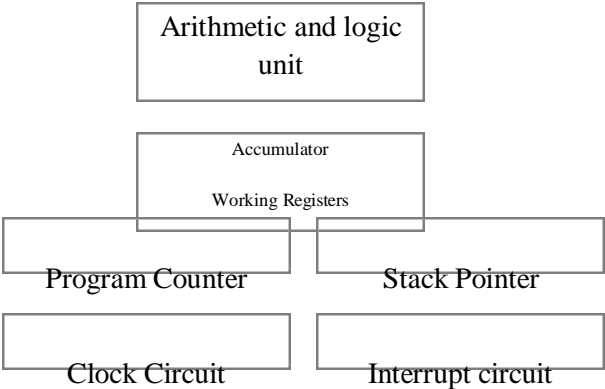
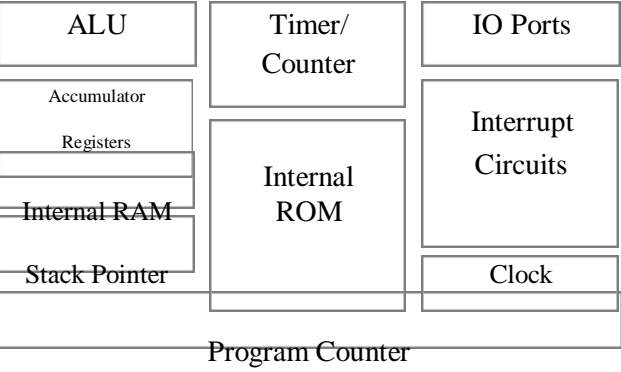


(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

UNIT-1

1.1 MICROPROCESSORS AND MICROCONTROLLERS

<i>Microprocessor</i>	<i>Microcontroller</i>
	
<i>Block diagram of microprocessor</i>	<i>Block diagram of microcontroller</i>
Microprocessor contains ALU, General purpose registers, stack pointer, program counter, clock timing circuit, interrupt circuit	Microcontroller contains the circuitry of microprocessor, and in addition it has built in ROM, RAM, I/O Devices, Timers/Counters etc.
It has many instructions to move data between memory and CPU	It has few instructions to move data between memory and CPU
Few bit handling instruction	It has many bit handling instructions
Less number of pins are multifunctional	More number of pins are multifunctional
Single memory map for data and code (program)	Separate memory map for data and code (program)
Access time for memory and IO are more	Less access time for built in memory and IO.
Microprocessor based system requires additional hardware	It requires less additional hardware
More flexible in the design point of view	Less flexible since the additional circuits which is residing inside the microcontroller is fixed for a particular microcontroller
Large number of instructions with flexible addressing modes	Limited number of instructions with few addressing modes

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

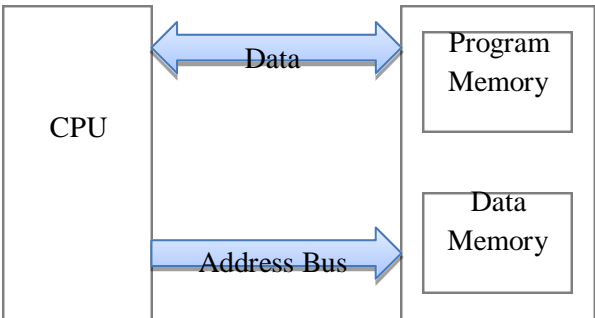
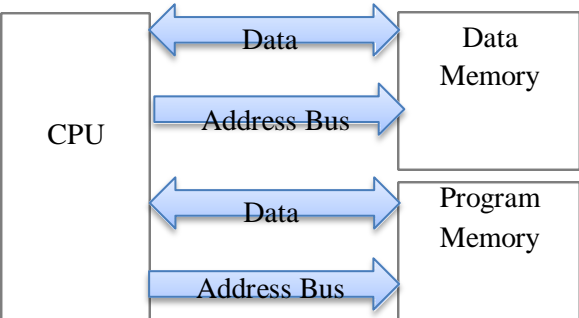
1.2. RISC AND CISC CPU ARCHITECTURES

Microcontrollers with small instruction set are called reduced instruction set computer (RISC) machines and those with complex instruction set are called complex instruction set computer (CISC). Intel 8051 is an example of CISC machine whereas microchip PIC 18F87X is an example of RISC machine.

RISC	CISC
Instruction takes one or two cycles	Instruction takes multiple cycles
Only load/store instructions are used to access memory	In additions to load and store instructions, memory access is possible with other instructions also.
Instructions executed by hardware	Instructions executed by the micro program
Fixed format instruction	Variable format instructions
Few addressing modes	Many addressing modes
Few instructions	Complex instruction set
Most of the have multiple register banks	Single register bank
Highly pipelined	Less pipelined
Complexity is in the compiler	Complexity in the microprogram

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

1.2. HARVARD & VON- NEUMANN CPU ARCHITECTURE

Von-Neumann (Princeton architecture)	Harvard architecture
	
Von-Neumann (Princeton architecture)	Harvard architecture
It uses single memory space for both instructions and data.	It has separate program memory and data memory
It is not possible to fetch instruction code and data	Instruction code and data can be fetched simultaneously
Execution of instruction takes more machine cycle	Execution of instruction takes less machine cycle
Uses CISC architecture	Uses RISC architecture
Instruction pre-fetching is a main feature	Instruction parallelism is a main feature
Also known as control flow or control driven computers	Also known as data flow or data driven computers
Simplifies the chip design because of single memory space	Chip design is complex due to separate memory space
Eg. 8085, 8086, MC6800	Eg. General purpose microcontrollers, special DSP chips etc.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

COMPUTER SOFTWARE

A set of instructions written in a specific sequence for the computer to solve a specific task is called a program and software is a collection of such programs.

The program stored in the computer memory in the form of binary numbers is called machine instructions. The *machine language* program is called *object code*.

An *assembly language* is a mnemonic representation of machine language. Machine language and assembly language are low level languages and are processor specific.

The assembly language program the programmer enters is called *source code*. The source code (assembly language) is translated to object code (machine language) using *assembler*.

Programs can be written in *high level languages* such as C, C++ etc. High level language will be converted to machine language using *compiler or interpreter*. Compiler reads the entire program and translate into the object code and then it is executed by the processor. Interpreter takes one statement of the high level language as input and translate it into object code and then executes.

Introduction to Embedded Systems

- An embedded system is an electronic system, which includes a single chip microcomputers (Microcontrollers) like the PIC.
- It is configured to perform a specific dedicated application.
- Software is programmed into the on chip ROM of the single chip computer. This software is not accessible to the user and software solves only a limited range of problems.
- Here the microcomputer is embedded or hidden inside the system. Every embedded microcomputer system, accepts inputs, performs computations, and generates outputs and runs in “real time.”

For Example, a typical automobile now a days contains an average of ten microcontrollers. In fact, modern houses may contain as many as 150 microcontrollers and on average a consumer now interacts with microcontrollers up to 300 times a day. General areas that employ embedded systems covers every branch of day to day science and technology, namely Communications, automotive, military, medical, consumer, machine control etc...

Ex: Cell phone , Digital camera , Microwave Oven, MP3 player, Portable digital

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

assistant & automobile antilock brake system etc.

Components of embedded system:

An embedded system has three components:

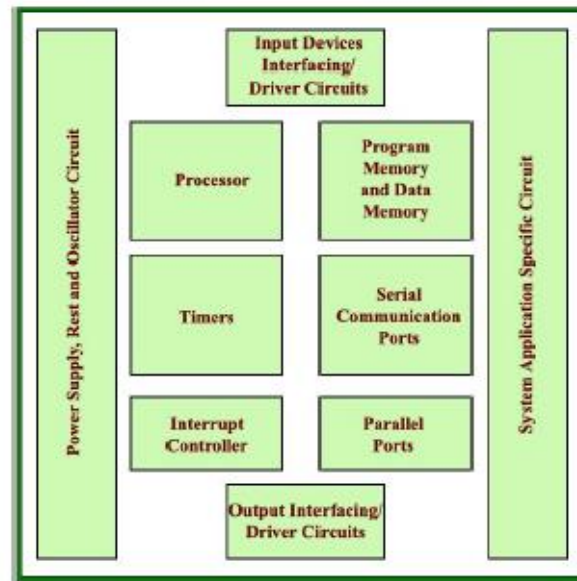
1. Hardware.
2. Application software.

This may perform concurrently the series of tasks or multiple tasks.

3. Real Time Operating system (RTOS) that supervises the application software and provide mechanism to let the processor run a process as per scheduling by following a plan to control the latencies. RTOS defines the way the system works. It sets the rules during the execution of application program. A small scale embedded system may not have RTOS.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

Hardware:



Embedded System hardware

Processor

- A Processor is the heart of the Embedded System.
- The main criteria for the processor are the processing power needed to perform the tasks within the system.

Processors can be of the following categories:

- General Purpose Processor (GPP)
- Microprocessor
- Microcontroller
- Embedded Processor
- Digital Signal Processor
- Media Processor
- Application Specific System Processor (ASSP)
- Application Specific Instruction Processors (ASIPs)

Power Source

Three possible methods of providing power to an embedded system are

- System own supply with separate supply rails for IOs, clock, basic processor and memory.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

- Supply from a system to which the embedded system interfaces, for example in a network card.
- Charge pump concept used in a system of little power needs, for examples, in the mouse or contact-less smart card.

