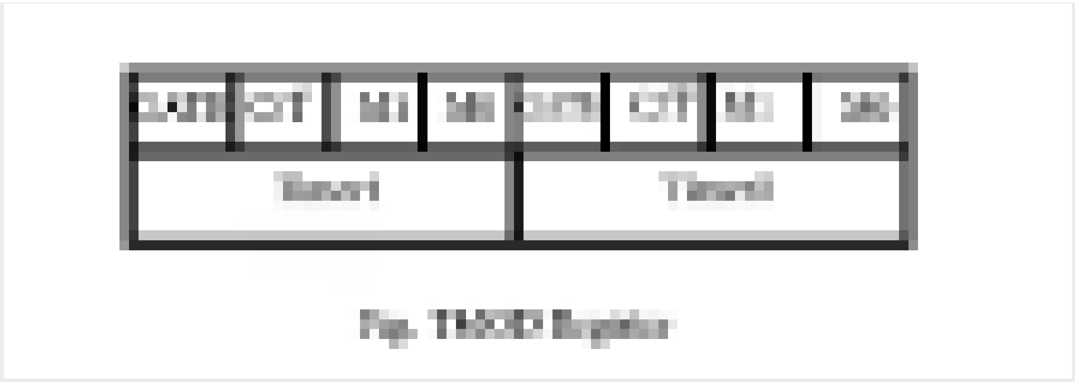


[illegible]

Timer1 registers is also a 16 bits register and is split into two bytes, referred to as TL1 and TH1.



TMOD (timer mode) Register: This is an 8-bit register which is used by both timers 0 and 1 to set the various timer modes. In this TMOD register, lower 4 bits are set aside for timer0 and the upper 4 bits are set aside for timer1. In each case, the lower 2 bits are used to set the timer mode and upper 2 bits to specify the operation.



TMOD

In upper or lower 4 bits, first bit is a GATE bit. Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register. And if we change to GATE=0 then we do no need external hardware to start and stop the timers.

The second bit is C/T bit and is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if it is 1 then it is used as a counter.
In upper or lower 4 bits, the last bits third and fourth are known as M1 and M0 respectively. These are used to select the timer mode.

M0	**M1**	**Mode**	**Operating Mode**
0	0	0	13-bit timer mode, 8-bit timer/counter THx and TLx as 5-bit prescalar.

0	1	1	16-bit timer mode, 16-bit timer/counters THx and TLx are cascaded; There are no prescalar.
1	0	2	8-bit auto reload mode, 8-bit auto reload timer/counter; THx holds a value which is to be reloaded into TLx each time it overflows.
1	1	3	Spilt timer mode.

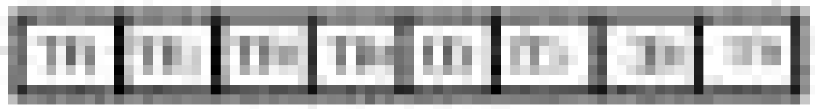
Mode 1- It is a 16-bit timer; therefore it allows values from 0000 to FFFFH to be loaded into the timer's registers TL and TH. After TH and TL are loaded with a 16-bit initial value, the timer must be started. We can do it by "SETB TR0" for timer 0 and "SETB TR1" for timer 1. After the timer is started. It starts count up until it reaches its limit of FFFFH. When it rolls over from FFFF to 0000H, it sets high a flag bit called TF (timer flag). This timer flag can be monitored. When this timer flag is raised, one option would be stop the timer with the instructions "CLR TR0" or CLR TR1 for timer 0 and timer 1 respectively. Again, it must be noted that each timer flag TF0 for timer 0 and TF1 for timer1. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value and TF must be reset to 0.

Mode0- Mode 0 is exactly same like mode 1 except that it is a 13-bit timer instead of 16-bit. The 13-bit counter can hold values between 0000 to 1FFFH in TH-TL. Therefore, when the timer reaches its maximum of 1FFH, it rolls over to 0000, and TF is raised.

Mode 2- It is an 8 bit timer that allows only values of 00 to FFH to be loaded into the timer's register TH. After TH is loaded with 8 bit value, the 8051 gives a copy of it to TL. Then the timer must be started. It is done by the instruction "SETB TR0" for timer 0 and "SETB TR1" for timer1. This is like mode 1. After timer is started, it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00. It sets high the TF (timer flag). If we are using timer 0, TF0 goes high; if using TF1 then TF1 is raised. When Tl register rolls from FFH to 00 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 auto reload, in contrast in mode 1 in which programmer has to reload TH and TL.

Mode3- Mode 3 is also known as a split timer mode. Timer 0 and 1 may be programmed to be in mode 0, 1 and 2 independently of similar mode for other timer. This is not true for mode 3; timers do not operate independently if mode 3 is chosen for timer 0. Placing timer 1 in mode 3 causes it to stop counting; the control bit TR1 and the timer 1 flag TF1 are then used by timer0.

TCON register- Bits and symbol and functions of every bits of TCON are as follows:



[illegible]

BIT	Symbol	Functions
7	TF1	Timer1 over flow flag. Set when timer rolls from all 1s to 0. Cleared When the processor vectors to execute interrupt service routine Located at program address 001Bh.
6	TR1	Timer 1 run control bit. Set to 1 by programmer to enable timer to count; Cleared to 0 by program to halt timer.
5	TF0	Timer 0 over flow flag. Same as TF1.
4	TR0	Timer 0 run control bit. Same as TR1.
3	IE1	External interrupt 1 Edge flag. Not related to timer operations.
2	IT1	External interrupt1 signal type control bit. Set to 1 by program to Enable external interrupt 1 to be triggered by a falling edge signal. Set To 0 by program to enable a low level signal on external interrupt1 to generate an interrupt.
1	IE0	External interrupt 0 Edge flag. Not related to timer operations.
0	IT0	External interrupt 0 signal type control bit. Same as IT0.

Source : http://www.circuitstoday.com/8051-timers-counters

Programmable Peripheral Interface

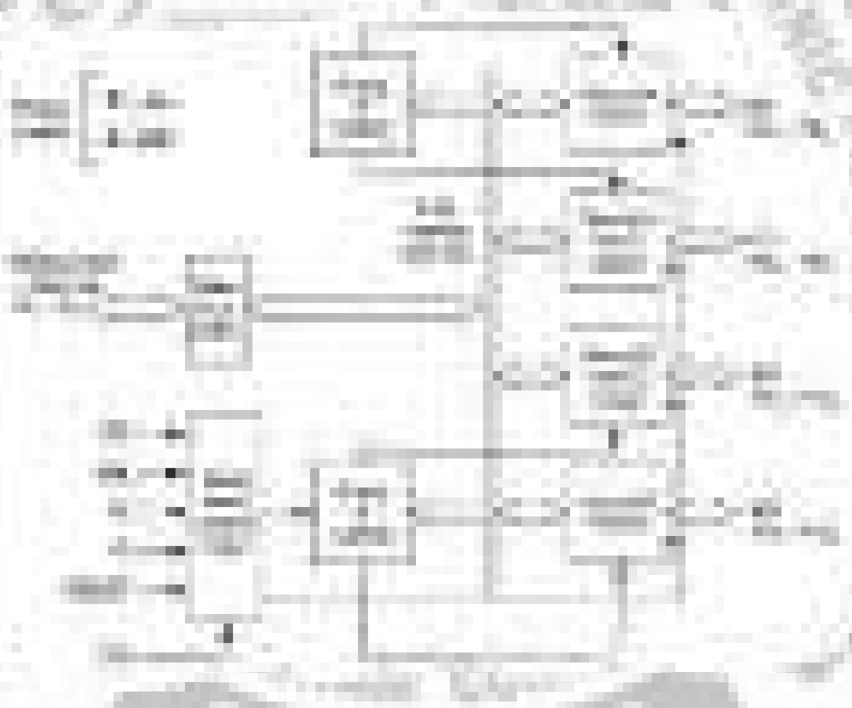
Prepared by: Prof. Md Istiyaque Ashraf
Guest Faculty,Dept't of ECE
PCE,Purnea

INTEL-8255

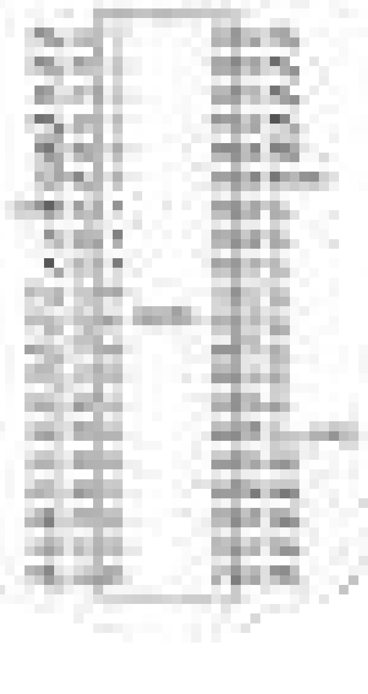
The 8255 is a general purpose programmable, Parallel I/O device designed for use with Intel microprocessors. It consists of three 8-bit bidirectional I/O ports (24 I/O lines) that can be programmed to transfer data under various conditions from simple I/O to interrupt I/O.

The three ports are PORT A, PORT B & PORT C. Port A contains one 8-bit output latch/buffer and one 8-bit input buffer. Port B is same as PORT A. However, PORT C can be split into two parts PORT C lower (PC0-PC3) and PORT C upper (PC7-PC4) by the control word. The three ports are divided in two groups Group A (PORT A and upper PORT C) Group B (PORT B and lower PORT C).

Block Diagram:



Pin Diagram:

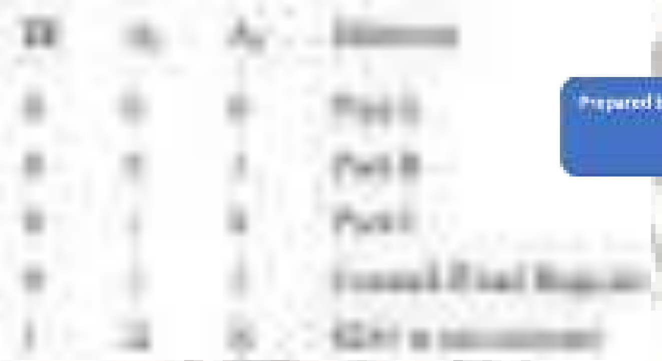


<u>Read write control logic</u>:

The function of this block is to manage all the internal and external transfers of both data and control or status word. The details of each pin connected with this block are described below.

$\overline{CS}$ (Chip Select)- A “Low” on this input pin enables the communication between the 8085 and MPU.

A_0 and A_1- These are the address lines of 8255 which are directly connected to the MPU lower address lines (A_0, A_1). The bit combination of these signals are shown below.



$\overline{CS}$	A_1	A_0	[illegible]
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Word Register
1	X	X	8255 is not selected

<u>PPI 8255 can operate in three modes:</u>

(a) Mode 0 (b) Mode 1 and (c) Mode 2.

Apart from these there is another mode called BSR mode (Bit Set/Reset mode)

The three modes are Mode 0, Mode 1 and Mode 2. These are I/O operations and selected only if D7 bit of the control word register is put as 1. The three operating modes of 8255 are distinguished in the following manner:

Mode 0: This is a basic or simple input/output mode, whose features are:

- Outputs are latched.
- Inputs are not latched.
- All ports (A, B, C_U, C_L) can be programmed in either input or output mode.
- Ports don’t have handshake or interrupt capability.
- Sixteen possible input/output configurations are possible.

Mode 1: In this mode, input or outputting of data is carried out by taking the help of handshaking signals, also known as strobe signals. The basic features of this mode are:

- Ports A and B can function as 8-bit I/O ports, taking the help of pins of Port C.
- I/Ps and O/Ps are latched.
- Interrupt logic is supported.
- Handshake signals are exchanged between CPU and peripheral prior to data transfer.
- In this mode, Port C is called status port.
- There are two groups in this mode—group A and group B. They can be configured separately. Each group consists of an 8-bit port and a 4-bit port. This 4-bit port is used for handshaking in each group.

Mode 2: In this mode, Port A can be set up for bidirectional data transfer using handshake signals from Port C. Port B can be set up either in mode 0 or mode 1.

The basic operations of the three modes are shown below:

The control word format, when 8255 is operated in I/O mode, is shown below. For 8255 PPI to be operated in I/O mode, D7 bit must be 1.

The three ports are divided into two groups—Groups A and B. Group A consists of Port A and C_U (PC_4–PC_7) . Port A can be operated in any of the modes—0, 1 or 2. Group B consists of Port B and C_L(PC_0–PC_3). Here Port B can be operated in either mode 0 or 1.

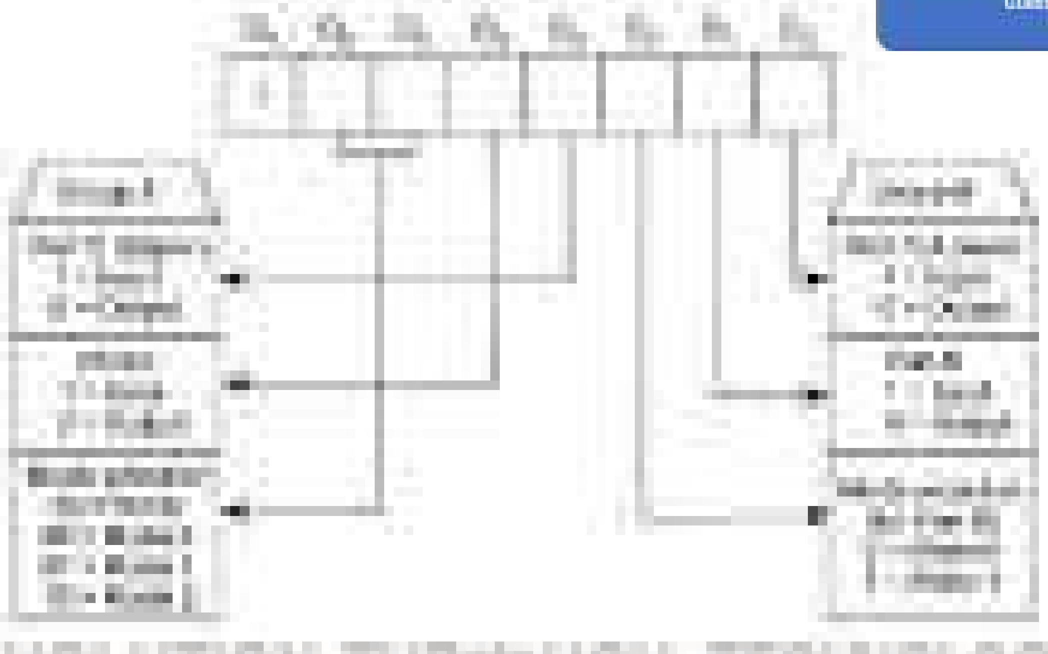


Fig: The CWR in the I/O mode

BSR (Bit Set Reset mode):

BSR mode stands for Bit Set Reset mode. The characteristics of BSR mode are:

- BSR mode is selected only when $D_7 = 0$ of the Control Word Register (CWR).
- It is concerned with bits of port C.
- Individual bits of Port C can either be Set or Reset.
- At a time, only a single bit of port C can be Set or Reset.
- Is used for control or on/off switch.
- BSR control word doesn’t affect ports A and B functioning.