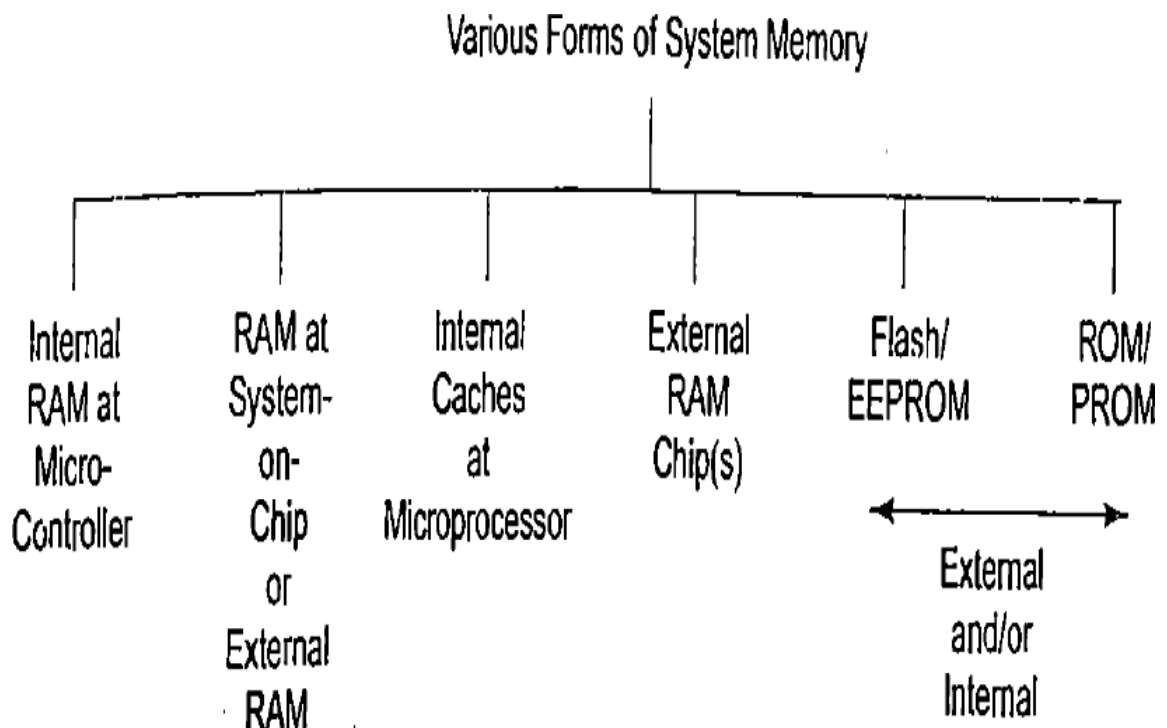
Clocking Circuits

- The clock controls the time for executing an instruction. The clock controls the various clocking requirements of CPU, Timer etc
- For this, clocking circuit provide highly stable clock pulses.

Memory

- An embedded system uses different types of memory modules for a wide range of tasks such as storage of software code and instructions for hardware.
- There are different varieties of memories in embedded system, each having their own particular mode of operation.
- An efficient memory increases the performance of embedded systems.

Various forms system memory:



(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

a. Functions Assigned to the ROM or EPROM or Flash

1. Storing 'Application' program from where the processor fetches the instruction codes
2. Storing codes for system booting, initializing, Initial input data and Strings.
3. Storing Codes for RTOS.
4. Storing Pointers (addresses) of various service routines.

b. Functions Assigned to the Internal, External and Buffer RAM

1. Storing the variables during program run,
2. Storing the stacks,
3. Storing input or output buffers for example, for speech or image .

c. Functions Assigned to the EEPROM or Flash

Storing non-volatile results of processing.

d. Functions Assigned to the Caches

1. Storing copies of the instructions, data and branch-transfer instructions in advance from external memories and
2. Storing temporarily the results in write back caches during fast processing

Timer

- Embedded systems often require mechanisms for counting the occurrence of events and for performing tasks at regular intervals.
- Embedded processors are often equipped with hardware support for this functionality.
- Timer is a device, which counts the input at regular interval using clock pulses at its input.
- The count increment on each pulse and store in a register, called count register
- Timer is used for generating delay and for generating waveforms with specific delay.

Serial Port

- A **serial port** is a serial communication interface through which information transfers in or out one bit at a time.
- Serial data transmission is much more common in new communication protocols due to a reduction in the I/O pin count, hence a reduction in cost.
- Common serial protocols include UART, SPI, SCI and I²C etc

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

- In most of the embedded systems at least two serial ports are provided.

Parallel Port

- A **parallel port** is a type of interface found on computers or embedded systems for connecting peripherals.
- The name refers to the way the data is sent; parallel ports send multiple bits of data at once.
- Parallel ports require multiple data lines in their cables and port connectors, and tend to be larger than contemporary serial ports.

Interrupt Controller

- An *interrupt* is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.
- Interrupts allow an embedded system to respond to multiple real world events in rapid time.
- By managing the interaction with external systems through effective use of interrupts can dramatically improve system efficiency and the use of processing resources.
- In an embedded system there are usually multiple interrupt sources. These interrupt sources share a single pin. The sharing is controlled by a piece of hardware called an **interrupt controller** that allows individual interrupts to be either enabled or disabled.

System Application Specific Circuit

- These are the dedicated circuits for the implementation of the application of particular system.
- This may vary from one system to other.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

PIC Microcontrollers

Introduction to PIC Microcontrollers:

PIC stands for Peripheral Interface Controller given by Microchip Technology to identify its single-chip microcontrollers. These devices have been very successful in 8-bit microcontrollers. The main reason is that Microchip Technology has continuously upgraded the device architecture and added needed peripherals to the microcontroller to suit customers' requirements. The architectures of various PIC microcontrollers can be divided as follows.

Low - end PIC Architectures:

Microchip PIC microcontrollers are available in various types. When PIC microcontroller MCU was first available from General Instruments in early 1980's, the microcontroller consisted of a simple processor executing 12-bit wide instructions with basic I/O functions. These devices are known as low-end architectures. They have limited program memory and are meant for applications requiring simple interface functions and small program & data memories. Some of the low-end device numbers are

12C5XX
16C5X
16C505

Mid range PIC Architectures

Mid range PIC architectures are built by upgrading low-end architectures with more number of peripherals, more number of registers and more data/program memory. Some of the mid-range devices are

16C6X
16C7X
16F87X

Program memory type is indicated by an alphabet. C = EPROM

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

F = Flash

RC = Mask ROM

Popularity of the PIC microcontrollers is due to the following factors.

1. Speed: Harvard Architecture, RISC architecture, 1 instruction cycle = 4 clock cycles.
2. Instruction set simplicity: The instruction set consists of just 35 instructions (as opposed to 111 instructions for 8051).
3. Power-on-reset and brown-out reset. Brown-out-reset means when the power supply goes below a specified voltage (say 4V), it causes PIC to reset; hence malfunction is avoided. A watch dog timer (user programmable) resets the processor if the software/program ever malfunctions and deviates from its normal operation.
4. PIC microcontroller has four optional clock sources.
 - Low power crystal
 - Mid range crystal
 - High range crystal
 - RC oscillator (low cost).
5. Programmable timers and on-chip ADC.
6. Up to 12 independent interrupt sources.
7. Powerful output pin control (25 mA (max.) current sourcing capability per pin.)
8. EPROM/OTP/ROM/Flash memory option.
9. I/O port expansion capability.
10. Free assembler and simulator support from Microchip at www.microchip.com

CPU Architecture: The CPU uses Harvard architecture with separate Program and Variable (data) memory interface. This facilitates instruction fetch and the operation on data/accessing of variables simultaneously.

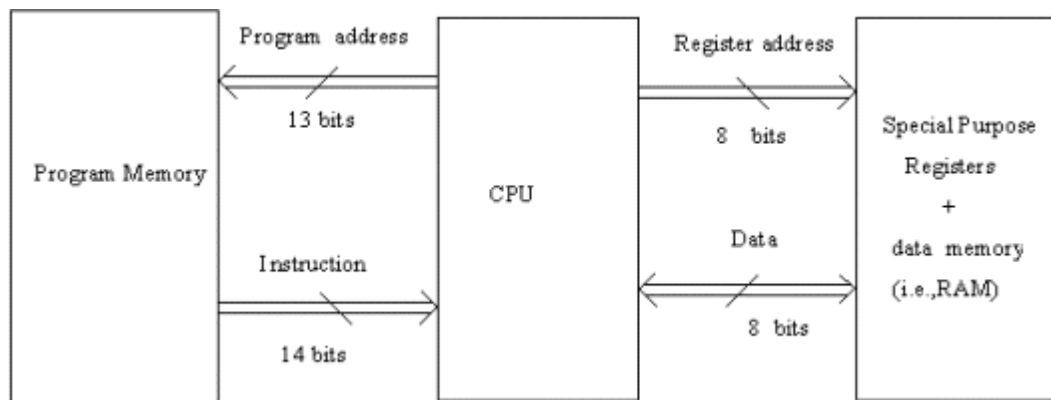


Fig : CPU Architecture of PIC microcontroller

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

PIC Memory Organisation:

PIC microcontroller has 13 bits of program memory address. Hence it can address up to 8k of program memory. The program counter is 13-bit. PIC 16C6X or 16C7X program memory is 2k or 4k. While addressing 2k of program memory, only 11- bits are required. Hence two most significant bits of the program counter are ignored. Similarly, while addressing 4k of memory, 12 bits are required. Hence the MSb of the program counter is ignored.

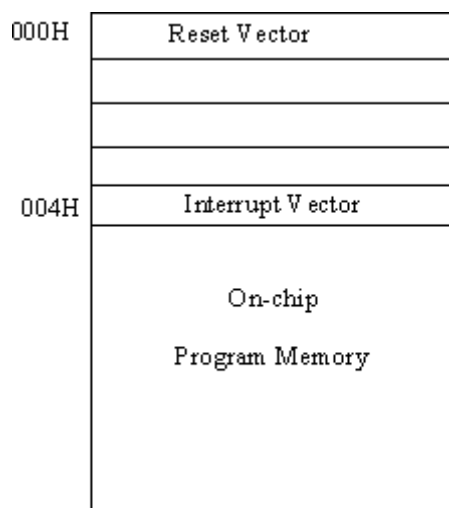


Fig : Program Memory map

The program memory map of PIC16C74A is shown in Fig 16.2. On reset, the program counter is cleared and the program starts at 00H. Here a 'goto' instruction is required that takes the processor to the mainline program.

When a peripheral interrupt, that is enabled, is received, the processor goes to 004H. A suitable branching to the interrupt service routine (ISR) is written at 004H.

Data memory (Register Files): Data Memory is also known as Register File. Register File consists of two components.

1. General purpose register file (same as RAM).
2. Special purpose register file (similar to SFR in 8051).

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

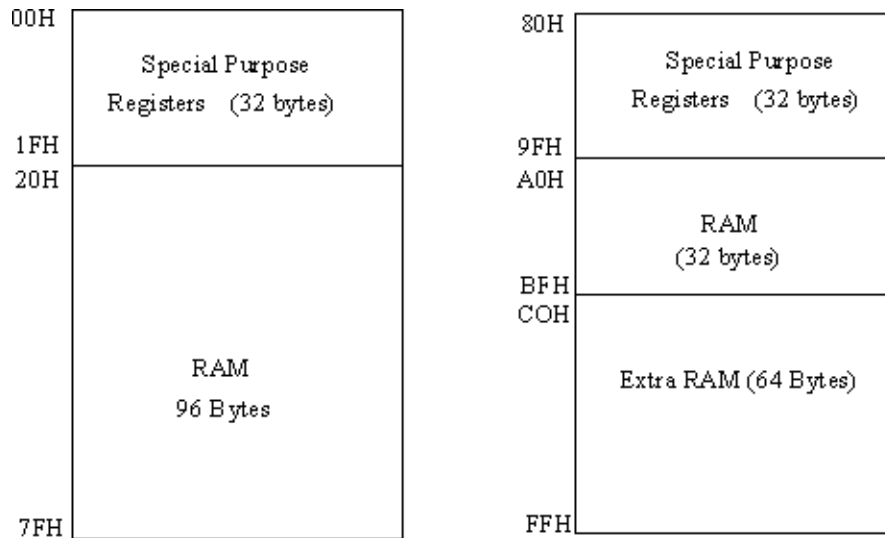


Fig : Data Memory map

The special purpose register file consists of input/output ports and control registers. Addressing from 00H to FFH requires 8 bits of address. However, the instructions that use direct addressing modes in PIC to address these register files use 7 bits of instruction only. Therefore the register bank select (RP0) bit in the STATUS register is used to select one of the register banks.

In indirect addressing FSR register is used as a pointer to anywhere from 00H to FFH in the data memory.

Basic Architecture of PIC Microcontrollers

Specifications of some popular PIC microcontrollers are as follows:

Device	Program Memory (14bits)	Data RAM (bytes)	I/O Pins	ADC	Timers 8/16 bits	CCP (PWM)	USART SPI / I ² C
16C74A	4K EPROM	192	33	8 bits x 8 channels	2/1	2	USART SPI / I ² C
16F877	8K Flash	368 (RAM) 256 (EEPROM)	33	10 bits x 8 channels	2/1	2	USART SPI

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

							/ °C
--	--	--	--	--	--	--	------

Device	Interrupt Sources	Instruction Set
16C74A	12	35
16F877	15	35

PIC Microcontroller Clock

Most of the PIC microcontrollers can operate upto 20MHz. One instructions cycle (machine cycle) consists of four clock cycles.

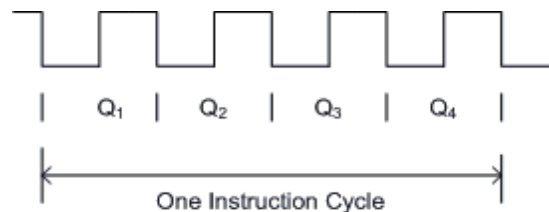


Fig : Relation between instruction cycles and clock cycles for PIC microcontrollers

Instructions that do not require modification of program counter content get executed in one instruction cycle.

Although the architectures of various midrange 8 - bit PIC microcontroller are not the same, the variation is mostly interns of addition of memory and peripherals. We will discuss here the architecture of a standard mid-range PIC microcontroller, 16C74A. Unless mentioned otherwise, the information given here is for a PIC 16C74A microcontroller Chip. Architecture of PIC16C74A

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

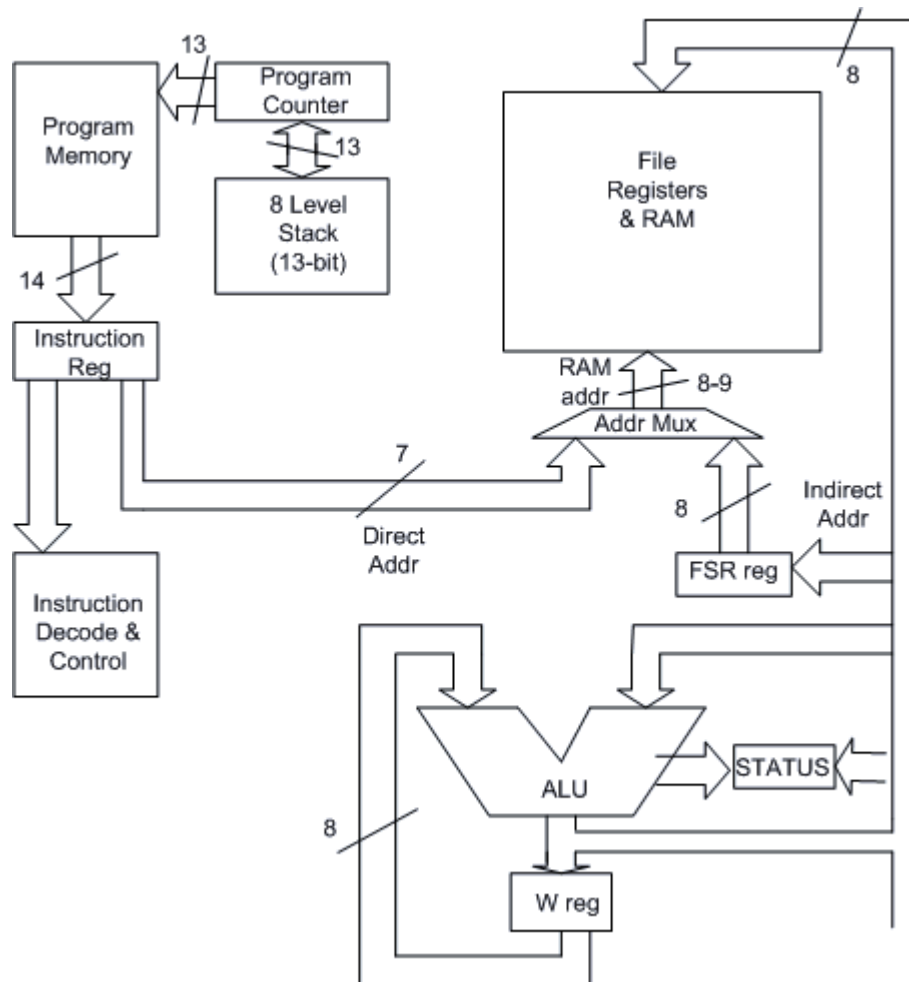


Fig : Basic Architecture of PIC 16C74A

The basic architecture of PIC16C74A is shown in fig 17.2. The architecture consists of Program memory, file registers and RAM, ALU and CPU registers. It should be noted that the program Counter is 13 - bit and the program memory is organised as 14 - bit word. Hence the program Memory capacity is 8k x 14 bit. Each instruction of PIC 16C74A is 14 - bit long. The various CPU registers are discussed here.

CPU registers (registers commonly used by the CPU)

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.



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

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

automatically loaded to the upper 5bits of the program counter, as shown in the figure.

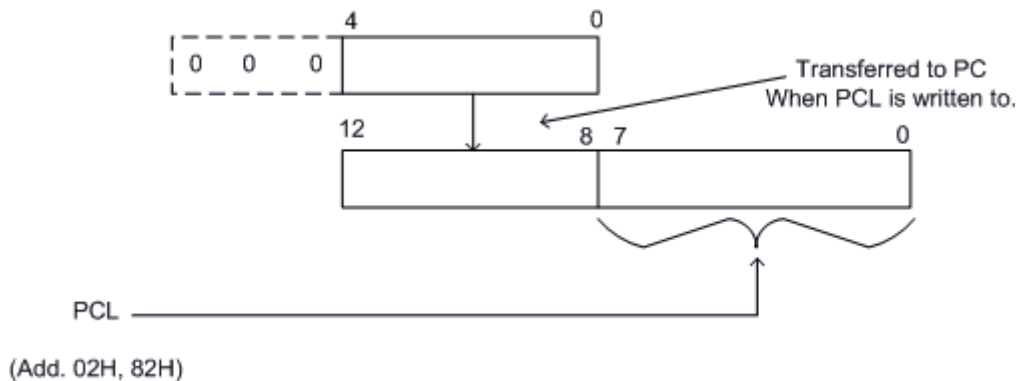


Fig : Schematic of how PCL is loaded from PCLATH

Program Counter Stack

An independent 8-level stack is used for the program counter. As the program counter is 13bit, the stack is organized as 8x13bit registers. When an interrupt occurs, the program counter is pushed onto the stack. When the interrupt is being serviced, 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.

Register File Map

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

00	INDF	INDF	80
01	TMR0	OPTION	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	FSR	FSR	84
05	PORTA	TRISA	85
06	PORTB	TRISB	86
07	PORTC	TRISC	87
08	PORTD	TRISD	88
09	PORTE	TRISE	89
0A	PCLATH	PCLATH	8A
0B	INTCON	INTCON	8B
0C	PIR1	PIE1	8C
0D	PIR2	PIE2	8D
0E	TMR1L	PCON	8E
0F	TMR1H	.	8F
10	T1CON	.	90
11	TRM2	.	91
12	T2CON	PR2	92
13	SSPBUF	SSPADD	93
14	SSPCON	SSPSTAT	94
15	CCPR1L	.	95
16	CCPR1H	.	96
17	CCP1CON	.	97
18	RCSTA	TXSTA	98
19	TXREG	SPBRG	99
1A	RCREG	.	9A
1B	CCPR2L	.	9B
1C	CCPR2H	.	9C
1D	CCP2CON	.	9D
1E	ADRES	.	9E
1F	ADCON0	ADCON1	9F
20	General Purpose RAM	General Purpose RAM	A0
7F			
Bank - 0		Bank - 1	

Fig : Register File Map

It can be noted that some of the special purpose registers are available both in Bank-0 and Bank-1. These registers have the same value in both banks. Changing the register content in one bank automatically changes its content in the other bank.

Port Structure and Pin Configuration of PIC 16C74A

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

As mentioned earlier, there is a large variety of PIC microcontrollers. However, the midrange architectures are widely used. Our discussion will mainly confine to PIC16C74A whose architecture has most of the required features of a mid-range PIC microcontroller. Study of any other mid-range PIC microcontroller will not cause much variation from the basic architecture of PIC 16C74A ..

PIC 16C74A has 5 I/O Ports. Each port is a bidirectional I/O port. In addition, they have the following alternate functions.

Port	Alternative uses of I/O pins	No. of I/O pins
Port A	A/D Converter 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
Total I/O pins		33
Total pins		40

In addition to I/O pins, there is a Master clear pin (MCLR) which is equivalent to reset in 8051. However, unlike 8051, MCLR should be pulled low to reset the micro controller. Since PIC16C74A has inherent power-on reset, no special connection is required with MCLR pin to reset the micro controller on power-on.