The content of the control word register will be as follows, when used in the BSR mode and selects (either Sets or Resets) a particular bit of Port C at a time

Fig: The CWR in the BSR mode

ARDUINO

tutorialspoint

S I M P L Y E A S Y L E A R N I N G

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Audience

This tutorial is intended for enthusiastic students or hobbyists. With Arduino, one can get to know the basics of micro-controllers and sensors very quickly and can start building prototype with very little investment.

Prerequisites

Before you start proceeding with this tutorial, we assume that you are already familiar with the basics of C and C++. If you are not well aware of these concepts, then we will suggest you go through our short tutorials on C and C++. A basic understanding of microcontrollers and electronics is also expected.

Copyright & Disclaimer

Table of Contents

Arduino – Basics

1 ARDUINO – OVERVIEW

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

The key features are:

- Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.

- You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).

- Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.

- Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.

- Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Board Types

Various kinds of Arduino boards are available depending on different microcontrollers used. However, all Arduino boards have one thing in common: they are programed through the Arduino IDE.

The differences are based on the number of inputs and outputs (the number of sensors, LEDs, and buttons you can use on a single board), speed, operating voltage, form factor etc. Some boards are designed to be embedded and have no programming interface (hardware), which you would need to buy separately. Some can run directly from a 3.7V battery, others need at least 5V.

Here is a list of different Arduino boards available.

Arduino boards based on ATMEGA328 microcontroller

Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Uno R3	5V	16MHz	14	6	6	1	USB via ATMega16U2
Arduino Uno R3 SMD	5V	16MHz	14	6	6	1	USB via ATMega16U2
Red Board	5V	16MHz	14	6	6	1	USB via FTDI
Arduino Pro 3.3v/8 MHz	3.3V	8 MHz	14	6	6	1	FTDI-Compatible Header
Arduino Pro 5V/16MHz	5V	16MHz	14	6	6	1	FTDI-Compatible Header
Arduino mini 05	5V	16MHz	14	8	6	1	FTDI-Compatible Header
Arduino Pro mini 3.3v/8mhz	3.3V	8MHz	14	8	6	1	FTDI-Compatible Header
Arduino Pro mini 5v/16mhz	5V	16MHz	14	8	6	1	FTDI-Compatible Header
Arduino Ethernet	5V	16MHz	14	6	6	1	FTDI-Compatible Header
Arduino Fio	3.3V	8MHz	14	8	6	1	FTDI-Compatible Header
LilyPad Arduino 328 main board	3.3V	8MHz	14	6	6	1	FTDI-Compatible Header
LilyPad Arduino simply board	3.3V	8MHz	9	4	5	0	FTDI-Compatible Header

Arduino boards based on ATMEGA32u4 microcontroller

Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface

Arduino Leonardo	5V	16MHz	20	12	7	1	Native USB
Pro micro 5V/16MHz	5V	16MHz	14	6	6	1	Native USB
Pro micro 3.3V/8MHz	5V	16MHz	14	6	6	1	Native USB
LilyPad Arduino USB	3.3V	8MHz	14	6	6	1	Native USB

Arduino boards based on ATMEGA2560 microcontroller

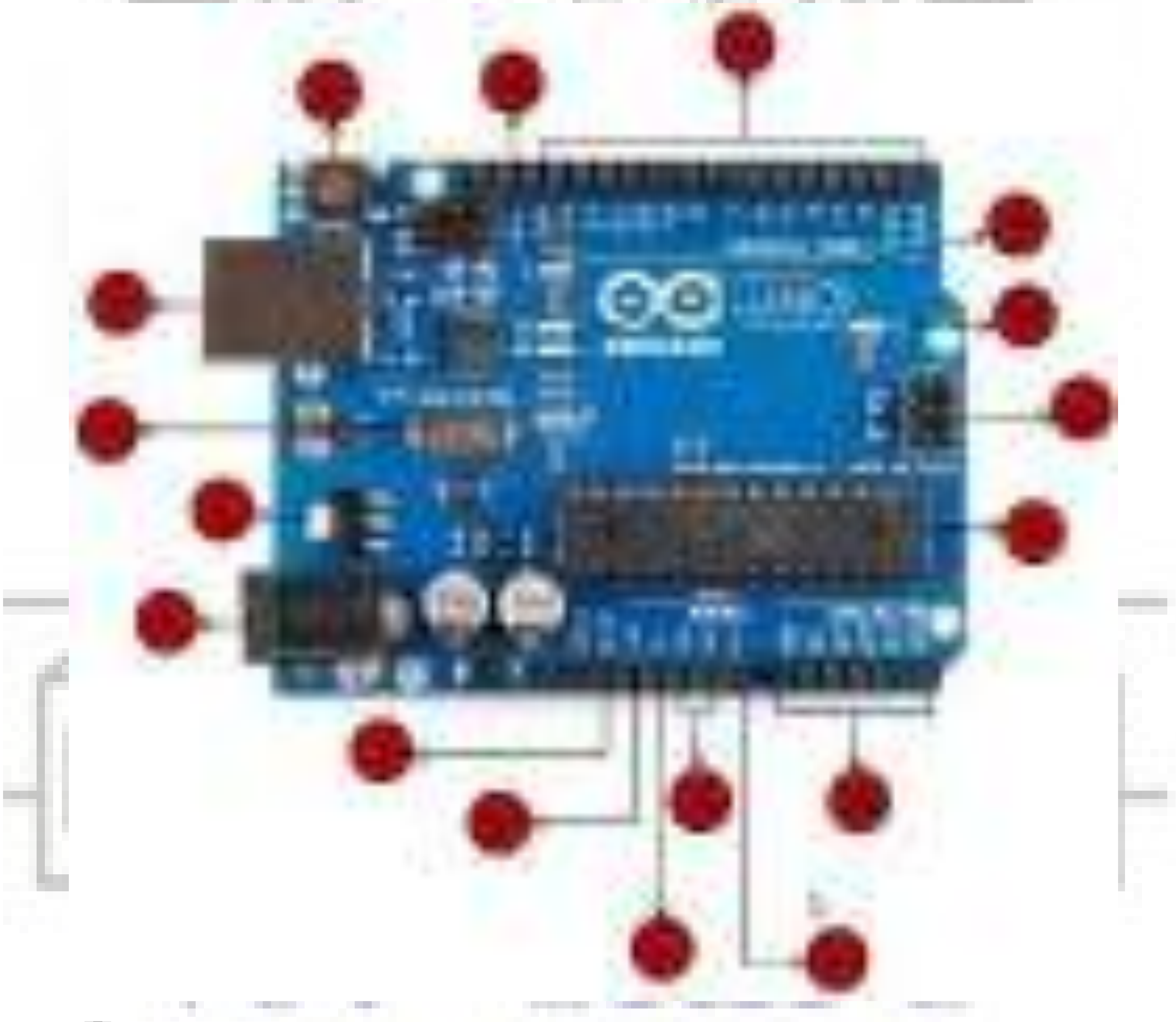
Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Mega 2560 R3	5V	16MHz	54	16	14	4	USB via ATMega16U2
Mega Pro 3.3V	3.3V	8MHz	54	16	14	4	FTDI-Compatible Header
Mega Pro 5V	5V	16MHz	54	16	14	4	FTDI-Compatible Header
Mega Pro Mini 3.3V	3.3V	8MHz	54	16	14	4	FTDI-Compatible Header

Arduino boards based on AT91SAM3X8E microcontroller

Board Name	Operating Volt	Clock Speed	Digital i/o	Analog Inputs	PWM	UART	Programming Interface
Arduino Due	3.3V	84MHz	54	12	12	4	USB native

ARDUINO – BOARD DESCRIPTION

In this chapter, we will learn about the different components on the Arduino board. We will study the Arduino UNO board because it is the most popular board in the Arduino board family. In addition, it is the best board to get started with electronics and coding. Some boards look a bit different from the one given below, but most Arduinos have majority of these components in common.



Power USB

Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection (1).

Power (Barrel Jack)

Arduino boards can be powered directly from the AC mains power supply by connecting it to the Barrel Jack (2).

Voltage Regulator

The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.

Crystal Oscillator

The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.

Arduino Reset

You can reset your Arduino board, i.e., start your program from the beginning. You can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).

Pins (3.3, 5, GND, V$_{in}$)

- 3.3V (6): Supply 3.3 output volt
-
- 5V (7): Supply 5 output volt
-
- Most of the components used with Arduino board works fine with 3.3 volt and 5 volt.
-
- GND (8)(Ground): There are several GND pins on the Arduino, any of which can be used to ground your circuit.
-
- Vin (9): This pin also can be used to power the Arduino board from an external power source, like AC mains power supply.
-

Analog pins

The Arduino UNO board has five analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.

Main microcontroller

Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Arduino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.

ICSP pin

Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.

Power LED indicator

This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.

TX and RX LEDs

On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.

Digital I / O

The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled "~" can be used to generate PWM.

AREF

AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins.

3. ARDUINO – INSTALLATION

After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board.

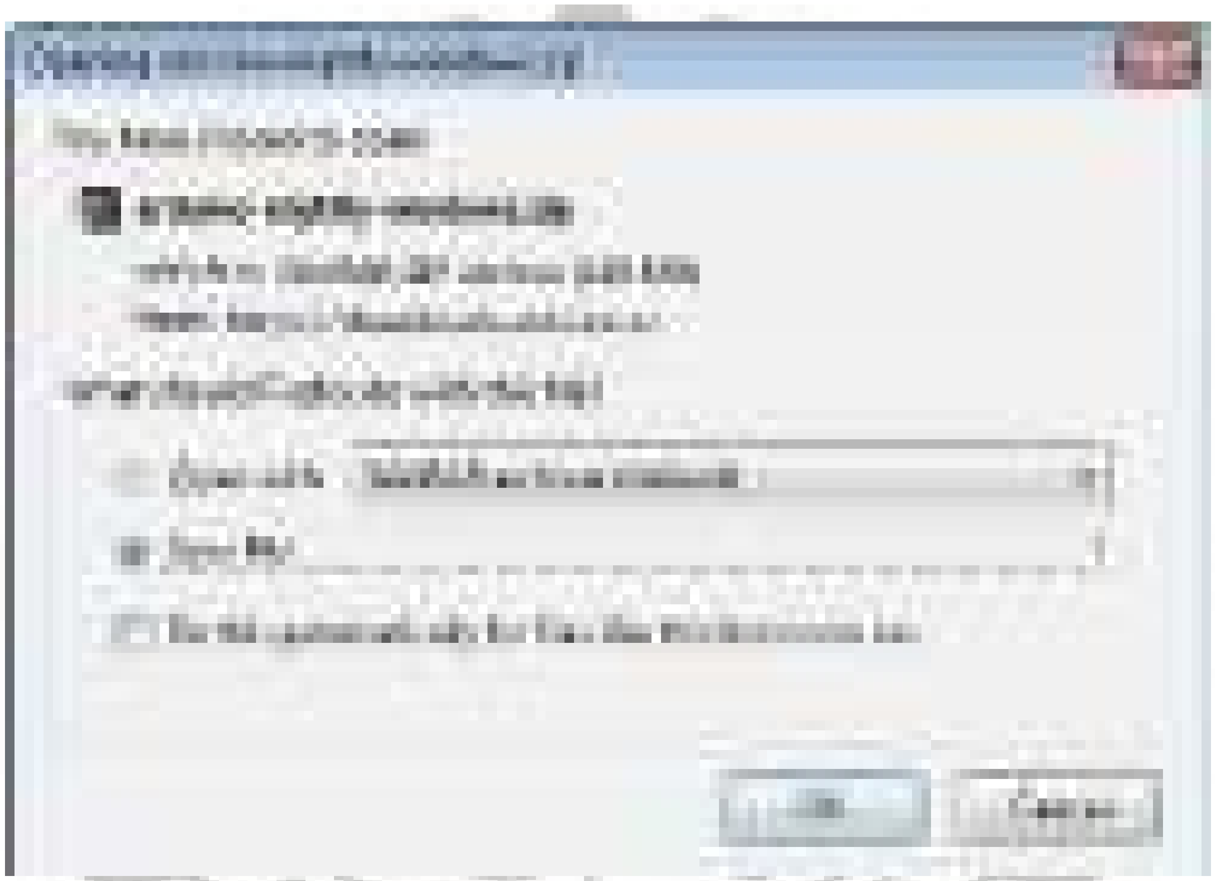
In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

Step 1: First you must have your Arduino board (you can choose your favorite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega 2560, or Diecimila, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the following image.

In case you use Arduino Nano, you will need an A to Mini-B cable instead as shown in the following image.

Step 2: Download Arduino IDE Software.

You can get different versions of Arduino IDE from the [Download page]() on the Arduino Official website. You must select your software, which is compatible with your operating system (Windows, IOS, or Linux). After your file download is complete, unzip the file.