There are two VDD pins and two VSS pins. There are two pins (OSC1 and OSC2) for connecting the crystal oscillator/ RC oscillator. Hence the total number of pins with a 16C74A is $33+7=40$. This IC is commonly available in a dual-in-pin (DIP) package.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

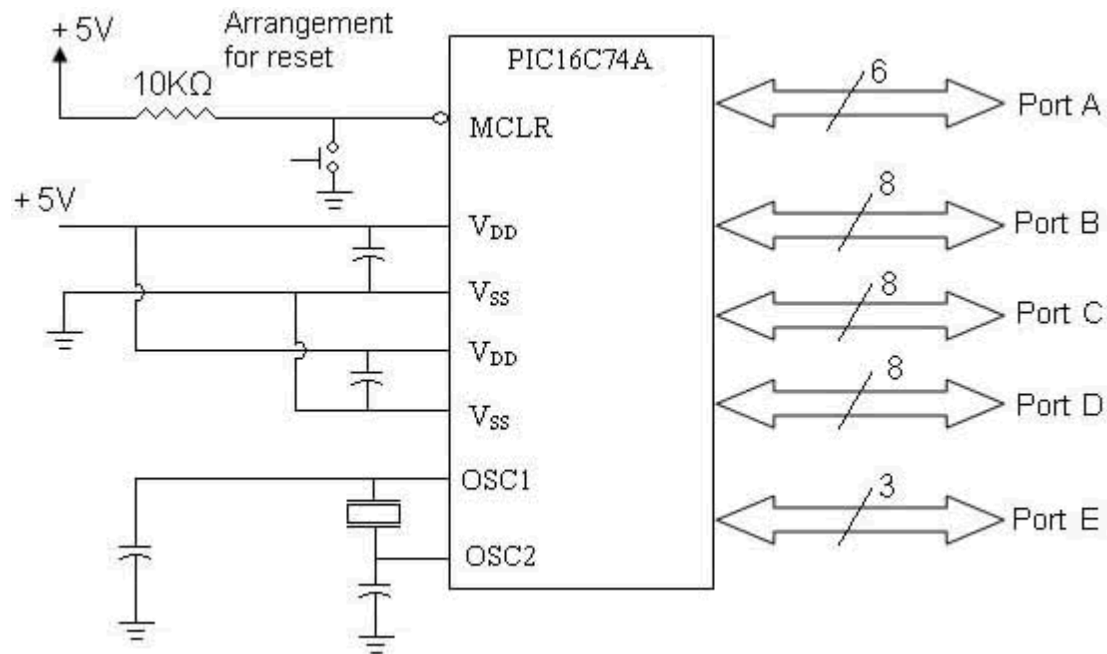


Fig : Pin configuration of PIC 16C74A

Instruction Set of PIC Microcontroller

Guidelines from Microchip Technology

For writing assembly language program Microchip Technology has suggested the following guidelines.

1. Write instruction mnemonics in lower case. (e.g., movwf)
2. Write the special register names, RAM variable names and bit names in upper case. (e.g., PCL, RP0, etc.)
3. Write instructions and subroutine labels in mixed case. (e.g., Mainline, LoopTime)

Instruction Set:

The instruction set for PIC16C74A consists of only 35 instructions. Some of these instructions are byte oriented instructions and some are bit oriented instructions.

The **byte oriented instructions** that require two parameters (For example, movf f, F(W)) expect the f to be replaced by the name of a special purpose register (e.g., PORTA) or the name of a RAM variable (e.g., NUM1), which serves as the source of the operand. 'f' stands for file register. The F(W) parameter is the destination of the result of the operation. It should be replaced by:

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

F, if the destination is to be the source register.

W, if the destination is to be the working register (i.e., Accumulator or W register).

The **bit oriented instructions** also expect parameters (e.g., btfsc f, b). Here 'f' is to be replaced by the name of a special purpose register or the name of a RAM variable. The 'b' parameter is to be replaced by a bit number ranging from 0 to 7.

For example:

```
Z equ 2
btfsc STATUS, Z
```

Z has been equated to 2. Here, the instruction will test the Z bit of the STATUS register and will skip the next instruction if Z bit is clear.

The **literal instructions** require an operand having a known value (e.g., 0AH) or a label that represents a known value.

For example:

```
NUM equ 0AH ;           Assigns 0AH to the label NUM ( a constant
) movlw NUM ;           will move 0AH to the W register.
```

Every instruction fits in a single 14-bit word. In addition, every instruction also executes in a single cycle, unless it changes the content of the Program Counter. These features are due to the fact that PIC micro controller has been designed on the principles of RISC (Reduced Instruction Set Computer) architecture.

Instruction set:

Mnemonics	Description	Instruction Cycles
bcf f, b	Clear bit b of register f	1
bsf f, b	Set bit b of register f	1
Clrw	Clear working register W	1
clrf f	Clear f	1
movlw k	Move literal 'k' to W	1
movwf f	Move W to f	1
movf f, F(W)	Move f to F or W	1
swaph f, F(W)	Swap nibbles of f, putting result in F or W	1

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

andlw k	And literal value into W	1
andwf f, F(W)	And W with F and put the result in W or F	1
andwf f, F(W)	And W with F and put the result in W or F	1
iorlw k	inclusive-OR literal value into W	1

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

iorwf f, F(W)	inclusive-OR W with f and put the result in F or W	1
xorlw k	Exclusive-OR literal value into W	1
xorwf f, F(W)	Exclusive-OR W with f and put the result in F or W	1
addlw k	Add the literal value to W and store the result in W	1
addwf f, F(W)	Add W to f and store the result in F or W	1
sublw k	Subtract the literal value from W and store the result in W	1
subwf f, F(W)	Subtract f from W and store the result in F or W	1
rlf f, F(W)	Copy f into F or W; rotate F or W left through the carry bit	1
rrf f, F(W)	Copy f into F or W; rotate F or W right through the carry bit	1
btfsc f, b	Test 'b' bit of the register f and skip the next instruction if bit is clear	1 / 2
btfss f, b	Test 'b' bit of the register f and skip the next instruction if bit is set	1 / 2
decfsz f, F(W)	Decrement f and copy the result to F or W; skip the next instruction if the result is zero	1 / 2
incfz f, F(W)	Increment f and copy the result to F or W; skip the next instruction if the result is zero	1 / 2
goto label	Go to the instruction with the label "label"	2
call label	Go to the subroutine "label", push the Program Counter in the stack	2
Retrun	Return from the subroutine, POP the Program Counter from the stack	2
retlw k	Retrun from the subroutine, POP the Program Counter from the stack; put k in W	2
Retie	Return from Interrupt Service Routine and re-enable interrupt	2
Clrwdt	Clear Watch Dog Timer	1
Sleep	Go into sleep/ stand by mode	1
Nop	No operation	1

Encoding of instruction:

As has been discussed, each instruction is of 14-bit long. These 14-bits contain both op-code and the operand. Some examples of instruction encoding are shown here.

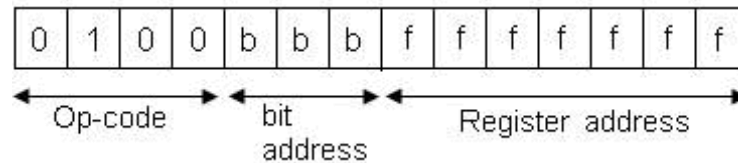
Example-1:

bcf f, b Clear 'b' bit of register 'f'

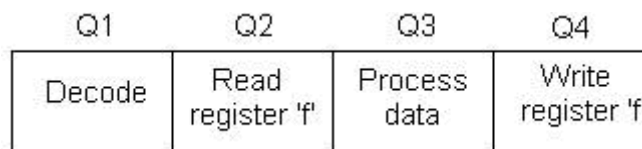
(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

Operands: $0 \leq f \leq 127$
 $0 \leq b \leq 7$

Encoding:



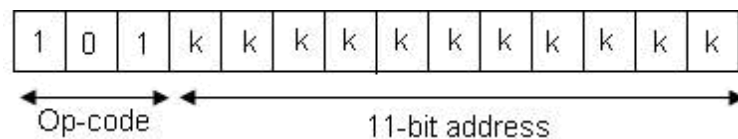
The instruction is executed in one instruction cycle, i.e., 4 clock cycles. The activities in various clock cycles are as follows.



Example-2:

goto K Go to label 'k' instruction

Operand: $0 \leq K \leq 2047$ (11-bit address is specified) Operation: $K \longrightarrow PC$
 $\langle 10:0 \rangle PCLATH \langle 4:3 \rangle \longrightarrow PC \langle 12:11 \rangle$ Encoding:



Since this instruction requires modification of program Counter, it takes two instruction cycles for execution.

Q-Cycle activities are shown as follows.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

	Q1	Q2	Q3	Q4
1st instruction cycle	Decode	Read literal 'k'	Process data	Write to PC
2nd Instruction cycle	No-Operation	No-Operation	No-Operation	No-Operation

Addressing Modes of pic microcontroller

To know the working principal and data handling, we need to have clear knowledge on addressing modes of pic microcontroller. Now we can see that how we can categorise different addressing modes of pic microcontroller. In PIC micro controller, it having mainly five addressing modes. Those are

1. Immediate addressing mode
2. Register operand addressing mode
3. Memory operand addressing mode
4. Direct addressing
5. Indirect addressing.

1. Immediate addressing mode:

In this addressing mode, the operand is a number or constant not an address as **MOVLW 43h**, the operand here is data not address. So in this addressing mode of pic microcontroller data is directly transfer. And data is immediate after the opcode. That is why this type of addressing is called immediate addressing. This way is fast in execution.

2. Register operand addressing mode:

In this addressing mode, the operand is a Register which holds the data to be execute. Register operand addressing mode deals with the registers like: CLR W

3. Memory operand addressing mode :

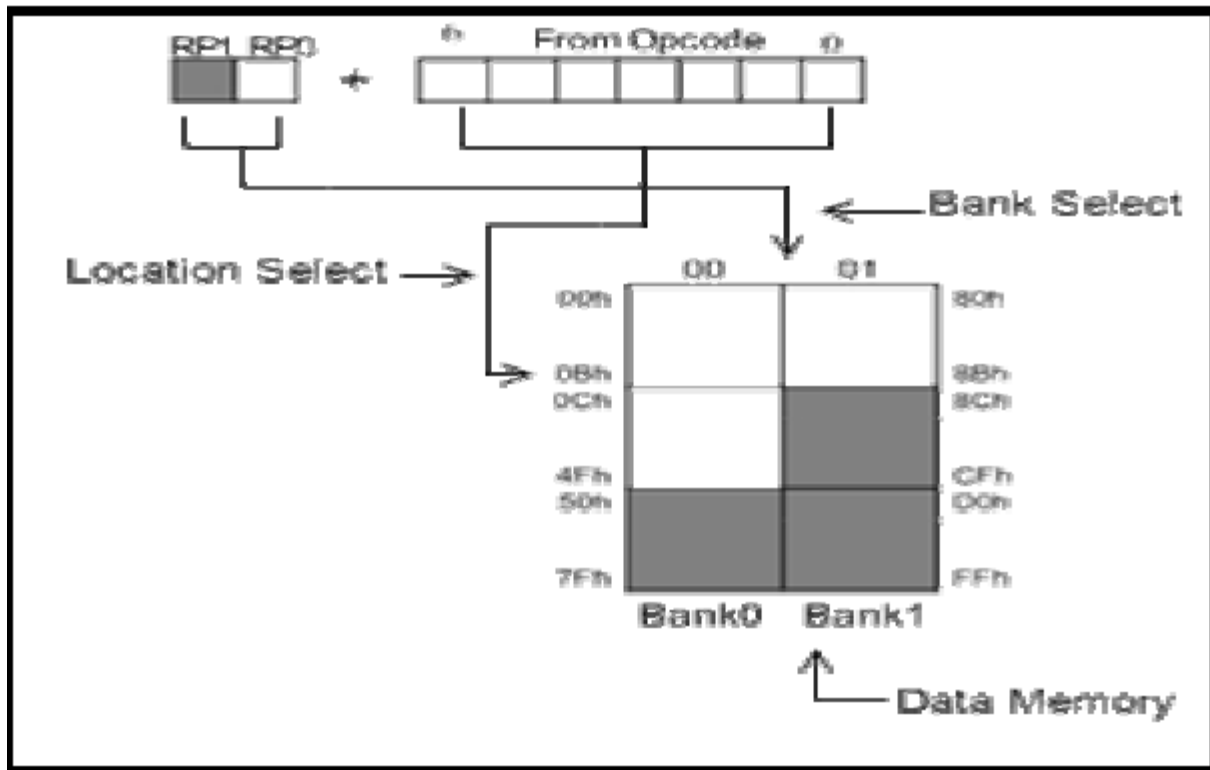
In this addressing mode, the operand is an address of Memory location which holds the data to be execute. Again memory operand addressing mode is under two category

- A). Direct addressing like CLRF 13h. We deal with the address or the memory location.
- B). Indirect addressing. we use in it INDF and FSR registers.