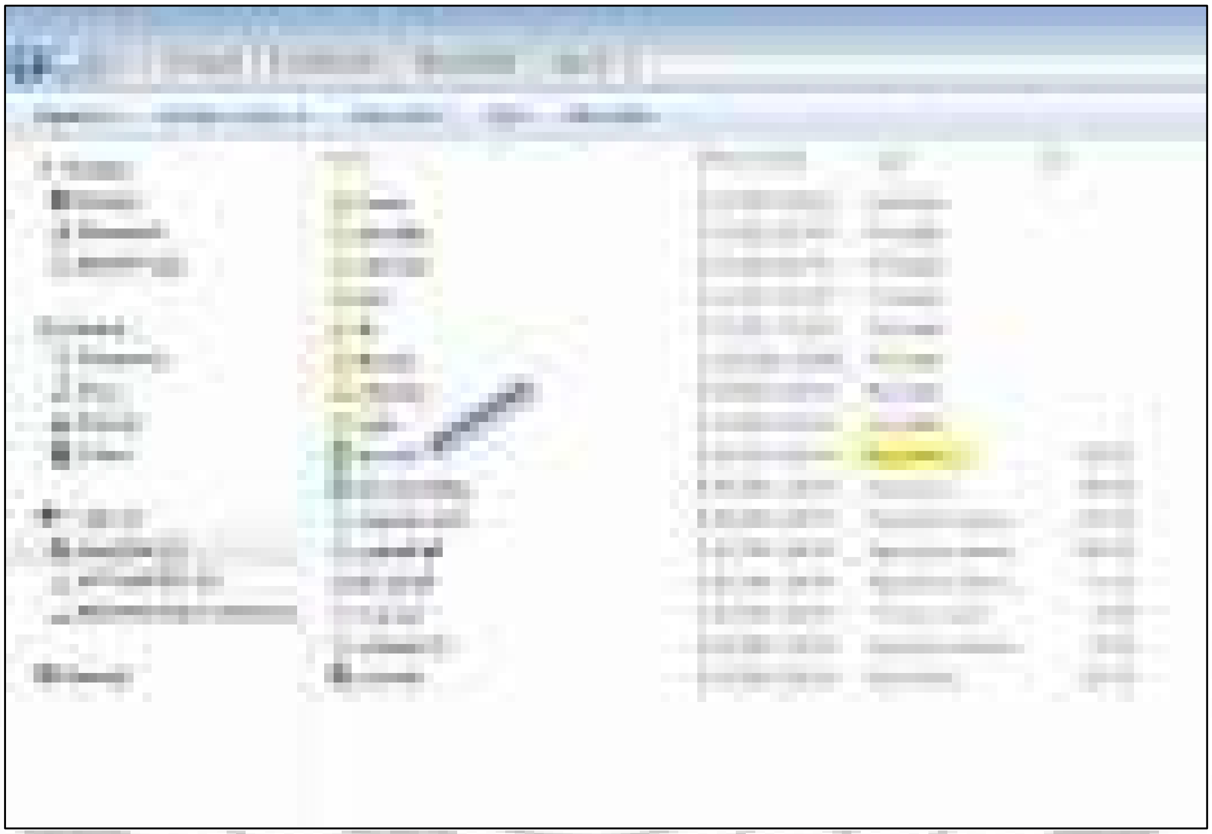
Step 3: Power up your board.

The Arduino Uno, Mega, Duemilanove and Arduino Nano automatically draw power from either, the USB connection to the computer or an external power supply. If you are using an Arduino Diecimila, you have to make sure that the board is configured to draw power from the USB connection. The power source is selected with a jumper, a small piece of plastic that fits onto two of the three pins between the USB and power jacks. Check that it is on the two pins closest to the USB port.

Connect the Arduino board to your computer using the USB cable. The green power LED (labeled PWR) should glow.

Step 4: Launch Arduino IDE.

After your Arduino IDE software is downloaded, you need to unzip the folder. Inside the folder, you can find the application icon with an infinity label (application.exe). Double-click the icon to start the IDE.



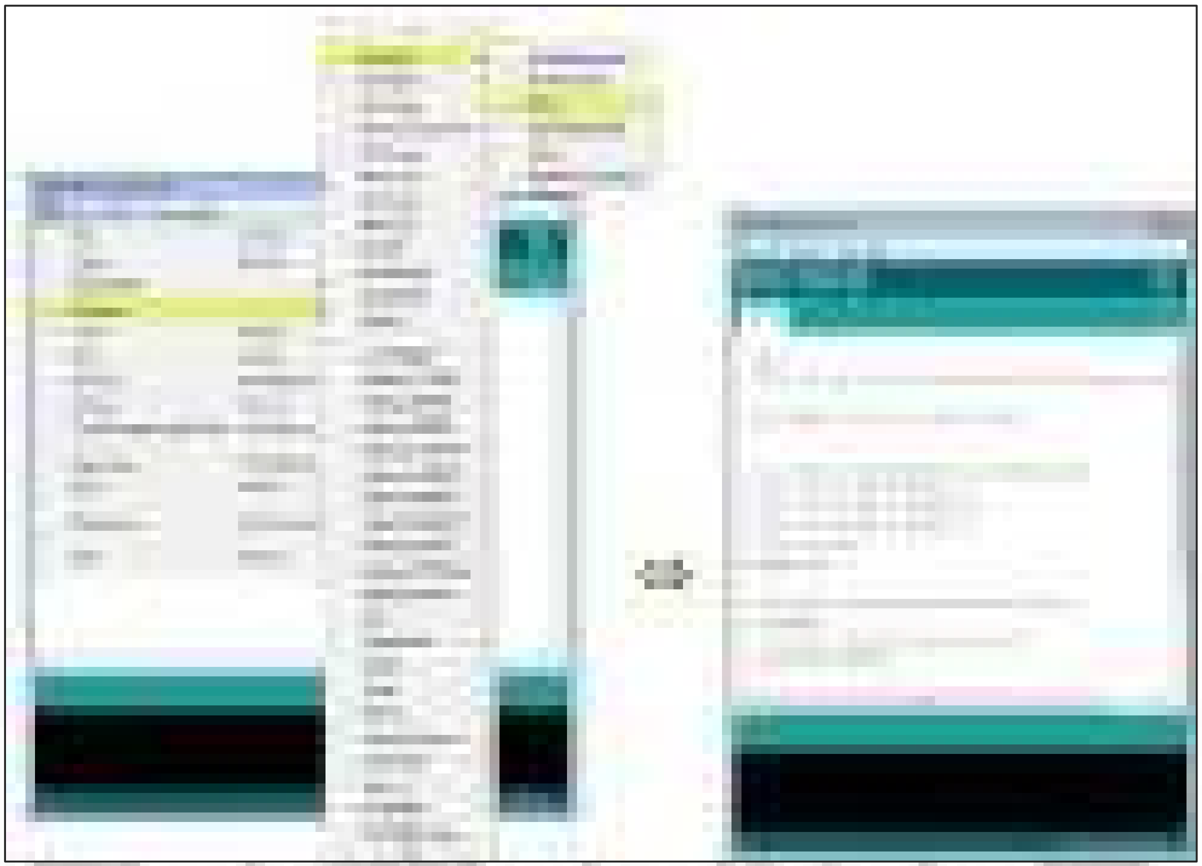
Step 5: Open your first project.

Once the software starts, you have two options:

- Create a new project.
- Open an existing project example.

To create a new project, select File --> New.

To open an existing project example, select File -> Example -> Basics -> Blink.



Here, we are selecting just one of the examples with the name **Blink**. It turns the LED on and off with some time delay. You can select any other example from the list.

Step 6: Select your Arduino board.

To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.

Go to Tools -> Board and select your board.

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**

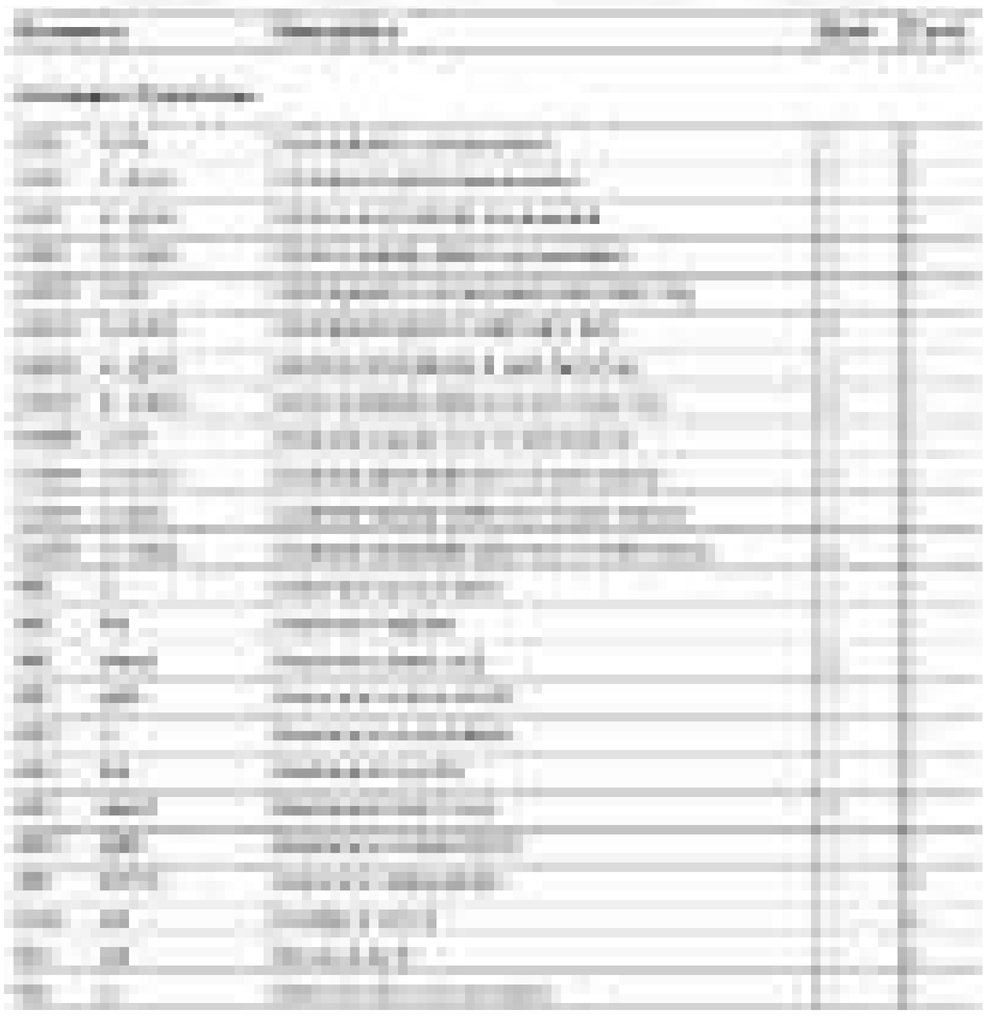
4.4 INSTRUCTION SET

8051 Microcontroller have set of instruction to perform different operations. There are five group of instruction which are listed below.

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

1.ARITHMETIC INSTRUCTION:

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


[illegible]

[Source: Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay, "The 8051Microcontroller and Embedded Systems: Using Assembly and C", Second Edition, Pearson Education,2011]

1. ADD A,Rn;

Adds the register Rn to the accumulator

Description:

Instruction adds the register Rn (R0-R7) to the accumulator. After addition, the result is stored in the accumulator

Before execution:

A=2Eh R4=12h

After execution:

A=40h R4=12h

2. ADD A,@Ri-

Adds the indirect RAM to the accumulator.

Ri: Register R0 or R1

Description:

Instruction adds the indirect RAM to the accumulator. Address of indirect RAM is stored in the Ri register (R0 or R1). After addition, the result is stored in the accumulator.

Register address:

R0=4Fh

Before execution:

A= 16h SUM= 33h

After execution :

A= 49h

3. ADDA,#DATA

Data: constant within 0-255 (0-FFh)

Description:

Instruction adds data (0-255) to the accumulator. After addition, the result is stored in the accumulator.

ADD A,#33h

Before execution:

A= 16h

After execution:

A= 49h)

4. SUBB A,direct-

Subtracts the direct byte from the accumulator witha borrow

Direct: arbitrary register with address 0-255(0-FFh)

Description:

Instruction subtracts the direct byte from the accumulator with a borrow. If the higher bit is subtracted from the lower bit then the carry flag is set. As it is direct addressing, the direct byte can be any SFRs or general-purpose register with address 0-7Fh. (0-127 dec.). The result is stored in the accumulator.

SUBB A,Rx

Before execution:

A=C9h, DIF=53h, C=0

After execution:

A=76h, C=0

5. INC A - Increments the accumulator by1

Description:

This instruction increments the value in the accumulator by 1. If the accumulator includes the number 255, the result of the operation will be 0.

Before execution:

A=E4h

After execution:

A=E5h

6. DEC A - Decrements the accumulator by1

Description: Instruction decrements the value in the accumulator by 1. If there is a 0 in the accumulator, the result of the operation is FFh. (255 dec.)

Syntax: DEC A;

Byte: 1 (instruction code);

STATUS register flags:

No flags are affected;

Before execution:

A=E4h

After execution:

A=E3h

7. DIV AB - Divides the accumulator by the registerB

Description:

Instruction divides the value in the accumulator by the value in the B register. After division the integer part of result is stored in the accumulator while the register contains the remainder. In case of dividing by 1, the flag OV is set and the result of division is unpredictable. The 8-bit quotient is stored in the accumulator and the 8-bit remainder is stored in the B register.

Before execution:

A=FBh (251dec.) B=12h (18 dec.)

After execution:

A=0Dh (13dec.) B=11h (17dec.)

13·18 + 17 =251

8. DA A - Decimal adjust accumulator

Description: Instruction adjusts the contents of the accumulator to correspond to a BCD number after two BCD numbers have been added by the ADD and ADDC instructions. The result in form of two 4-digit BCD numbers is stored in the accumulator.