4. Direct addressing:

Direct Addressing is done through a 9-bit address. This address is obtained by connecting 7th bit of direct address. By using an instruction with two bits (RP1, RP0) from STATUS register. this is shown on bellow Figure . Any access to SFR registers can be an example of direct addressing

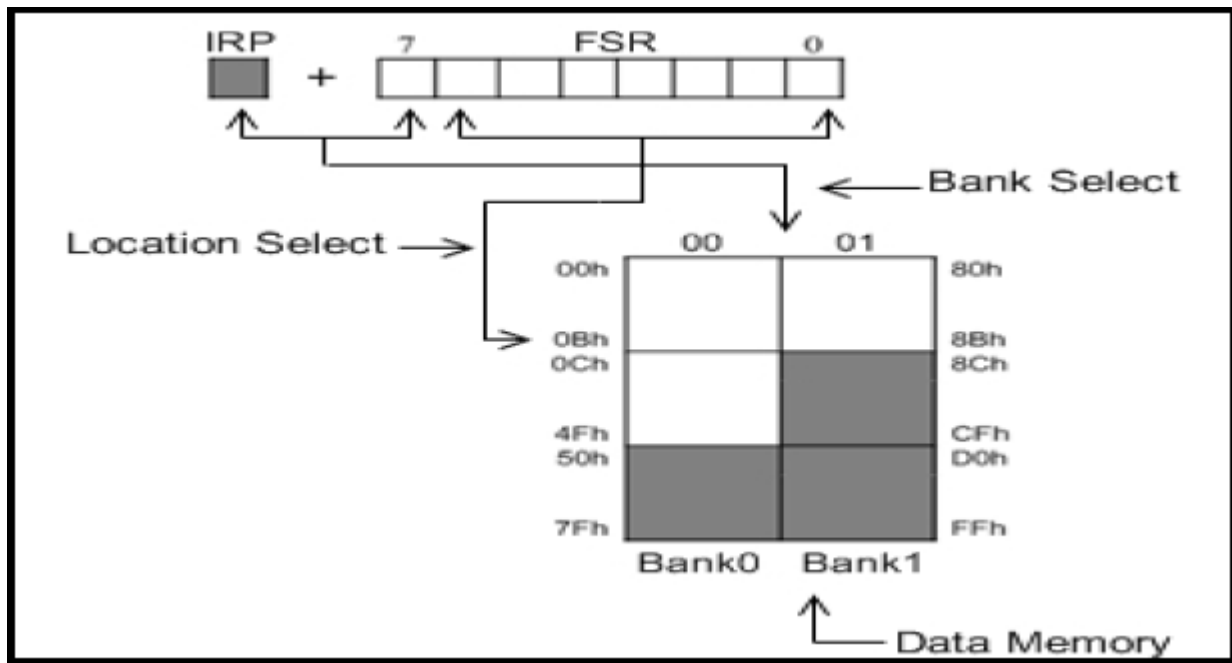
(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)



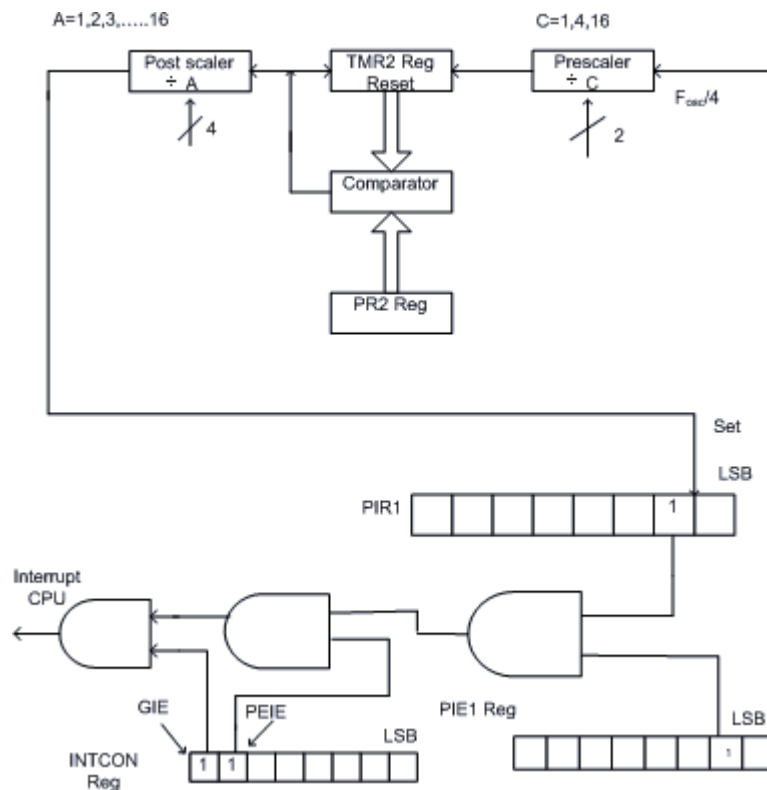
5. Indirect addressing:

It does not take an address from an instruction. But it derives from IRP bit of STATUS and FSR registers. Addressed location is accessed through INDF register. And INDF register in fact holds the address indicated by the FSR. Indirect addressing is very convenient for manipulating data arrays located in GPR registers. In this case, it is necessary to initialise FSR register with a starting address of the array, and the rest of the data can be accessed by increment the FSR register. Figure shows the indirect addressing concept.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)



Timer 2 Overview



(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

Fig : Schematic diagram showing operation of Timer 2

Timer 2 is an 8-bit timer with a pre-scaler and a post-scaler. It can be used as the PWM time base for PWM mode of capture compare PWM (CCP) modules. The TMR2 register is readable and writable and is cleared on device reset.

The input clock ($f_{osc}/4$) has a pre-scaler option of 1:1, 1:4 or 1:16 which is selected by bit 0 and bit 1 of T2CON register respectively.

The Timer 2 module has an 8-bit period register (PR2). Timer-2 increments from 00H until it is equal to PR2 and then resets to 00H on the next clock cycle. PR2 is a readable and writable register. PR2 is initialised to FFH on reset.

The output of TMR2 goes through a 4-bit post-scaler (1:1, 1:2, to 1:16) to generate a TMR2 interrupt by setting TMR2IF.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

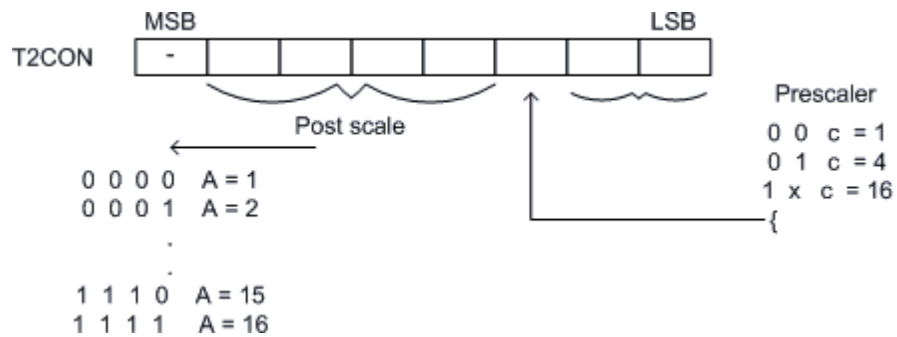


Fig : The T2CON Register

Interrupt Logic in PIC 16C74A

PIC 16C74A microcontroller has one vectored interrupt location (i.e., 0004H) but has 12 interrupt sources. There is no interrupt priority. Only one interrupt is served at a time. However interrupts can be masked. The interrupt logic is shown below :