Before execution:

A=56h (01010110) 56BCD

B=67h (01100111)67BCD

After execution:

A=BDh (10111101)

After BCD conversion:

A=23h (00100011), C=1 (Overflow)

(C+23=123) = 56+67

9. MUL AB - Multiplies A andB

Description: Instruction multiplies the value in the accumulator with the value in the B register. The low-order byte of the 16-bit result is stored in the accumulator, while the high byte remains in the B register. If the result is larger than 255, the overflow flag is set. The carry flag is not affected.

Before execution:

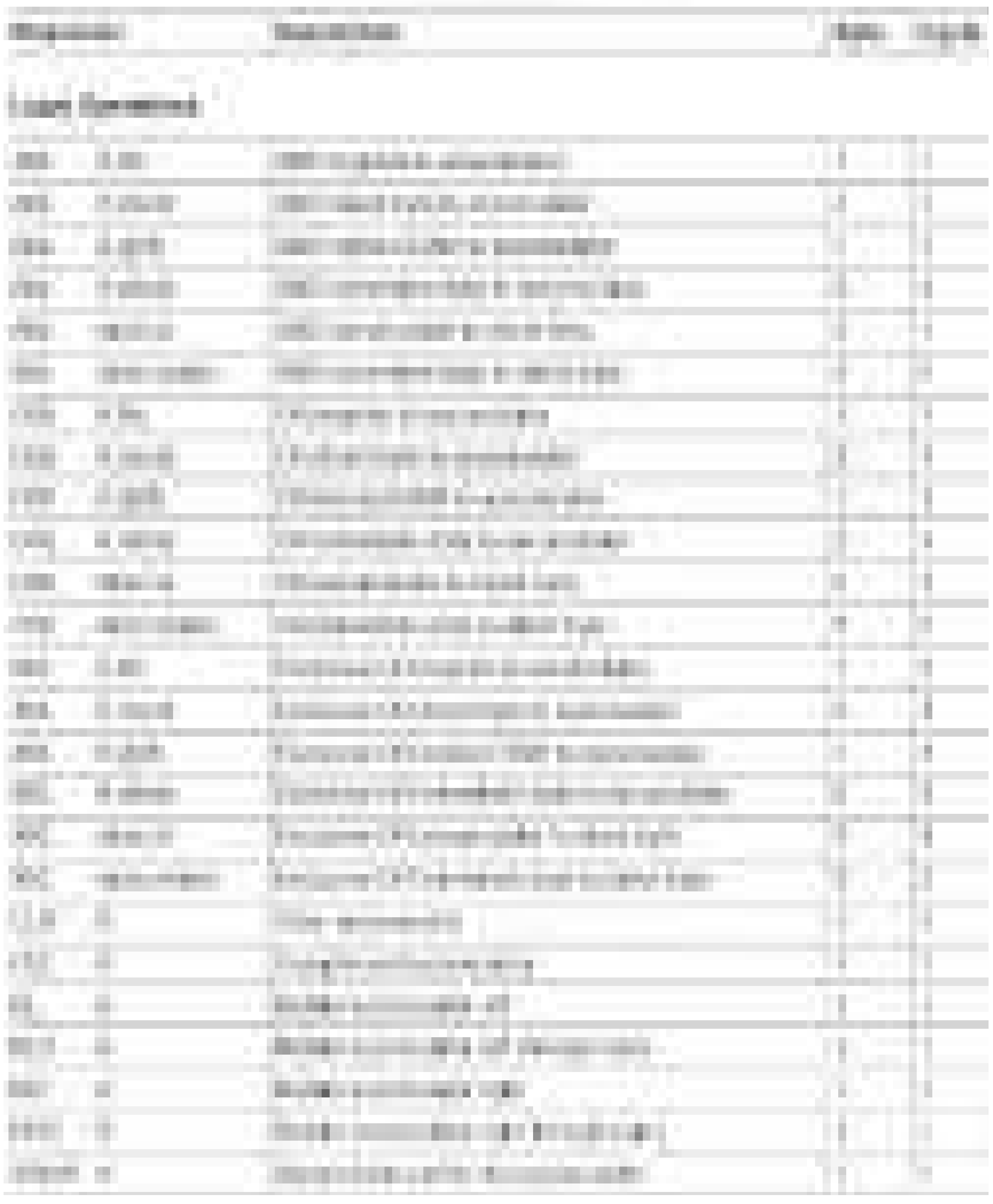
A=80 (50h) B=160 (A0h)

After execution:

A=0 B=32h A·B=80·160=12800 (3200h)

2.LOGICAL INSTRUCTION OF 8051 MICRO-CONTROLLER

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



[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay]

ANL A,Rn - AND register to the accumulator A: accumulator Rn: any R register(R0-R7)

Instruction performs logic AND operation between the accumulator and Rn register. The result is stored in the accumulator.

Syntax:

ANL A,Rn

Before execution:

A= C3h (11000011 Bin.) R5= 55h (01010101 Bin.)

After execution:

A= 41h (01000001 Bin.)

ORL A,Rn - OR register to the accumulator Rn: any R register (R0-R7)

Instruction performs logic OR operation between the accumulator and Rn register. The result is stored in the accumulator.

Syntax:

ORLA,Rn

Before execution:

A= C3h (11000011 Bin.) R5= 55h (01010101 Bin.)

After execution:

A= D7h (11010111 Bin.)

XRL A,Rn - Exclusive OR register to accumulator Rn: any R register (R0-R7)

Instruction performs exclusive OR operation between the accumulator and the Rn register. The result is stored in the accumulator.

Syntax:

XRL A,Rn

Before execution:

A= C3h (11000011 Bin.) R3= 55h (01010101 Bin.)

After execution:

A= 96h (10010110 Bin.)

CLR A - Clears the accumulator A: accumulator

Instruction clears the accumulator.

Syntax:

CLR A

After execution:

A=0

CPL A - Complements the accumulator

Instruction complements all the bits in the accumulator (1==>0, 0==>1).

Syntax:

CPL A

Before execution:

A=(00110110)

After execution:

A=(11001001)

RL A - Rotates the accumulator one bit left A: accumulator

Eight bits in the accumulator are rotated one bit left, so that the bit 7 is rotated into the bit 0 position.

Syntax:

RL A

Before execution:

A= C2h (11000010 Bin.)

After execution:

A=85h (10000101 Bin.)

RR A - Rotates the accumulator one bit right

A: accumulator All eight bits in the accumulator are rotated one bit right so that the bit 0 is rotated into the bit 7 position.

Syntax:

RR A

Before execution:

A= C2h (11000010Bin.)

After execution:

A= 61h (01100001Bin.)

RLC A - Rotates the accumulator one bit left through the carry flag A: accumulator

All eight bits in the accumulator and carry flag are rotated one bit left. After this operation, the bit 7 is rotated into the carry flag position and the carry flag is rotated into the bit 0 position.

Syntax:

RLC A

Before execution:

A= C2h(11000010 Bin.) C=0

After execution:

A= 85h(10000100Bin.) C=1

RRC A - Rotates the accumulator one bit right through the carry flag A: accumulator

All eight bits in the accumulator and carry flag are rotated one bit right. After this operation, the carry flag is rotated into the bit 7 position and the bit 0 is rotated into the carry flag position.

Syntax:

RRC A

Before execution:

A= C2h(11000010 Bin.) C=0

After execution:

A= 61h(01100001Bin.) C=0

SWAP A - Swaps nibbles within the accumulator

A: accumulator

A nibble refers to a group of 4 bits within one register (bit0-bit3 and bit4-bit7). This instruction interchanges high and low nibbles of the accumulator.

Syntax:

SWAPA

Before execution:

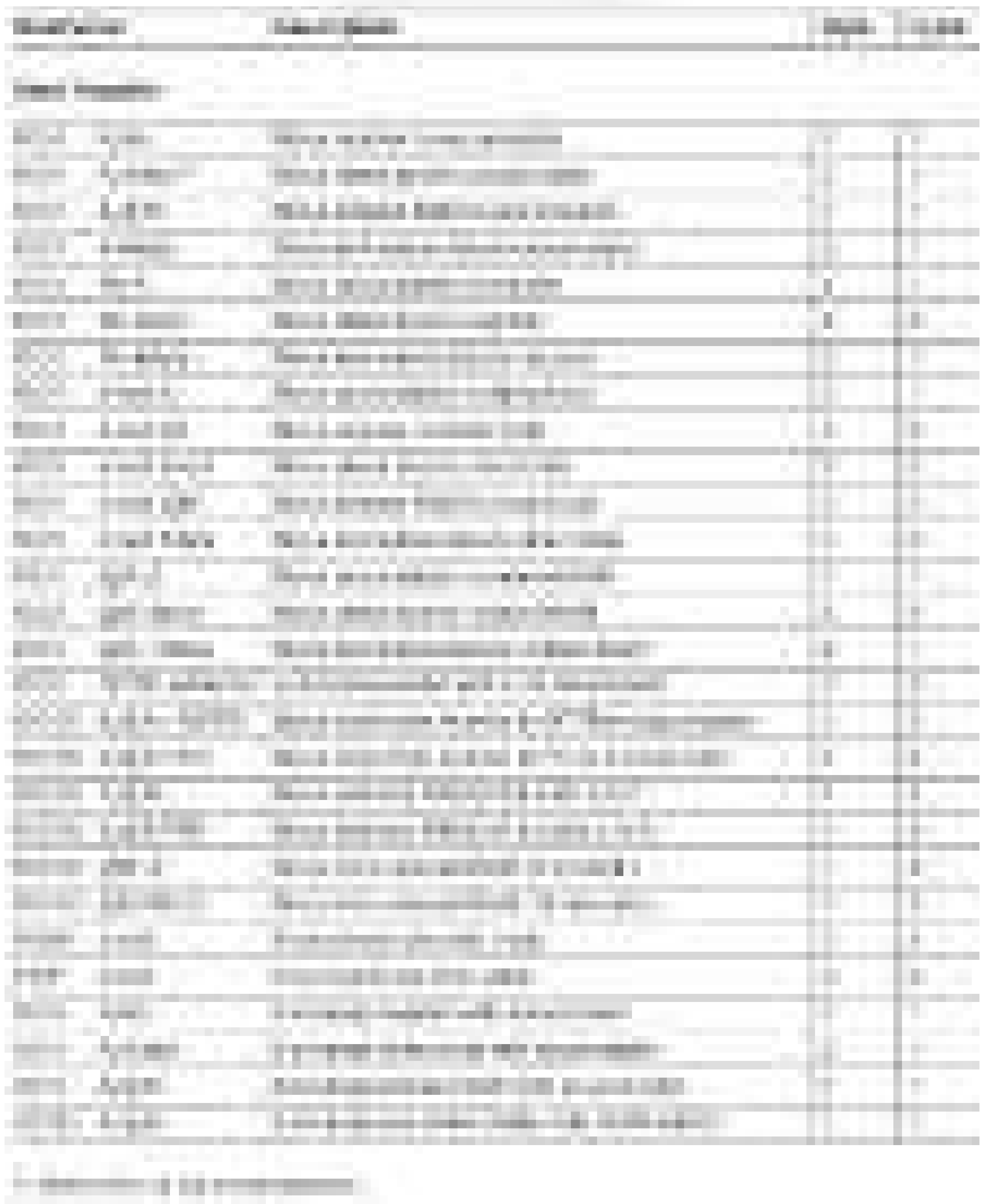
A=E1h (11100001)bin.

After execution:

A=1Eh (00011110)bin.

3. DATA TRANSFER INSTRUCTION OF 8051

These instructions are used to copy the content of source operand to Destination operand


[illegible]

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay]

MOV A,Rn - Moves the Rn register to the accumulator

The instruction moves the Rn register to the accumulator. The Rn register is not affected.

Syntax

MOV A,Rn

Beforeexecution:

R3=58h

After execution:

R3=58h A=58h

MOV A,@Ri - Moves the indirect RAM to the accumulator

Instruction moves the indirectly addressed register of RAM to the accumulator. The register address is stored in the Ri register (R0 or R1). The result is stored in the accumulator. The register is not affected.

Syntax:

MOV A,@Ri

Register Address SUM=F2h R0=F2h

Before execution:

SUM=58h

After execution:

A=58h SUM=58h

MOV A,#data - Moves the immediate data to the accumulator

Instruction moves the immediate data to the accumulator.

Syntax:

MOV A, #28

After execution:

A=28h

MOV direct,@Ri - Moves the indirect RAM to the direct byte

Instruction moves the indirectly adressed register of RAM to the direct byte. The register is not affected.

Syntax:

MOV Rx,@Ri

Register Address SUM=F3

Before execution:

SUM=58h R1=F3

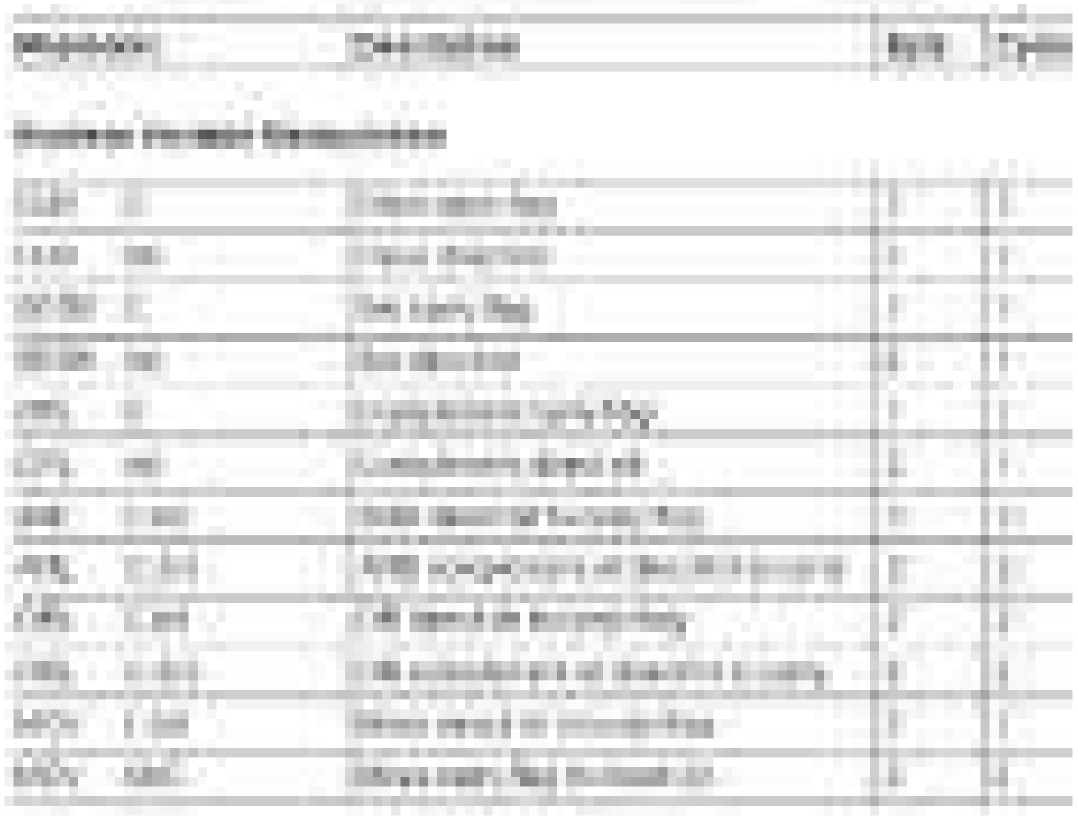
After execution:

SUM=58h TEMP=58h

MOVC A,@A+DPTR - Moves the code byte relative to the DPTR to the accumulator Instruction first adds the 16-bit DPTR register to the accumulator. The result of addition is then used as a memory address from which the 8-bit data is moved to the accumulator.