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

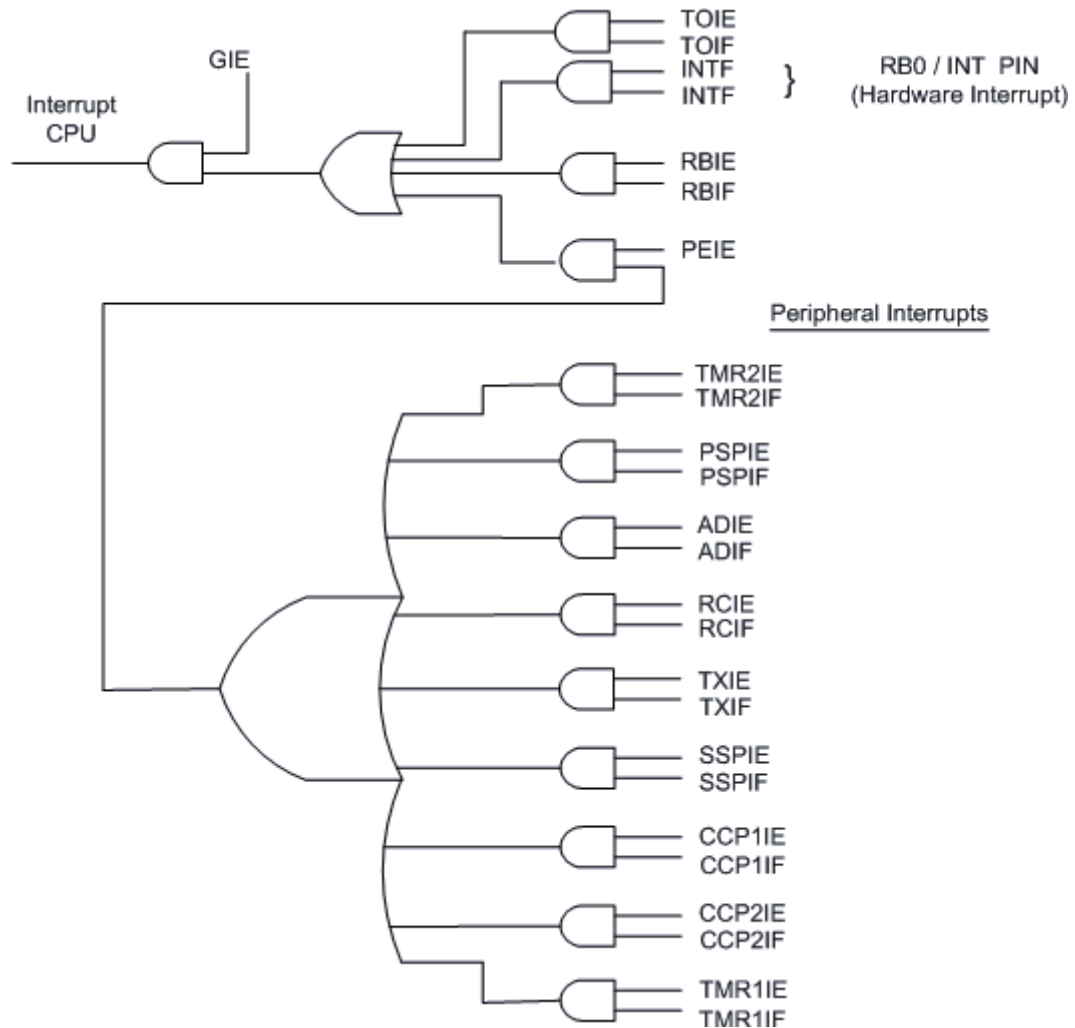


Fig : Schematic diagram showing the interrupt logic for PIC

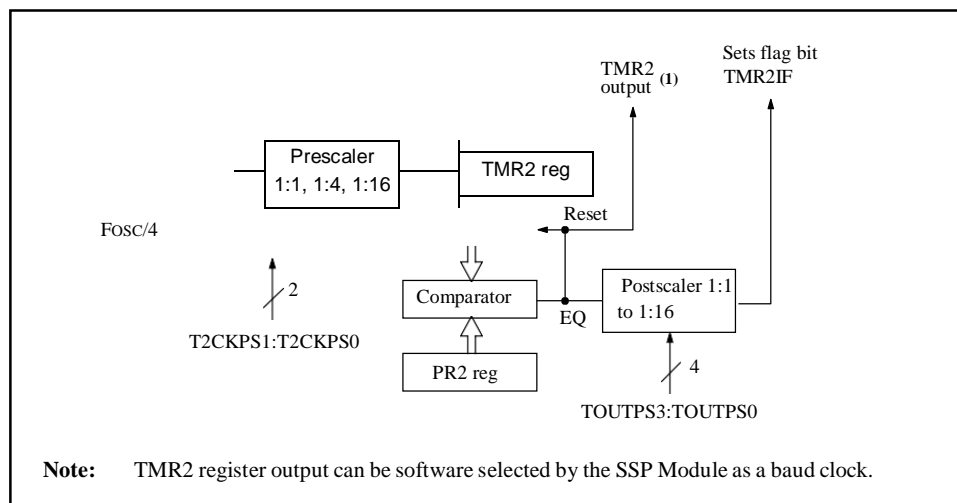
(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

TIMER 2 SCALAR INITIALISATION:

Timer2 is an 8-bit timer with a prescaler, a postscaler, and a period register. Using the prescaler and postscaler at their maximum settings, the overflow time is the same as a 16-bit timer. Timer2 is the PWM time-base when the CCP module(s) is used in the PWM mode.

Figure shows a block diagram of Timer2. The postscaler counts the number of times that the TMR2 register matched the PR2 register. This can be useful in reducing the overhead of the interrupt service routine on the CPU performance.

Figure : Timer2 Block Diagram



Control Register:

Register shows the Timer2 control register.

Register : T2CON: Timer2 Control Register

—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
b-7							b-0

bit 7

bit 6:3

Unimplemented: Read as '0'

TOUTPS3:TOUTPS0: Timer2 Output Postscale Select bits

0000 = 1:1 Postscale

0001 = 1:2 Postscale

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

•
•
•

1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit
 1 = Timer2 is on
 0 = Timer2 is off

bit 1:0 **T2CKPS1:T2CKPS0:** Timer2 Clock Prescale Select bits
 00 = Prescaler is 1
 01 = Prescaler is 4
 1x = Prescaler is 16

Legend

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

- n = Value at POR reset

Timer Clock Source

The Timer2 module has one source of input clock, the device clock ($F_{osc}/4$). A prescale option of 1:1, 1:4 or 1:16 is software selected by control bits T2CKPS1:T2CKPS0 (T2CON<1:0>).

Timer (TMR2) and Period (PR2) Registers

The TMR2 register is readable and writable, and is cleared on all device resets. Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register.

TMR2 is cleared when a WDT, POR, MCLR, or a BOR reset occurs, while the PR2 register is set.

Timer2 can be shut off (disabled from incrementing) by clearing the TMR2ON control bit (T2CON<2>). This minimizes the power consumption of the module.

TMR2 Match Output

The match output of TMR2 goes to two sources:

1. Timer2 Postscaler
2. SSP Clock Input

There are four bits which select the postscaler. This allows the postscaler a 1:1 to 1:16 scaling (inclusive). After the postscaler overflows, the TMR2 interrupt flag bit (TMR2IF) is set to indicate the Timer2 overflow. This is useful in reducing the software overhead of the Timer2 interrupt service routine, since it will only execute once every postscaler # of matches.

The match output of TMR2 is also routed to the Synchronous Serial Port module, which may software select this as the clock source for the shift clock.

Clearing the Timer2 Prescaler and Postscaler

The prescaler and postscaler counters are cleared when any of the following occurs:

- a write to the TMR2 register
- a write to the T2CON register

Note: When T2CON is written TMR2 does not clear.

- any device reset (Power-on Reset, MCLR reset, Watchdog Timer Reset, Brown-out Reset, or Parity Error Reset)

Sleep Operation

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

During sleep, TMR2 will not increment. The prescaler will retain the last prescale count, ready for operation to resume after the device wakes from sleep.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

Table : Registers Associated with Timer2

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR, PER	Value on all other resets
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
PIR	TMR2IF ⁽¹⁾								0	0
PIE	TMR2IE ⁽¹⁾								0	0
TMR2	Timer2 module's register								0000 0000	0000 0000
T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
PR2	Timer2 Period Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented read as '0'.

Shaded cells are not used by the Timer2 module.

Note 1: The position of this bit is device dependent.

INITIALISATION

Example shows how to initialize the Timer2 module, including specifying the Timer2 prescaler and postscaler.

```

CLRf    T2CON          ; Stop Timer2, Prescaler = 1:1,
                        ; Postscaler = 1:1
CLRf    TMR2           ; Clear Timer2 register
CLRf    INTCON          ; Disable interrupts
BSF     STATUS, RP0    ; Bank1
CLRf    PIE1           ; Disable peripheral interrupts
BCF     STATUS, RP0    ; Bank0
CLRf    PIR1           ; Clear peripheral interrupts Flags
MOVL    0x72           ; Postscaler = 1:15, Prescaler = 1:16
W
MOVW    T2CON          ; Timer2 is off
F
BSF     T2CON, TMR2ON  ; Timer2 starts to increment
;
; The Timer2 interrupt is disabled, do polling on the overflow bit
;
T2_OVFL_WAIT
    BTFSS PIR1, TMR2IF ; Has TMR2 interrupt occurred? GOTO
                        T2_OVFL_WAIT ; NO, continue loop
;
; Timer has overflowed
;
    BCF    PIR1, TMR2IF ; YES, clear flag and continue.

```

INTSERVICE INTERRUPT SERVICE ROUTINE:

- Whenever an interrupt occur, the CPU automatically pushes the return address in the PC onto the stack and clear the GIE (global interrupt enable) bit, disabling further interrupts. No other registers, or **W**, are automatically set aside.
- The first job of Intservice is to set aside the content of **W** and of **STATUS**. Then they can be restored at the end of the ISR to exactly the same state they were in when the interrupt occurred, as required for the proper execution of the machine code.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

Once interrupt have been enabled , Bank 1 register and RAM should only be accessed by indirect addressing

IntService

; set asude W and STATUS

```
movwf  W_TEMP      ;copy W to RAM
swapf  STATUS,W    ;move STATUS to W without affecting Z bit
movwf  STATUS_TEMP ;copy to RAM
```

; execute polling routine

```
btfsc  PIR1,TMR2IF ;check for timer2 interrupt
call   TIMER2      ; if ready, service it
;       btfsc      ..... ; check another interrupt source
;       call       ..... ; if ready, service it
;       btfsc      ..... ; check another interrupt source
;       call       ..... ; if ready, service it
```

; restore STATUS and W and retirn from interrupt

```
swapf  STATUS_TEMP,W ; Restore STATUS bits
movwf  STATUS         ;without affecting z bit
swapf  W_TEMP,F       ; swap W_TEMP
swapf  W_TEMP,W       ;swap again into W without affect Z bit
```

;;;;;;;;;;TIMER2 SUBROUTINE;;;;;;;;;;

Timer2

```
bcf     PIR1,TMR2IF      ; Clear interrupt flag (bank 0)
decf    SCALEWR,F
return
```

The central code of **IntService** is a sequence of **btfsc**, **call** instruction pairs. If tested flag is set, ISR is called otherwise call is skipped. This sequence is called polling subroutine.

LOOPTIME SUBROUTINE:

- **LoopTime** subroutine that is called within mainline loop is able to make the time around the loop take exactly 10 ms. For the loop time work correctly, the worst-case (i.e. longest) execution of the remainder of the code in the main loop plus the worst-case execution time for all the ISR that could request service within a 10-ms interval must be less than 10-ms.
- The mainline overrun condition is easily avoided for many, if not most, applications. If it is not avoided during one loop, the next looptime will be shortened to compensate.
- On the other hand, even if this mainline overrun condition does not occur, the long range timing provided by the loop time subroutine will still be accurate as long as no counts of SCALAR are

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

ever lost.

```
;;LoopTime subroutine:;;
```

LoopTime

```
    btfss    SCALER, 7
    goto     LoopTime
    movlw    5
    addwf    SCALAR, F
    return
```

IntService, which in turn wait on successive interrupt from timer2. When the timer2 interrupt occurs that finally decrements SCALER down from H'00' to H'FF', the goto Loop Time instruction will be skipped, five will be added to SCALER (resulting in SCALER =4), and the CPU will return from the ISR.