4. BIT-ORIENTED INSTRUCTIONS

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



Mnemonic		Description	Byte	Cycl
Boolean Variable Manipulation				
CLR	C	Clear carry flag	[illegible]	[illegible]
CLR	bit	Clear direct bit	[illegible]	[illegible]
SETB	C	Set carry flag	[illegible]	[illegible]
SETB	bit	Set direct bit	[illegible]	[illegible]
CPL	C	Complement carry flag	[illegible]	[illegible]
CPL	bit	Complement direct bit	[illegible]	[illegible]
ANL	C,bit	AND direct bit to carry flag	[illegible]	[illegible]
ANL	C,/bit	AND complement of direct bit to carry	[illegible]	[illegible]
ORL	C,bit	OR direct bit to carry flag	[illegible]	[illegible]
ORL	C,/bit	OR complement of direct bit to carry	[illegible]	[illegible]
MOV	C,bit	Move direct bit to carry flag	[illegible]	[illegible]
MOV	bit,C	Move carry flag to direct bit	[illegible]	[illegible]

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay , pg.no.29]

ANL C,bit - AND direct bit to the carry flag C: Carry flag

Bit: any bit of RAM,

Instruction performs logic AND operation between the direct bit and the carry flag.

Syntax:

ANL C,bit

Before execution:

ACC= 43h (01000011 Bin.)C=1

After execution:

ACC= 43h(01000011 Bin.) C=0

CLR C - clears the carry flag

C: Carry flag, Instruction clears the carry flag. After execution: C=0

CLR bit - clears the direct bit

Bit: any bit of RAM, Instruction clears the specified bit.

Syntax:

CLR P0.3

Before execution:

P0.3=1 (input pin)

After execution:

P0.3=0 (output pin)

CPL bit - Complements the direct bit

Bit: any bit of RAM, Instruction complements the specified bit of RAM (0==>1, 1==>0).

Syntax:

CPL P0.3

Before execution:

P0.3=1 (input pin)

After execution:

P0.3=0 (output pin)

CPL C - Complements the carry flag

C: Carry flag, Instruction complements the carry flag (0==>1, 1==>0).

Syntax:

CPL C

Before execution:

C=1

After execution:

C=0

MOV bit,C - Moves the carry flag to the direct bit C: Carry flag, Bit: any bit ofRAM Instruction moves the carry flag to the direct bit. After executing the instruction, the carry flag is not affected.

Syntax:

MOVP1.2,C

After execution:

If C=0 P1.2=0 If C=1 P1.2=1

MOV C,bit - Moves the direct bit to the carryflag C: Carry flag, Bit: any bit of RAM Instruction moves the direct bit to the carry flag. After executing the instruction, the bit is not affected.

Syntax:

MOV C, P1.4

After execution:

If P1.4=0 C=0 If P1.4=1 C=1

SETB C - Sets the carry flag

C: Carry flag, Instruction sets the carry flag.

Syntax:

SETB C

After execution: C=1

SETB bit - Sets the direct bit

Bit: any bit of RAM

Instruction sets the specified bit. The register containing that bit must belong to the group of the so called bit addressable registers.

Syntax:

SETB P0.1

Before execution:

P0.1 = 34h (00110100) pin 1 is configured as an output

After execution:

P0.1 = 35h (00110101) pin 1 is configured as an inputs

5. BRANCH INSTRUCTION OF 8051 CONTROLLER

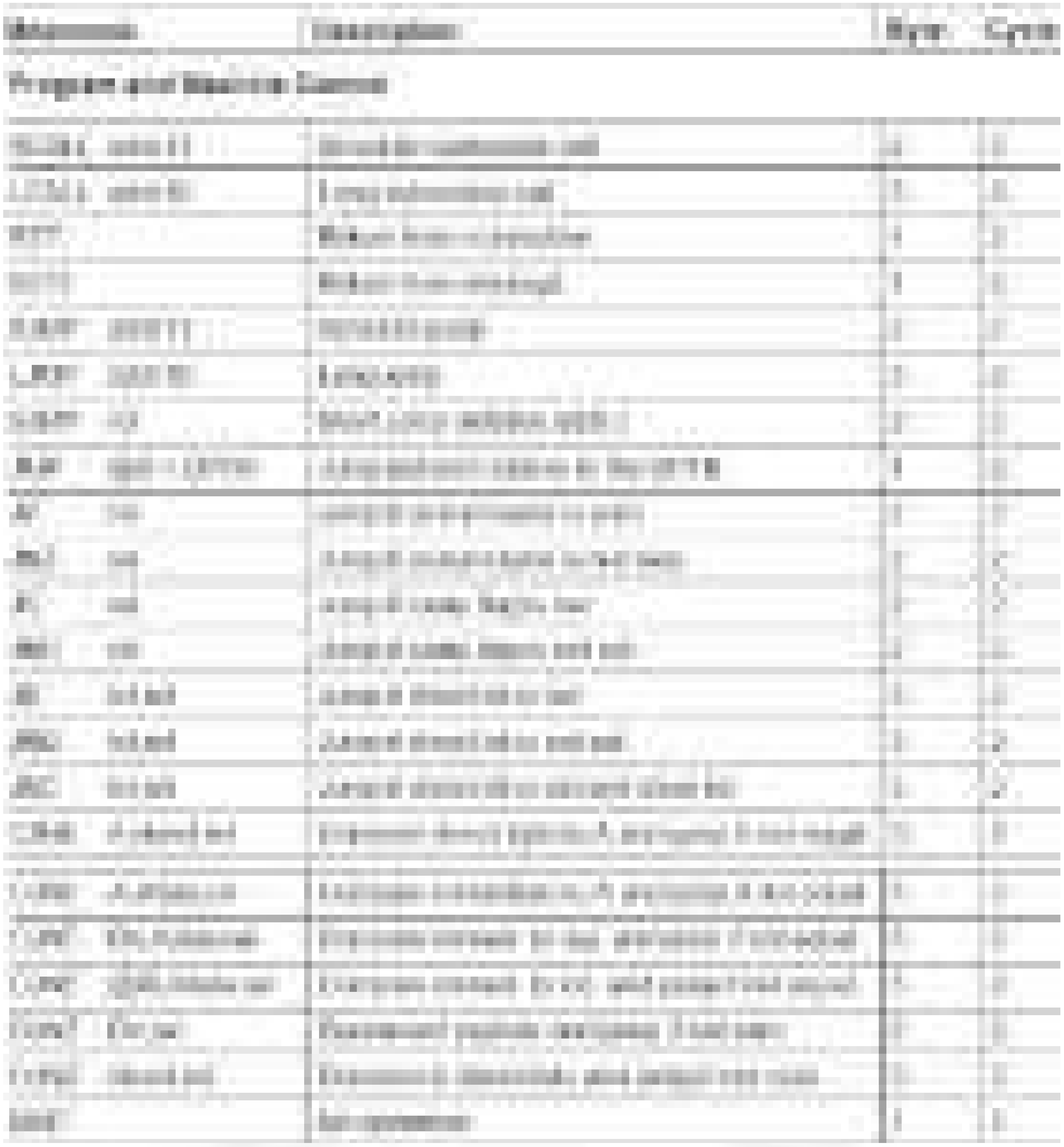
There are two kinds of branch instructions:

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.



[illegible]	[illegible]	[illegible]	[illegible]
[illegible]			
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi, Janice Gillispie Mazidi, Rolin McKinlay]

Notes on Data Addressing Modes

Rn	-	Working register R0-R7
direct	-	128 internal RAM locations, any I/O port, control or status register
@Ri	-	Indirect internal or external RAM location addressed by register R0 or R1
#data	-	8-bit constant included in instruction
#data 16	-	16-bit constant included as bytes 2 and 3 of instruction
bit	-	128 software flags, any bitaddressable I/O pin, control or status bit
A	-	Accumulator

Notes on Program Addressing Modes

addr16	-	Destination address for LCALL and LJMP may be anywhere within the 64-Kbyte program memory address space.
addr11	-	Destination address for ACALL and AJMP will be within the same 2-Kbyte page of program memory as the first byte of the following instruction.
rel	-	SJMP and all conditional jumps include an 8 bit offset byte. Range is + 127/– 128 bytes relative to the first byte of the following instruction.

All mnemonics copyrighted: © Intel Corporation 1980

ACALL addr11

Function:	Absolute call
Description:	ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, op code bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of program memory as the first byte of the instruction following ACALL. No flags are affected.
Example:	Initially SP equals 07_H. The label "SUBRTN" is at program memory location 0345_H. After executing the instruction ACALL SUBRTN at location 0123_H, SP will contain 09_H, internal RAM location 08_H and 09_H will contain 25_H and 01_H, respectively, and the PC will contain 0345_H.
Operation:	ACALL (PC) ← (PC) + 2 (SP) ← (SP) + 1 ((SP)) ← (PC7-0) (SP) ← (SP) + 1 ((SP)) ← (PC15-8) (PC10-0) ← page address

Encoding:

a10 a9 a8 1	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Bytes: 2

Cycles: 2

ADD A, <src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary carry flags are set, respectively, if there is a carry out of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The accumulator holds $0C3_H$ (11000011_B) and register 0 holds $0AA_H$ (10101010_B). The instruction

```
ADD     A,R0
```

will leave $6D_H$ (01101101_B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,Rn

Operation: ADD
(A) ← (A) + (Rn)

Encoding:

0 0 1 0	1 r r r

Bytes: 1

Cycles: 1

ADD A,direct

Operation: ADD
(A) ← (A) + (direct)

Encoding:

0 0 1 0	0 1 0 1	direct address

Bytes: 2

Cycles: 1

ADD A, @Ri

Operation:	ADD (A) ← (A) + ((Ri))
Encoding:	0 0 1 0 | 0 1 1 i
Bytes:	1
Cycles:	1

ADD A, #data

Operation:	ADD (A) ← (A) + #data
Encoding:	0 0 1 0 | 0 1 0 0 immediate data
Bytes:	2
Cycles:	1

ADDC A, < src-byte>

Function: Add with carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary carry flags are set, respectively, if there is a carry out of bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The accumulator holds $0C3_H$ (11000011B) and register 0 holds $0AA_H$ (10101010B) with the carry flag set. The instruction

ADDC A,R0

will leave $6E_H$ (01101110B) in the accumulator with AC cleared and both the carry flag and OV set to 1.

ADDC A,Rn

Operation: ADDC
$(A) \leftarrow (A) + (C) + (Rn)$

Encoding:

0 0 1 1	1 r r r

Bytes: 1

Cycles: 1

ADDC A,direct

Operation: ADDC
$(A) \leftarrow (A) + (C) + (\text{direct})$

Encoding:

0 0 1 1	0 1 0 1	direct address

Bytes: 2

Cycles: 1

ADDC A, @Ri

Operation: ADDC
(A) ← (A) + (C) + ((Ri))

Encoding:

0 0 1 1	0 1 1 i

Bytes: 1

Cycles: 1

ADDC A, #data

Operation: ADDC
(A) ← (A) + (C) + #data

Encoding:

0 0 1 1	0 1 0 0	immediate data

Bytes: 2

Cycles: 1

AJMP addr11

Function:	Absolute jump
Description:	AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), op code bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.
Example:	The label "JMPADR" is at program memory location 0123_H. The instruction AJMP JMPADR is at location 0345_H and will load the PC with 0123_H.
Operation:	AJM P (PC) ← (PC) + 2 (PC10-0) ← page address
Encoding:	a10 a9 a8 0 | 0 0 0 1 a7 a6 a5 a4 | a3 a2 a1 a0
Bytes:	2
Cycles:	2

ANL <dest-byte>, <src-byte>

Function: Logical AND for byte variables

Description: ANL performs the bitwise logical AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is a accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

Note:

When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: If the accumulator holds $0C3_H$ (11000011B) and register 0 holds $0AA_H$ (10101010B) then the instruction

ANL A,R0

will leave 81_H (10000001B) in the accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the accumulator at run-time.
The instruction

ANL P1, #01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A,Rn

Operation: ANL
$(A) \leftarrow (A) \wedge (Rn)$

Encoding:

0 1 0 1	1 r r r

Bytes: 1

Cycles: 1

ANL A,direct

Operation: ANL
$(A) \leftarrow (A) \wedge (direct)$

Encoding:

0 1 0 1	0 1 0 1	direct address

Bytes: 2

Cycles: 1

ANL A, @Ri

Operation: ANL
$(A) \leftarrow (A) \wedge ((Ri))$

Encoding:

0 1 0 1	0 1 1 i

Bytes: 1

Cycles: 1

ANL A, #data

Operation: ANL
$(A) \leftarrow (A) \wedge \#data$

Encoding:

0 1 0 1	0 1 0 0	immediate data

Bytes: 2

Cycles: 1

ANL direct,A

Operation: ANL
$(direct) \leftarrow (direct) \wedge (A)$

Encoding:

0 1 0 1	0 1 0 1	direct address

Bytes: 2

Cycles: 1

ANL direct, #data

Operation: ANL
(direct) ← (direct) ∧ #data

Encoding:

0 1 0 1	0 0 1 1	direct address	immediate data

Bytes: 3

Cycles: 2

ANL C, <src-bit>

Function: Logical AND for bit variables

Description: If the Boolean value of the source bit is a logic 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/" preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected*. No other flags are affected.

Only direct bit addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC.7 = 1, and OV = 0:

```
MOV     C,P1.0              ; Load carry with input pin state
ANL     C,ACC.7             ; AND carry with accumulator bit 7
ANL     C,/OV               ; AND with inverse of overflow flag
```

ANL C,bit

Operation: ANL
$(C) \leftarrow (C) \wedge (bit)$

Encoding:

1 0 0 0	0 0 1 0	bit address

Bytes: 2

Cycles: 2

ANL C,/bit

Operation: ANL
$(C) \leftarrow (C) \wedge \neg (bit)$

Encoding:

1 0 1 1	0 0 0 0	bit address

Bytes: 2

Cycles: 2

CJNE <dest-byte >, < src-byte >, rel

Function: Compare and jump if not equal

Description: CJNE compares the magnitudes of the tirst two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The accumulator contains 34_H. Register 7 contains 56_H. The first instruction in the sequence

```
              CJNE     R7, # 60H, NOT_EQ
;             . . .    . . . . .                 ; R7 = 60H
NOT_EQ        JC       REQ_LOW                   ; If R7 < 60H
;             . . .    . . . . .                 ; R7 > 60H
```

sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60_H.

If the data being presented to port 1 is also 34_H, then the instruction

```
WAIT:    CJNE     A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was input on P1, the program will loop at this point until the P1 data changes to 34_H).

CJNE A,direct,rel

Operation: (PC) ← (PC) + 3
if (A) < > (direct)
then (PC) ← (PC) + relative offset
if (A) < (direct)
then (C) ←1
else (C) ← 0

Encoding:

1 0 1 1	0 1 0 1	direct address	rel. address

Bytes: 3

Cycles: 2

CJNE A, #data,rel

Operation: (PC) ← (PC) + 3
if (A) < > data
then (PC) ← (PC) + relative offset
if (A) ← data
then (C) ←1
else (C) ← 0

Encoding:

1 0 1 1	0 1 0 0	immediate data	rel. address

Bytes: 3

Cycles: 2

CJNE RN, #data, rel

Operation: (PC) ← (PC) + 3
if (Rn) < > data
then (PC) ← (PC) + relative offset
if (Rn) < data
then (C) ← 1
else (C) ← 0

Encoding:

1 0 1 1	1 r r r	immediate data	rel. address

Bytes: 3

Cycles: 2

CJNE @Ri, #data,rel

Operation: (PC) ← (PC) + 3
if ((Ri)) < > data
then (PC) ← (PC) + relative offset
if ((Ri)) < data
then (C) ← 1
else (C) ← 0

Encoding:

1 0 1 1	0 1 1 i	immediate data	rel. address

Bytes: 3

Cycles: 2

CLR A

Function:	Clear accumulator
Description:	The accumulator is cleared (all bits set to zero). No flags are affected.
Example:	The accumulator contains $5C_H$ (01011100B). The instruction CLR A will leave the accumulator set to 00_H (00000000B).
Operation:	CLR $(A) \leftarrow 0$
Encoding:	1 1 1 0 | 0 1 0 0
Bytes:	1
Cycles:	1

CLR bit

Function:	Clear bit
Description:	The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.
Example:	Port 1 has previously been written with $5D_H$ (01011101B). The instruction CLR P1.2 will leave the port set to 59_H (01011001B).

CLR C

Operation:	CLR (C) ← 0	
Encoding:	1 1 0 0	0 0 1 1
Bytes:	1	
Cycles:	1	

CLR bit

Operation:	CLR (bit) ← 0		
Encoding:	1 1 0 0	0 0 1 0	bit address
Bytes:	2		
Cycles:	1		

CPL A

Function:	Complement accumulator
Description:	Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice versa. No flags are affected.
Example:	The accumulator contains $5C_H$ (01011100B). The instruction CPL A will leave the accumulator set to $0A3_H$ (10100011 B).
Operation:	CPL $(A) \leftarrow \neg (A)$
Encoding:	1 1 1 1 | 0 1 0 0
Bytes:	1
Cycles:	1

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice versa. No other flags are affected. CPL can operate on the carry or any directly addressable bit.

Note:

When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: Port 1 has previously been written with $5D_H$ (01011101_B). The instruction sequence

```
CPL     P1.1
CPL     P1.2
```

will leave the port set to $5B_H$ (01011011_B).

CPL C

Operation: CPL
$(C) \leftarrow \neg (C)$

Encoding:

1 0 1 1	0 0 1 1

Bytes: 1

Cycles: 1

CPL bit

Operation: CPL
$(\text{bit}) \leftarrow \neg (\text{bit})$

Encoding:

1 0 1 1	0 0 1 0	bit address

Bytes: 2

Cycles: 1

DA A

Function: Decimal adjust accumulator for addition

Description: DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially; this instruction performs the decimal conversion by adding 00_H, 06_H, 60_H, or 66_H to the accumulator, depending on initial accumulator and PSW conditions.

Note:

DA A *cannot* simply convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The accumulator holds the value 56_H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67_H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence

```
ADDC    A,R3
DA      A
```

will first perform a standard two's-complement binary addition, resulting in the value $0BE_H$ (10111110B) in the accumulator. The carry and auxiliary carry flags will be cleared.

The decimal adjust instruction will then alter the accumulator to the value 24_H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the decimal adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01_H or 99_H. If the accumulator initially holds 30_H (representing the digits of 30 decimal), then the instruction sequence

```
ADD     A, #99H
DA      A
```

will leave the carry set and 29_H in the accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 – 1 = 29.

Operation: DA

```
contents of accumulator are BCD
if [[(A3-0) > 9] ∨ [(AC) = 1]]
then (A3-0) ← (A3-0) + 6
and
if [[(A7-4) > 9] ∨ [(C) = 1]]
then (A7-4) ← (A7-4) + 6
```

Encoding:

1 1 0 1	0 1 0 0

Bytes: 1

Cycles: 1

DEC byte

Function:	Decrement
Description:	The variable indicated is decremented by 1. An original value of 00_H will underflow to $0FF_H$. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note:

When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains $7F_H$ (01111111B). Internal RAM locations $7E_H$ and $7F_H$ contain 00_H and 40_H, respectively. The instruction sequence

```
DEC     @R0
DEC     R0
DEC     @R0
```

will leave register 0 set to $7E_H$ and internal RAM locations $7E_H$ and $7F_H$ set to $0FF_H$ and $3F_H$.

DEC A

Operation: DEC
$(A) \leftarrow (A) - 1$

Encoding:

0 0 0 1	0 1 0 0

Bytes: 1

Cycles: 1

DEC Rn

Operation: DEC
$(Rn) \leftarrow (Rn) - 1$

Encoding:

0 0 0 1	1 r r r

Bytes: 1

Cycles: 1

DEC direct

Operation:	DEC (direct) ← (direct) – 1
Encoding:	0 0 0 1 | 0 1 0 1 | direct address
Bytes:	2
Cycles:	1

DEC @Ri

Operation:	DEC ((Ri)) ← ((Ri)) – 1
Encoding:	0 0 0 1 | 0 1 1 i
Bytes:	1
Cycles:	1

DIV AB

Function:	Divide
Description:	DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.
	Exception: If B had originally contained 00_H, the values returned in the accumulator and B register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.
Example:	The accumulator contains 251 ($0FB_H$ or 11111011B) and B contains 18 (12_H or 00010010B). The instruction
	DIV AB
	will leave 13 in the accumulator ($0D_H$ or 00001101 B) and the value 17 (11_H or 00010001B) in B, since 251 = (13x18) + 17. Carry and OV will both be cleared.
Operation:	DIV
	(A15-8) (B7-0) ← (A) / (B)
Encoding:	1 0 0 0 | 0 1 0 0
Bytes:	1
Cycles:	4

DJNZ <byte>, < rel-addr>

Function: Decrement and jump if not zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00_H will underflow to $0FF_H$. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note:

When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Internal RAM locations 40_H, 50_H, and 60_H contain the values, 01_H, 70_H, and 15_H, respectively. The instruction sequence

```
DJNZ 40H,LABEL_1
DJNZ 50H,LABEL_2
DJNZ 60H,LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00_H, $6F_H$, and 15_H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence

```
        MOV     R2, #8
TOGGLE: CPL     P1.7
        DJNZ    R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Operation: DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) – 1
if (Rn) > 0 or (Rn) < 0
then (PC) ← (PC) + rel

Encoding:

1 1 0 1	1 r r r	rel. address

Bytes: 2

Cycles: 2

DJNZ direct,rel

Operation: DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) – 1
if (direct) > 0 or (direct) < 0
then (PC) ← (PC) + rel

Encoding:

1 1 0 1	0 1 0 1	direct address	rel. address

Bytes: 3

Cycles: 2

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of $0FF_H$ will overflow to 00_H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note:

When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains $7E_H$ (01111110B). Internal RAM locations $7E_H$ and $7F_H$ contain $0FF_H$ and 40_H, respectively. The instruction sequence

```
INC     @R0
INC     R0
INC     @R0
```

will leave register 0 set to $7F_H$ and internal RAM locations $7E_H$ and $7F_H$ holding (respectively) 00_H and 41_H.

INC A

Operation: INC
$(A) \leftarrow (A) + 1$

Encoding:

0 0 0 0	0 1 0 0

Bytes: 1

Cycles: 1

INC Rn

Operation: INC
$(Rn) \leftarrow (Rn) + 1$

Encoding:

0 0 0 0	1 r r r

Bytes: 1

Cycles: 1

INC direct

Operation: INC
(direct) ← (direct) + 1

Encoding:	0 0 0 0	0 1 0 1		direct address
Bytes:	2			
Cycles:	1			

INC @Ri

Operation: INC
((Ri)) ← ((Ri)) + 1

Encoding:	0 0 0 0	0 1 1 i
Bytes:	1	
Cycles:	1	

INC DPTR

Function: Increment data pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from $0FF_H$ to 00_H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Registers DPH and DPL contain 12_H and $0FE_H$, respectively. The instruction sequence

```
INC     DPTR
INC     DPTR
INC     DPTR
```

will change DPH and DPL to 13_H and 01_H.

Operation: INC
(DPTR) $\leftarrow$ (DPTR) + 1

Encoding:

1 0 1 0	0 0 1 1

Bytes: 1

Cycles: 2

JB bit,rel

Function:	Jump if bit is set
Description:	If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Example: The data present at input port 1 is 11001010B. The accumulator holds 56 (01010110B). The instruction sequence

```
JB      P1.2,LABEL1
JB      ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

Operation: JB

```
(PC) ← (PC) + 3
if (bit) = 1
then (PC) ← (PC) + rel
```

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Bytes: 3

Cycles: 2

JBC bit,rel

Function:	Jump if bit is set and clear bit
Description:	If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *In either case, clear the designated bit.* The branch destination is computed by adding the signed relative displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note:

When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: The accumulator holds 56_H (01010110B). The instruction sequence

```
JBC     ACC.3,LABEL1
JBC     ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 52_H (01010010B).

Operation:

```
JBC
(PC) ← (PC) + 3
if (bit) = 1
then (bit) ← 0
     (PC) ← (PC) + rel
```

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Bytes:	3
Cycles:	2

JC rel

Function:	Jump if carry is set
Description:	If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.
Example:	The carry flag is cleared. The instruction sequence

```
JC      LABEL1
CPL     C
JC      LABEL2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Operation:
```
JC
(PC) ← (PC) + 2
if (C) = 1
then (PC) ← (PC) + rel
```

Encoding:

0 1 0 0	0 0 0 0	rel. address

Bytes: 2

Cycles: 2

JMP @A + DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
          MOV    DPTR, #JMP_TBL
          JMP    @A + DPTR
JMP_TBL:  AJMP   LABEL0
          AJMP   LABEL1
          AJMP   LABEL2
          AJMP   LABEL3
```

If the accumulator equals 04_H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Operation: JMP
(PC) ← (A) + (DPTR)

Encoding:

0 1 1 1	0 0 1 1

Bytes: 1

Cycles: 2

JNB bit,rel

Function: Jump if bit is not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The accumulator holds 56_H (01010110B). The instruction sequence

```
JNB     P1.3,LABEL1
JNB     ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

Operation: JNB
(PC) ← (PC) + 3
if (bit) = 0
then (PC) ← (PC) + rel.

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Bytes: 3

Cycles: 2

JNC rel

Function:	Jump if carry is not set
Description:	If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.
Example:	The carry flag is set. The instruction sequence JNC LABEL1 CPL C JNC LABEL2 will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.
Operation:	JNC (PC) ← (PC) + 2 if (C) = 0 then (PC) ← (PC) + rel
Encoding:	0 1 0 1 | 0 0 0 0 rel. address
Bytes:	2
Cycles:	2

JNZ rel

Function:	Jump if accumulator is not zero
Description:	If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Example: The accumulator originally holds 00_H. The instruction sequence

```
JNZ     LABEL1
INC     A
JNZ     LABEL2
```

will set the accumulator to 01_H and continue at label LABEL2.

Operation: JNZ
$(PC) \leftarrow (PC) + 2$
if $(A) \neq 0$
then $(PC) \leftarrow (PC) + rel.$

Encoding:

0 1 1 1	0 0 0 0	rel. address

Bytes: 2

Cycles: 2

JZ rel

Function: Jump if accumulator is zero

Description: If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Example: The accumulator originally contains 01_H. The instruction sequence

```
JZ      LABEL1
DEC     A
JZ      LABEL2
```

will change the accumulator to 00_H and cause program execution to continue at the instruction identified by the label LABEL2.