UART:

An universal asynchronous receiver and transmitter (UART) is an integrated circuit which is programmed to control a computer's interface to its attached serial devices [3]. Specifically, it provides the system with the RS-232C Data Terminal Equipment (DTE) interface, enabling it to talk to and exchange data with modems and some other serial devices. Being a part of this interface, the UART also provides the basic operations as:

- Converts the bytes it gets from the computer along parallel circuits to a single serial bit stream for outbound transmission.
- For inbound transmission, converts the serial bit stream to the bytes that the system handles.
- Adds a parity bit after selection in outbound transmissions, checks the parity of incoming bytes (if selected) and rejects the parity bit.
- Adds start and stop delineators for outbound and helps to strip them from inbound transmissions.
- Handles interrupts from keyboard and mouse (which are serial devices with special ports).

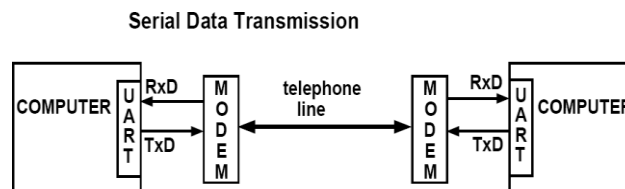


Fig. . Serial data transmission.

I. UART DESIGN:

The UART block diagram consists of three main components, *Transmitter control*, *Receiver control* and *Baud rate generator*. When transmitting , the UART takes eight bits of parallel data, converts the data to a serial bit stream that has a start bit (logic '0'), 8 data bits, and a stop bit (logic '1'). When receiving, the UART initially detects a start bit, then receives a stream of 8 data bits and

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

translates the data into parallel when it detects the stop bit. As no clock is transmitted, the UART must synchronize incoming stream of bits with the local clock.

The following six 8bit registers are used.

- 1- RSR- Receive shift register.
- 2- RDR- Receive data register.
- 3- TDR- Transmit data register.
- 4- TSR- Transmit shift register.
- 5-SCCR- Serial communications control register.
- 6-SCSR- Serial communications status registers.

Assume that the UART is connected to a microcontroller data and address bus so that the CPU can read and write to the registers. RDR, TDR, SCCR and SCSR are memory mapped. RDR, SCSR, SCCR can drive the data bus through tristate buffers. TDR and SCCR can be loaded from the data bus.

Besides the registers, the three main components of the UART are the Baud rate generator, the receiver and transmitter control. The Baud rate generator divides the clock of the system down to provide the bit clock (bclk) with a period equal to one bit time and also bclkx8, which has a frequency eight times the bclk frequency. The TDRE (transmit data register empty) bit in the SCSR is set when TDR is empty.

A. Baudrate Generator

The 8 MHz clock system clock is first divided by 13 using a counter. This counter output goes to 8 bit binary counter. The output from the flip-flops in this counter relates to divide by 2, 4 and so on upto divide by 256. Out of these outputs, one is selected by a multiplexer. The multiplexer selects inputs that come from the lower 3bits of the sccr. The output corresponds to bclkx8, which is again divided by 8 to give bclk.

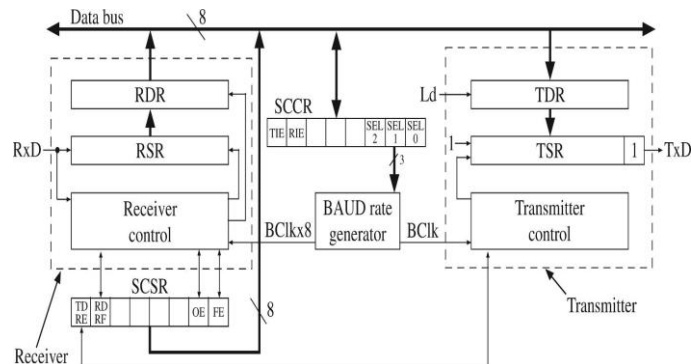


Fig. UART design model block diagram

Initially the process increments the divide by 13 counters on the rising edge of the system clock. The second process increments the divide by 256 counters on the rising edge of clkdiv13. A concurrent statement generates the mux output, bclkx8. The third process increments the divide by 8 counters on the rising edge of bclkx8 to generate bclk.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

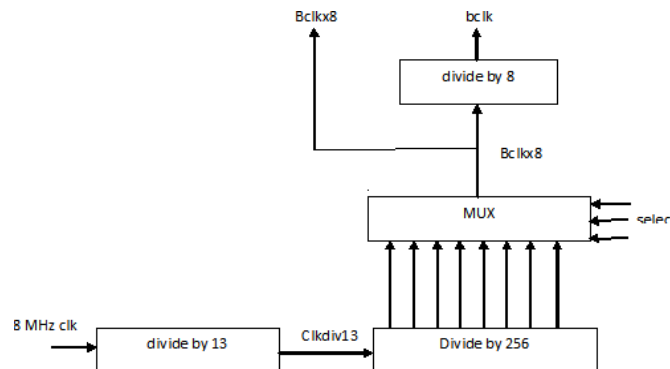


Fig. Baud rate generator block diagram.

B. UART Transmitter

When the microcontroller is ready to transmit data, the following occurs.

1. The microcontroller waits until TDRE='1' [5] and then loads a byte of data into TDR clears TDRE.
2. UART transfers data from TDR to TSR and sets TDRE.
3. The UART provides an output as a start bit '0' for one bit time and then shifts TSR right to transmit the eight data bits followed by the stop bit '1'.
4. In this step the UART transfers data from TDR to TSR and sets TDRE.
5. The UART gives a output as a start bit '0' for one bit time and shifts TSR right to transmit the eight data bits followed by the stop bit '1'.

SM chart of transmitter

In the IDLE state, the SM waits until TDR is loaded and consequently TDRE is cleared. In SYNCH state, the SM waits for the rising edge of the bit clock and then clears the low order bit of the TSR to transmit a '0' for one bit time. In the TDATA state, each time bclk↑ is detected, TSR is shifted right to transmit the next data bit and the bit counter (bct) is incremented. When bct=9 eight data bits and a stop bit have been transmitted, bct is then cleared and the SM goes back to idle.

The transmitter contains the TDR and TSR registers and the transmit control. It interfaces with TDRE and data bus (DBUS). The first process represents the combinational network, which generates the next state and control signals. The second process updates the registers on the rising edge of the clock. The signal bclk_rising is '1' for one system clock time following the rising edge of bclk. To generate bclk_rising, bclk is stored in a flip-flop named bclk_dlyd. Then bclk_rising is '1' if the current value of bclk is '1' and the previous value is '0'.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

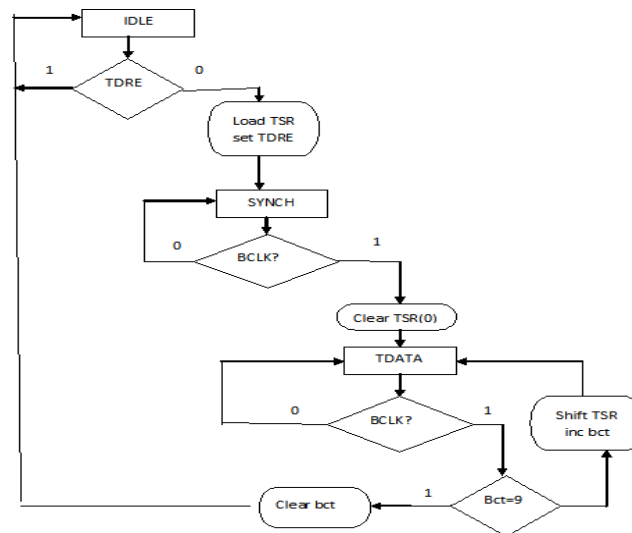


Fig. SM chart of transmitter

C. UART Receiver

1. When UART detects a start bit, it reads the left over bits serially and shifts them into RSR.
2. When all the data bits and the stop bit are received, the RSR loads into RDR and the flag of Receive Data Register Full (RDRF) in the SCSR is set.
3. The microcontroller checks for the RDRF flag, and if it is set, the flag is cleared by reading RDR.

RxD is sampled eight times during each bit time. It is sampled on the rising edge of the BClkX8. The bit is read in middle of each bit time for maximum reliability. When RxD first goes low, we will wait for four BClkX8 periods, which should take us closer to the middle of the first data bit. Then reading once per eight BClkX8 clocks is continued until we have read the stop bit.

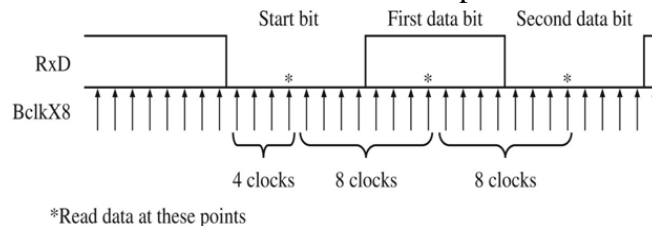


Fig. Sampling RxD with Bclkx8

SM chart of receiver

Two counters are used ct1 counts the number of BClkX8 clocks. ct2 counts the number of bits received after the start bit is encountered. In the IDLE state, the SM waits for the start bit (RxD = '0') and then goes to start detected state. Now the SM waits for the rising edge of BClkX8 and samples RxD once more. Since the start bit should be '0' for eight BClkX8 clocks, a '0' should be read. As ct1 is still 0, it is incremented and SM waits for the rising edge of BClkX8. If RxD = '1', this is an error

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

condition and SM clears ct1 and resets to the IDLE state. Otherwise SM keeps looping till RxD is '0'. When RxD is '0' for the fourth time, ct1 = 3, so ct1 is cleared the state goes to receive data. In this state, the SM increments ct1 after every rising edge of BClkX8. After the eighth clock, ct1 = 7 and ct2 is checked. If it is not 8, the current value of RxD is shifted to RSR, ct2 is incremented, and ct1 is cleared. If ct2 = 8, all the 8 bits have been read and we should be at the middle of the stop bit. If RDRF = 1, the microcontroller has not yet read the previously received data byte, and an overrun error has occurred, where the OE flag in the status register is set and the new data is ignored. If RxD = '0', the stop bit has not been detected properly, and the framing error (FE) flag in the status register is set. If no errors have occurred, RDR is loaded from RSR. In all cases, RDRF is set to indicate that receive operation is completed and counter is cleared.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

UNIT-3

8051 Microcontroller Assembly Language Programming

What is a Programming Language?

Programming in the sense of Microcontrollers (or any computer) means writing a sequence of instructions that are executed by the processor in a particular order to perform a predefined task. Programming also involves debugging and troubleshooting of instructions and instruction sequence to make sure that the desired task is performed.

Like any language, Programming Languages have certain words, grammar and rules. There are three types or levels of Programming Languages for 8051 Microcontroller. These levels are based on how closely the statements in the language resemble the operations or tasks performed by the Microcontroller.