Operation:

```
JZ
(PC) ← (PC) + 2
if (A) = 0
then (PC) ← (PC) + rel
```

Encoding:

0 1 1 0	0 0 0 0	rel. address

Bytes: 2

Cycles: 2

LCALL addr16

Function:	Long call
Description:	LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64 Kbyte program memory address space. No flags are affected.
Example:	Initially the stack pointer equals 07_H. The label ”SUBRTN” is assigned to program memory location 1234_H. After executing the instruction LCALL SUBRTN at location 0123_H, the stack pointer will contain 09_H, internal RAM locations 08_H and 09_H will contain 26_H and 01_H, and the PC will contain 1234_H.
Operation:	LCALL (PC) ← (PC) + 3 (SP) ← (SP) + 1 ((SP)) ← (PC7-0) (SP) ← (SP) + 1 ((SP)) ← (PC15-8) (PC) ← addr15-0

Encoding:

0 0 0 1	0 0 1 0	addr15 . . addr8	addr7 . . addr0

Bytes: 3

Cycles: 2

LJMP addr16

Function:	Long jump
Description:	LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.
Example:	The label ”JMPADR” is assigned to the instruction at program memory location 1234_H. The instruction LJMP JMPADR at location 0123_H will load the program counter with 1234_H.
Operation:	LJMP (PC) ← addr15-0
Encoding:	0 0 0 0 | 0 0 1 0 addr15 . . . addr8 addr7 . . . addr0
Bytes:	3
Cycles:	2

MOV <dest-byte>, <src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30_H holds 40_H. The value of RAM location 40_H is 10_H. The data present at input port 1 is 11001010B ($0CA_H$).

```
MOV    R0, #30H        ; R0 < = 30H
MOV    A, @R0          ; A < = 40H
MOV    R1,A            ; R1 < = 40H
MOV    B, @R1          ; B < = 10H
MOV    @R1,P1          ; RAM (40H) < = 0CAH
MOV    P2,P1           ; P2 < = 0CAH
```

leaves the value 30_H in register 0, 40_H in both the accumulator and register 1, 10_H in register B, and $0CA_H$ (11001010B) both in RAM location 40_H and output on port 2.

MOV A,Rn

Operation: MOV
(A) ← (Rn)

Encoding:

1 1 1 0	1 r r r

Bytes: 1

Cycles: 1

MOV A,direct *)

Operation: MOV
(A) ← (direct)

Encoding:

1 1 1 0	0 1 0 1	direct address

Bytes: 2

Cycles: 1

*) MOV A,ACC is not a valid instruction.

MOV A,@Ri

Operation: MOV
(A) ← ((Ri))

Encoding:

1 1 1 0	0 1 1 i	

Bytes: 1

Cycles: 1

MOV A, #data

Operation: MOV
(A) ← #data

Encoding:

0 1 1 1	0 1 0 0	immediate data

Bytes: 2

Cycles: 1

MOV Rn,A

Operation: MOV
(Rn) ← (A)

Encoding:

1 1 1 1	1 r r r	

Bytes: 1

Cycles: 1

MOV Rn,direct

Operation: MOV
(Rn) ← (direct)

Encoding:

1 0 1 0	1 r r r	direct address

Bytes: 2

Cycles: 2

MOV Rn, #data

Operation: MOV
(Rn) ← #data

Encoding:

0 1 1 1	1 r r r	immediate data

Bytes: 2

Cycles: 1

MOV direct,A

Operation: MOV
(direct) ← (A)

Encoding:

1 1 1 1	0 1 0 1	direct address

Bytes: 2

Cycles: 1

MOV direct,Rn

Operation: MOV
(direct) ← (Rn)

Encoding:

1 0 0 0	1 r r r	direct address

Bytes: 2

Cycles: 2

MOV direct,direct

Operation: MOV
(direct) ← (direct)

Encoding:

1 0 0 0	0 1 0 1	dir.addr. (src)	dir.addr. (dest)

Bytes: 3

Cycles: 2

MOV direct, @ Ri

Operation: MOV
(direct) ← ((Ri))

Encoding:

1 0 0 0	0 1 1 i	direct address

Bytes: 2

Cycles: 2

MOV direct, #data

Operation: MOV
(direct) ← #data

Encoding:

0 1 1 1	0 1 0 1	direct address	immediate data

Bytes: 3

Cycles: 2

MOV @ Ri,A

Operation: MOV
((Ri)) ← (A)

Encoding:

1 1 1 1	0 1 1 i

Bytes: 1

Cycles: 1

MOV @ Ri,direct

Ooeration: MOV
((Ri)) ← (direct)

Encoding:

1 0 1 0	0 1 1 i	direct address

Bytes: 2

Cycles: 2

MOV @ Ri,#data

Operation: MOV
((Ri)) ← #data

Encoding:

0 1 1 1	0 1 1 i	immediate data

Bytes: 2

Cycles: 1

MOV <dest-bit>, <src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input port 3 is 11000101B. The data previously written to output port 1 is 35_H (00110101B).

```
MOV     P1.3,C
MOV     C,P3.3
MOV     P1.2,C
```

will leave the carry cleared and change port 1 to 39_H (00111001 B).

MOV C,bit

Operation: MOV
(C) ← (bit)

Encoding:

1 0 1 0	0 0 1 0	bit address

Bytes: 2

Cycles: 1

MOV bit,C

Operation: MOV
(bit) ← (C)

Encoding:

1 0 0 1	0 0 1 0	bit address

Bytes: 2

Cycles: 2

MOV DPTR, #data16

Function: Load data pointer with a 16-bit constant

Description: The data pointer is loaded with the 16-bit constant indicated. The 16 bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example: The instruction

MOV DPTR, $\#1234_H$

will load the value 1234_H into the data pointer: DPH will hold 12_H and DPL will hold 34_H.

Operation: MOV
(DPTR) ← #data15-0
DPH □ DPL ← #data15-8 □ #data7-0

Encoding:

1 0 0 1	0 0 0 0	immed. data 15 . . . 8	immed. data 7 . . . 0

Bytes: 3

Cycles: 2

MOVC A, @A + <base-reg>

Function: Move code byte

Description: The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added to the accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC:  INC        A
         MOVC       A, @A + PC
         RET
         DB         66H
         DB         77H
         DB         88H
         DB         99H
```

If the subroutine is called with the accumulator equal to 01_H, it will return with 77_H in the accumulator. The INC A before the MOVC instruction is needed to ”get around” the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the accumulator instead.

MOVC A, @A + DPTR

Operation: MOVC
(A) ← ((A) + (DPTR))

Encoding:

1 0 0 1	0 0 1 1

Bytes: 1

Cycles: 2

MOVC A, @A + PC

Operation:	MOVC (PC) ← (PC) + 1 (A) ← ((A) + (PC))
Encoding:	1 0 0 0 | 0 0 1 1
Bytes:	1
Cycles:	2

MOVX <dest-byte>, <src-byte>

Function: Move external

Description: The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the ”X” appended to MOV. There are two types of instructions, differing in whether they provide an eight bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instructions, the data pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 special function register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64 Kbyte), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the data pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g. an SAB 8155 RAM/I/O/timer) is connected to the SAB 80(c)5XX port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12_H and 34_H. Location 34_H of the external RAM holds the value 56_H. The instruction sequence

```
MOVX    A, @R1
MOVX    @R0,A
```

copies the value 56_H into both the accumulator and external RAM location 12_H.

MOVX A,@Ri

Operation:	MOVX (A) ← ((Ri))
Encoding:	1 1 1 0 | 0 0 1 i
Bytes:	1
Cycles:	2

MOVX A,@DPTR

Operation:	MOVX (A) ← ((DPTR))
Encoding:	1 1 1 0 | 0 0 0 0
Bytes:	1
Cycles:	2

MOVX @Ri,A

Operation:	MOVX ((Ri)) ← (A)
Encoding:	1 1 1 1 | 0 0 1 i
Bytes:	1
Cycles:	2

MOVX @DPTR,A

Operation:	MOVX ((DPTR)) ← (A)
Encoding:	1 1 1 1 | 0 0 0 0
Bytes:	1
Cycles:	2

MUL AB

Function:	Multiply
Description:	MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 ($0FF_H$) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.
Example:	Originally the accumulator holds the value 80 (50_H). Register B holds the value 160 ($0A0_H$). The instruction MUL AB will give the product 12,800 (3200_H), so B is changed to 32_H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.
Operation:	MUL (A7-0) (B15-8) ← (A) x (B)
Encoding:	1 0 1 0 | 0 1 0 0
Bytes:	1
Cycles:	4

NOP

Function: No operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence

```
CLR    P2.7
NOP
NOP
NOP
NOP
SETB  P2.7
```

Operation: NOP

Encoding:

0 0 0 0	0 0 0 0

Bytes: 1

Cycles: 1

ORL <dest-byte> <src-byte>

Function:	Logical OR for byte variables

Description: ORL performs the bitwise logical OR operation between the indicated variables, storing the results in the destination byte. No flags are affected .

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

Note:

When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the accumulator holds $0C3_H$ (11000011B) and R0 holds 55_H (01010101B) then the instruction

```
ORL     A,R0
```

will leave the accumulator holding the value $0D7_H$ (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the accumulator at run-time. The instruction

```
ORL     P1,#00110010B
```

will set bits 5, 4, and 1 of output port 1.

ORL A,Rn

Operation: ORL
$(A) \leftarrow (A) \vee (Rn)$

Encoding:	0 1 0 0	1 r r r

Bytes:	1
Cycles:	1

ORL A,direct

Operation: ORL
(A) ← (A) ∨ (direct)

Encoding: | 0 1 0 0 | 0 1 0 1 | direct address |

Bytes: 2

Cycles: 1

ORL A,@Ri

Operation: ORL
(A) ← (A) ∨ ((Ri))

Encoding: | 0 1 0 0 | 0 1 1 i |

Bytes: 1

Cycles: 1

ORL A,#data

Operation: ORL
(A) ← (A) ∨ #data

Encoding: | 0 1 0 0 | 0 1 0 0 | immediate data |

Bytes: 2

Cycles: 1

ORL direct,A

Operation: ORL
(direct) ← (direct) ∨ (A)

Encoding: | 0 1 0 0 | 0 0 1 0 | direct address |

Bytes: 2

Cycles: 1

ORL direct, #data

Operation: ORL
(direct) ← (direct) ∨ #data

Encoding:

0 1 0 0	0 0 1 1	direct address	immediate data

Bytes: 3

Cycles: 2

ORL C, <src-bit>

Function: Logical OR for bit variables

Description: Set the carry flag if the Boolean value is a logic 1; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC.7 = 1, or OV = 0:

```
MOV     C,P1.0          ; Load carry with input pin P1.0
ORL     C,ACC.7         ; OR carry with the accumulator bit 7
ORL     C,/OV           ; OR carry with the inverse of OV
```

ORL C,bit

Operation: ORL
$(C) \leftarrow (C) \vee (bit)$

Encoding:

0 1 1 1	0 0 1 0	bit address

Bytes: 2

Cycles: 2

ORL C,/bit

Operation: ORL
$(C) \leftarrow (C) \vee \neg (bit)$

Encoding:

1 0 1 0	0 0 0 0	bit address

Bytes: 2

Cycles: 2

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the stack pointer is read, and the stack pointer is decremented by one. The value read is the transfer to the directly addressed byte indicated. No flags are affected.

Example: The stack pointer originally contains the value 32_H, and internal RAM locations 30_H through 32_H contain the values 20_H, 23_H, and 01_H, respectively. The instruction sequence

```
POP     DPH
POP     DPL
```

will leave the stack pointer equal to the value 30_H and the data pointer set to 0123_H. At this point the instruction

```
POP     SP
```

will leave the stack pointer set to 20_H. Note that in this special case the stack pointer was decremented to $2F_H$ before being loaded with the value popped (20_H).

Operation: POP
(direct) $\leftarrow$ ((SP))
(SP) $\leftarrow$ (SP) – 1

Encoding:

1 1 0 1	0 0 0 0		direct address

Bytes: 2

Cycles: 2

PUSH direct

Function:	Push onto stack
Description:	The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the stack pointer. Otherwise no flags are affected.
Example:	On entering an interrupt routine the stack pointer contains 09_H. The data pointer holds the value 0123_H. The instruction sequence PUSH DPL PUSH DPH will leave the stack pointer set to $0B_H$ and store 23_H and 01_H in internal RAM locations $0A_H$ and $0B_H$, respectively.
Operation:	PUSH $(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (direct)$
Encoding:	1 1 0 0 │ 0 0 0 0 direct address
Bytes:	2
Cycles:	2

RET

Function:	Return from subroutine
Description:	RET pops the high and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.
Example:	The stack pointer originally contains the value $0B_H$. Internal RAM locations $0A_H$ and $0B_H$ contain the values 23_H and 01_H, respectively. The instruction RET will leave the stack pointer equal to the value 09_H. Program execution will continue at location 0123_H.
Operation:	RET (PC15-8) $\leftarrow$ ((SP)) (SP) $\leftarrow$ (SP) – 1 (PC7-0) $\leftarrow$ ((SP)) (SP) $\leftarrow$ (SP) – 1
Encoding:	0 0 1 0 │ 0 0 1 0
Bytes:	1
Cycles:	2

RETI

Function:	Return from interrupt
Description:	RETI pops the high and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is *not* automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower or same-level interrupt is pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.
Example:	The stack pointer originally contains the value $0B_H$. An interrupt was detected during the instruction ending at location 0122_H. Internal RAM locations $0A_H$ and $0B_H$ contain the values 23_H and 01_H, respectively. The instruction RETI will leave the stack pointer equal to 09_H and return program execution to location 0123_H.
Operation:	RETI (PC15-8) $\leftarrow$ ((SP)) (SP) $\leftarrow$ (SP) – 1 (PC7-0) $\leftarrow$ ((SP)) (SP) $\leftarrow$ (SP) – 1
Encoding:	0 0 1 1 | 0 0 1 0
Bytes:	1
Cycles:	2

RL A

Function:	Rotate accumulator left
Description:	The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.
Example:	The accumulator holds the value $0C5_H$ (11000101B). The instruction RL A leaves the accumulator holding the value $8B_H$ (10001011B) with the carry unaffected.
Operation:	RL $(An + 1) \leftarrow (An)$ n = 0-6 $(A0) \leftarrow (A7)$
Encoding:	0 0 1 0 | 0 0 1 1
Bytes:	1
Cycles:	1

RLC A

Function:	Rotate accumulator left through carry flag
Description:	The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.
Example:	The accumulator holds the value $0C5_H$ (11000101B), and the carry is zero. The instruction RLC A leaves the accumulator holding the value $8A_H$ (10001010B) with the carry set.
Operation:	RLC $(An + 1) \leftarrow (An)$ n = 0-6 $(A0) \leftarrow (C)$ $(C) \leftarrow (A7)$
Encoding:	0 0 1 1 | 0 0 1 1
Bytes:	1
Cycles:	1

RR A

Function:	Rotate accumulator right
Description:	The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.
Example:	The accumulator holds the value $0C5_H$ (11000101B). The instruction RR A leaves the accumulator holding the value $0E2_H$ (11100010B) with the carry unaffected.
Operation:	RR $(An) \leftarrow (An + 1)$ n = 0-6 $(A7) \leftarrow (A0)$
Encoding:	0 0 0 0 | 0 0 1 1
Bytes:	1
Cycles:	1

RRC A

Function:	Rotate accumulator right through carry flag
Description:	The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.
Example:	The accumulator holds the value 0C5$_H$ (11000101B), the carry is zero. The instruction RRC A leaves the accumulator holding the value 62$_H$ (01100010B) with the carry set.
Operation:	RRC (An) ← (An + 1) n=0-6 (A7) ← (C) (C) ← (A0)
Encoding:	0 0 0 1 | 0 0 1 1
Bytes:	1
Cycles:	1

SETB <bit>

Function: Set bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directiy addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output port 1 has been written with the value 34_H (00110100B). The instructions

```
SETB    C
SETB    P1.0
```

will leave the carry flag set to 1 and change the data output on port 1 to 35_H (00110101B).

SETB C

Operation: SETB
(C) $\leftarrow$ 1

Encoding:

1 1 0 1	0 0 1 1

Bytes: 1

Cycles: 1

SETB bit

Operation: SETB
(bit) $\leftarrow$ 1

Encoding:

1 1 0 1	0 0 1 0	bit address

Bytes: 2

Cycles: 1

SJMP rel

Function: Short jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Example: The label ”RELADR” is assigned to an instruction at program memory location 0123_H. The instruction

SJMP RELADR

will assemble into location 0100_H. After the instruction is executed, the PC will contain the value 0123_H.

Note:

Under the above conditions the instruction following SJMP will be at 102_H. Therefore, the displacement byte of the instruction will be the relative offset (0123_H-0102_H) = 21_H. In other words, an SJMP with a displacement of $0FE_H$ would be a one-instruction infinite loop.

Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

Encoding:

1 0 0 0	0 0 0 0		rel. address

Bytes: 2

Cycles: 2

SUBB A, <src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand). AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6 but not into bit 7, or into bit 7 but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The accumulator holds $0C9_H$ (11001001B), register 2 holds 54_H (01010100B), and the carry flag is set. The instruction

```
SUBB    A,R2
```

will leave the value 74_H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that $0C9_H$ minus 54_H is 75_H. The difference between this and the above result is due to the (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A,Rn

Operation: SUBB
(A) ← (A) – (C) – (Rn)

Bytes: 1

Cycles: 1

SUBB A,direct

Operation: SUBB
(A) ← (A) – (C) – (direct)

Encoding:

1 0 0 1	0 1 0 1	direct address

Bytes: 2

Cycles: 1

SUBB A, @ Ri

Operation: SUBB
(A) ← (A) – (C) – ((Ri))

Encoding:

1 0 0 1	0 1 1 i

Bytes: 1

Cycles: 1

SUBB A, #data

Operation: SUBB
(A) ← (A) – (C) – #data

Encoding:

1 0 0 1	0 1 0 0	immediate data

Bytes: 2

Cycles: 1

SWAP A

Function:	Swap nibbles within the accumulator
Description:	SWAP A interchanges the low and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.
Example:	The accumulator holds the value $0C5_H$ (11000101B). The instruction SWAP A leaves the accumulator holding the value $5C_H$ (01011100B).
Operation:	SWAP (A3-0) $\leftrightarrows$ (A7-4), (A7-4) $\leftarrow$ (A3-0)
Encoding:	1 1 0 0 | 0 1 0 0
Bytes:	1
Cycles:	1

XCH A, <byte>

Function: Exchange accumulator with byte variable

Description: XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20_H. The accumulator holds the value $3F_H$ (00111111B). Internal RAM location 20_H holds the value 75_H (01110101B). The instruction

XCH A, @R0

will leave RAM location 20_H holding the value $3F_H$ (00111111 B) and 75_H (01110101B) in the accumulator.

XCH A,Rn

Operation: XCH
(A) ⇆ (Rn)

Encoding:

1 1 0 0	1 r r r

Bytes: 1

Cycles: 1

XCH A,direct

Operation: XCH
(A) ⇆ (direct)

Encoding:

1 1 0 0	0 1 0 1	direct address

Bytes: 2

Cycles: 1

XCH A, @ Ri

Operation:	XCH (A) ⇆ ((Ri))
Encoding:	1 1 0 0 | 0 1 1 i
Bytes:	1
Cycles:	1

XCHD A,@Ri

Function:	Exchange digit
Description:	XCHD exchanges the low-order nibble of the accumulator (bits 3-0, generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.
Example:	R0 contains the address 20_H. The accumulator holds the value 36_H (00110110B). Internal RAM location 20_H holds the value 75_H (01110101B). The instruction XCHD A, @ R0 will leave RAM location 20_H holding the value 76_H (01110110B) and 35_H (00110101B) in the accumulator.
Operation:	XCHD (A3-0) $\leftrightarrows$ ((Ri)3-0)
Encoding:	1 1 0 1 | 0 1 1 i
Bytes:	1
Cycles:	1

XRL <dest-byte>, <src-byte>

Function: Logical Exclusive OR for byte variables

Description: XRL performs the bitwise logical Exclusive OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be accumulator or immediate data.

Note:

When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the accumulator holds $0C3_H$ (11000011B) and register 0 holds $0AA_H$ (10101010B) then the instruction

```
XRL     A,R0
```

will leave the accumulator holding the value 69_H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run-time. The instruction

```
XRL     P1,#00110001B
```

will complement bits 5, 4, and 0 of output port 1.

XRL A,Rn

Operation: XRL2
$(A) \leftarrow (A) \veebar (Rn)$

Encoding:

0 1 1 0	1 r r r

Bytes: 1

Cycles: 1

XRL A,direct

Operation: XRL
(A) ← (A) ∀ (direct)

Encoding:

0 1 1 0	0 1 0 1	direct address

Bytes: 2

Cycles: 1

XRL A, @ Ri

Operation: XRL
(A) ← (A) ∀ ((Ri))

Encoding:

0 1 1 0	0 1 1 i

Bytes: 1

Cycles: 1

XRL A, #data

Operation: XRL
(A) ← (A) ∀ #data

Encoding:

0 1 1 0	0 1 0 0	immediate data

Bytes: 2

Cycles: 1

XRL direct,A

Operation: XRL
(direct) ← (direct) ∀ (A)

Encoding:

0 1 1 0	0 0 1 0	direct address

Bytes: 2

Cycles: 1

XRL direct, #data

Operation:	XRL (direct) $\leftarrow$ (direct) $\veebar$ #data
Encoding:	0 1 1 0 | 0 0 1 1 | direct address | immediate data
Bytes:	3
Cycles:	2

Instruction Set Summary

Mnemonic	Description	Byte	Cycle
Arithmetic Operations			
ADD A,Rn	Add register to accumulator	1	1
ADD A,direct	Add direct byte to accumulator	2	1
ADD A, @Ri	Add indirect RAM to accumulator	1	1
ADD A,#data	Add immediate data to accumulator	2	1
ADDC A,Rn	Add register to accumulator with carry flag	1	1
ADDC A,direct	Add direct byte to A with carry flag	2	1
ADDC A, @Ri	Add indirect RAM to A with carry flag	1	1
ADDC A, #data	Add immediate data to A with carry flag	2	1
SUBB A,Rn	Subtract register from A with borrow	1	1
SUBB A,direct	Subtract direct byte from A with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from A with borrow	1	1
SUBB A,#data	Subtract immediate data from A with borrow	2	1
INC A	Increment accumulator	1	1
INC Rn	Increment register	1	1
INC direct	Increment direct byte	2	1
INC @Ri	Increment indirect RAM	1	1
DEC A	Decrement accumulator	1	1
DEC Rn	Decrement register	1	1
DEC direct	Decrement direct byte	2	1
DEC @Ri	Decrement indirect RAM	1	1
INC DPTR	Increment data pointer	1	2
MUL AB	Multiply A and B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal adjust accumulator	1	1

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Logic Operations			
ANL A,Rn	AND register to accumulator	1	1
ANL A,direct	AND direct byte to accumulator	2	1
ANL A,@Ri	AND indirect RAM to accumulator	1	1
ANL A,#data	AND immediate data to accumulator	2	1
ANL direct,A	AND accumulator to direct byte	2	1
ANL direct,#data	AND immediate data to direct byte	3	2
ORL A,Rn	OR register to accumulator	1	1
ORL A,direct	OR direct byte to accumulator	2	1
ORL A,@Ri	OR indirect RAM to accumulator	1	1
ORL A,#data	OR immediate data to accumulator	2	1
ORL direct,A	OR accumulator to direct byte	2	1
ORL direct,#data	OR immediate data to direct byte	3	2
XRL A,Rn	Exclusive OR register to accumulator	1	1
XRL A direct	Exclusive OR direct byte to accumulator	2	1
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1	1
XRL A,#data	Exclusive OR immediate data to accumulator	2	1
XRL direct,A	Exclusive OR accumulator to direct byte	2	1
XRL direct,#data	Exclusive OR immediate data to direct byte	3	2
CLR A	Clear accumulator	1	1
CPL A	Complement accumulator	1	1
RL A	Rotate accumulator left	1	1
RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within the accumulator	1	1

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Data Transfer			
MOV A,Rn	Move register to accumulator	1	1
MOV A,direct *)	Move direct byte to accumulator	2	1
MOV A,@Ri	Move indirect RAM to accumulator	1	1
MOV A,#data	Move immediate data to accumulator	2	1
MOV Rn,A	Move accumulator to register	1	1
MOV Rn,direct	Move direct byte to register	2	2
MOV Rn,#data	Move immediate data to register	2	1
MOV direct,A	Move accumulator to direct byte	2	1
MOV direct,Rn	Move register to direct byte	2	2
MOV direct,direct	Move direct byte to direct byte	3	2
MOV direct,@Ri	Move indirect RAM to direct byte	2	2
MOV direct,#data	Move immediate data to direct byte	3	2
MOV @Ri,A	Move accumulator to indirect RAM	1	1
MOV @Ri,direct	Move direct byte to indirect RAM	2	2
MOV @Ri, #data	Move immediate data to indirect RAM	2	1
MOV DPTR, #data16	Load data pointer with a 16-bit constant	3	2
MOVC A,@A + DPTR	Move code byte relative to DPTR to accumulator	1	2
MOVC A,@A + PC	Move code byte relative to PC to accumulator	1	2
MOVX A,@Ri	Move external RAM (8-bit addr.) to A	1	2
MOVX A,@DPTR	Move external RAM (16-bit addr.) to A	1	2
MOVX @Ri,A	Move A to external RAM (8-bit addr.)	1	2
MOVX @DPTR,A	Move A to external RAM (16-bit addr.)	1	2
PUSH direct	Push direct byte onto stack	2	2
POP direct	Pop direct byte from stack	2	2
XCH A,Rn	Exchange register with accumulator	1	1
XCH A,direct	Exchange direct byte with accumulator	2	1
XCH A,@Ri	Exchange indirect RAM with accumulator	1	1
XCHD A,@Ri	Exchange low-order nibble indir. RAM with A	1	1

*) MOV A,ACC is not a valid instruction

Instruction Set Summary (cont'd)

Mnemonic		Description	Byte	Cycle
Boolean Variable Manipulation				
CLR	C	Clear carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set carry flag	1	1
SETB	bit	Set direct bit	2	1
CPL	C	Complement carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to carry flag	2	2
ANL	C,/bit	AND complement of direct bit to carry	2	2
ORL	C,bit	OR direct bit to carry flag	2	2
ORL	C,/bit	OR complement of direct bit to carry	2	2
MOV	C,bit	Move direct bit to carry flag	2	1
MOV	bit,C	Move carry flag to direct bit	2	2
Program and Machine Control				
ACALL	addr11	Absolute subroutine call	2	2
LCALL	addr16	Long subroutine call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute jump	2	2
LJMP	addr16	Long iump	3	2
SJMP	rel	Short jump (relative addr.)	2	2
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if accumulator is zero	2	2
JNZ	rel	Jump if accumulator is not zero	2	2
JC	rel	Jump if carry flag is set	2	2
JNC	rel	Jump if carry flag is not set	2	2
JB	bit,rel	Jump if direct bit is set	3	2
JNB	bit,rel	Jump if direct bit is not set	3	2
JBC	bit,rel	Jump if direct bit is set and clear bit	3	2
CJNE	A,direct,rel	Compare direct byte to A and jump if not equal	3	2

Instruction Set Summary (cont'd)

Mnemonic	Description	Byte	Cycle
Program and Machine Control (cont'd)			
CJNE A,#data,rel	Compare immediate to A and jump if not equal	3	2
CJNE Rn,#data rel	Compare immed. to reg. and jump if not equal	3	2
CJNE @Ri,#data,rel	Compare immed. to ind. and jump if not equal	3	2
DJNZ Rn,rel	Decrement register and jump if not zero	2	2
DJNZ direct,rel	Decrement direct byte and jump if not zero	3	2
NOP	No operation	1	1