The three levels of Programming Languages are:

- Machine Language
- Assembly Language
- High-level Language

Machine language

In Machine language or Machine Code, the instructions are written in binary bit patterns i.e. combination of binary digits 1 and 0, which are stored as HIGH and LOW Voltage Levels. This is the lowest level of programming languages and is the language that a Microcontroller or Microprocessor actually understands.

Assembly Language

The next level of Programming Language is the Assembly Language. Since Machine Language or Code involves all the instructions in 1's and 0's, it is very difficult for humans to program using it.

Assembly Language is a pseudo-English representation of the Machine Language. The 8051 Microcontroller Assembly Language is a combination of English like words called Mnemonics and Hexadecimal codes.

It is also a low level language and requires extensive understanding of the architecture of the Microcontroller.

High-level Language

The name High-level language means that you need not worry about the architecture or other internal details of a microcontroller and they use words and statements that are easily understood by humans.

Few examples of High-level Languages are BASIC, C Pascal, C++ and Java. A program called Compiler will convert the Programs written in High-level languages to Machine Code.

Why Assembly Language?

Although High-level languages are easy to work with, the following reasons point out the

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

advantage of Assembly Language

- The Programs written in Assembly gets executed faster and they occupy less memory.
- With the help of Assembly Language, you can directly exploit all the features of a Microcontroller.
- Using Assembly Language, you can have direct and accurate control of all the Microcontroller's resources like I/O Ports, RAM, SFRs, etc.
- Compared to High-level Languages, Assembly Language has less rules and restrictions.

Structure of the 8051 Microcontroller Assembly Language

The Structure or Syntax of the 8051 Microcontroller Assembly Language is discussed here. Each line or statement of the assembly language program of 8051 Microcontroller consists of three fields: Label, Instruction and Comments.

The arrangement of these fields or the order in which they appear is shown below.

[Label:]	Instructions	[//Comments]
-----------------	---------------------	---------------------

Before seeing about these three fields, let us first see an example of how a typical statement or line in an 8051 Microcontroller Assembly Language looks like.

TESTLABEL: MOV A, 24H ; THIS IS A SAMPLE COMMENT

In the above statement, the “TESTLABEL” is the name of the Label, “MOV A, 24H” is the Instruction and the “THIS IS A SAMPLE COMMENT” is a Comment.

TEST LABEL:	MOV A, 24H	; THIS IS A SAMPLE COMMENT
		
Label	Instruction	Comment

Label

The Label is programmer chosen name for a Memory Location or a statement in a program. The Label part of the statement is optional and if present, the Label must be terminated with a Colon (:).

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

An important point to remember while selecting a name for the Label is that they should reduce the need for documentation.

Instruction

The Instruction is the main part of the 8051 Microcontroller Assembly Language Programming as it is responsible for the task performed by the Microcontroller. Any Instruction in the Assembly Language consists of two parts: Op-code and Operand(s).



The first part of the Instruction is the Op-code, which is short for Operation Code, specifies the operation to be performed by the Microcontroller. Op-codes in Assembly Language are called as Mnemonics. Op-codes are in binary format (used in Machine Language) while the Mnemonic (which are equivalent to Op-codes) are English like statements.

The second part of the instruction is called the Operand(s) and it represents the Data on which the operation is performed. There are two types of Operands: the Source Operand and the Destination Operand. The Source Operand is the Input of the operation and the Destination Operand is where the result is stored.

Comments

The last part of the Structure of 8051 Assembly Language is the Comments. Comments are statements included by the developer for easier understanding of the code and is used for proper documentation of the Program.

Comments are optional and if used, they must begin with a semicolon (;) or double slash (//) depending on the Assembler.

The following statements will show a few possible ways of using Label, Instruction and Comments.

Label without instruction and comment: `LABEL:`

Line with Label and Instruction: `LABEL: MOV A, 22H`

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

Line with Instruction and Comment:

MOV A, 22H ; THIS IS A COMMENT

8051 Microcontroller Assembly Language Directives

Assembly Language Directives are not the instructions to the 8051 Microcontroller Assembler even though they are written in the Mnemonic field of the program. Assembly Language Directives are actually instructions to the Assembler and directs the Assembler Program what to do during the process of Assembling.

The Assembly Language Directives do not have any effect on the contents of the 8051 Microcontroller Memory (except DB and DW directives).

These Directives are dependent on the Assembler Program and in case of ASM51 Assembler, the following are the categories of Directives.

Assembler State Control	ORG, END, USING
Symbol Definition	SEGMENT, EQU, SET, DATA, IDATA, XDATA, BIT, CODE
Storage Initialization	DS, DBIT, DB, DW
Program Linkage	PUBLIC, EXTERN, NAME
Segment Selection	RSEG, CSEG, DSEG, ISEG, BSEG, XSEG

We will now see about few of the important and frequently used Assembly Language Directives.

ORG – Set Origin

The 8051 Microcontroller Assembly Language Program will start assembling from the Program Memory Address 0000H. This is also the address from which the 8051 Microcontroller will start executing the code.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

In order place the Program and Data anywhere in the Address Space of the 8051 Microcontroller, you can use the ORG Directive.

Examples

```
ORG 0000H      ; Tells the Assembler to assemble the next statement at 0000H

LJMP MAIN      ; Code Memory at 0000H. Jump to MAIN.

ORG 000BH      ; Tells the Assembler to assemble the next statement at 000BH

MAIN: NOP      ; Code Memory at 000BH. MAIN starts here.
```

DB – Define Byte

The DB Directive is used to define a Byte type variable. Using this directive, you can define data in Decimal, Binary, HEX or ASCII formats. There should be a suffix of 'B' for binary and 'H' for HEX. The ASCII Characters are placed in single quotation marks (like 'string').

Examples

```
ORG 0000H

DB 10          ; Define Byte 10 (Decimal) and store at 0000H

DB 30H         ; Define Byte 30 (HEX) and store at 0001H

DB 'STRING'     ; Define String 'STRING' and store at 0002H to 0007H

DB 00001111B   ; Define Byte 00001111 (Binary) and store at 0008H

DB 1234H       ; Define Byte 34 (HEX) and store at 0009H. Only lower byte is
                accepted as DB can allocate only a Byte of Memory.
```

DW – Define Word

The Define Word (DW) Directive is used to include a 16-bit data in a program. The functionality of DW is similar to that of DB except that DW generates 16-bit values.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

EQU – Equate

Using the EQU Directive, you can associate a Symbol (or Label) with a Value.

Examples

```
TMP EQU #30           ; Assigns the value #30 to the name TMP
```

```
RED_LED EQU P1.0       ; P1.0 is defined as RED_LED
```

END

The END Directive is used to stop the assembling process. This should be the last statement in the program. END Directive cannot have a Label and the statements beyond END will not be processed by the Assembler.

Example

```
ORG 0000H
```

```
MOV A, 20H
```

```
MOV R0, #30
```

```
END
```

Arithmetic Instructions

Using Arithmetic Instructions, you can perform addition, subtraction, multiplication and division. The arithmetic instructions also include increment by one, decrement by one and a special instruction called Decimal Adjust Accumulator.

The Mnemonics associated with the Arithmetic Instructions of the 8051 Microcontroller Instruction Set are:

- *ADD*
- *ADDC*
- *SUBB*
- *INC*

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

- *DEC*
- *MUL*
- *DIV*
- *DA A*

The arithmetic instructions has no knowledge about the data format i.e. signed, unsigned, ASCII, BCD, etc. Also, the operations performed by the arithmetic instructions affect flags like carry, overflow, zero, etc. in the PSW Register.

All the possible Mnemonics associated with Arithmetic Instructions are mentioned in the following table.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

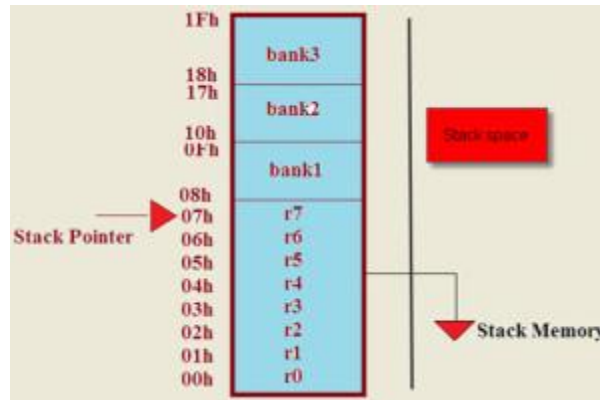
Mnemonic	Instruction	Description	Addressing Mode	# of Bytes	# of Cycles
ADD	A, #Data	$A \leftarrow A + \text{Data}$	Immediate	2	1
	A, Rn	$A \leftarrow A + Rn$	Register	1	1
	A, Direct	$A \leftarrow A + (\text{Direct})$	Direct	2	1
	A, @Ri	$A \leftarrow A + @Ri$	Indirect	1	1
ADDC	A, #Data	$A \leftarrow A + \text{Data} + C$	Immediate	2	1
	A, Rn	$A \leftarrow A + Rn + C$	Register	1	1
	A, Direct	$A \leftarrow A + (\text{Direct}) + C$	Direct	2	1
	A, @Ri	$A \leftarrow A + @Ri + C$	Indirect	1	1
SUBB	A, #Data	$A \leftarrow A - \text{Data} - C$	Immediate	2	1
	A, Rn	$A \leftarrow A - Rn - C$	Register	1	1
	A, Direct	$A \leftarrow A - (\text{Direct}) - C$	Direct	2	1
	A, @Ri	$A \leftarrow A - @Ri - C$	Indirect	1	1
MUL	AB	Multiply A with B ($A \leftarrow \text{Lower Byte of } A*B \text{ and } B \leftarrow \text{Higher Byte of } A*B$)	--	1	4
DIV	AB	Divide A by B ($A \leftarrow \text{Quotient and } B \leftarrow \text{Remainder}$)	--	1	4
DEC	A	$A \leftarrow A - 1$	Register	1	1
	Rn	$Rn \leftarrow Rn - 1$	Register	1	1
	Direct	$(\text{Direct}) \leftarrow (\text{Direct}) - 1$	Direct	2	1
	@Ri	$@Ri \leftarrow @Ri - 1$	Indirect	1	1
INC	A	$A \leftarrow A + 1$	Register	1	1
	Rn	$Rn \leftarrow Rn + 1$	Register	1	1
	Direct	$(\text{Direct}) \leftarrow (\text{Direct}) + 1$	Direct	2	1
	@Ri	$@Ri \leftarrow @Ri + 1$	Indirect	1	1
	DPTR	$DPTR \leftarrow DPTR + 1$	Register	1	2
DA	A	Decimal Adjust Accumulator	--	1	1

Stack Memory Allocation in 8051 Microcontroller

The stack is an area of random access memory (RAM) allocated to hold temporarily all the parameters of the variables. The stack is also responsible for reminding the order in which a function is called so that it can be returned correctly. Whenever the function is called, the parameters and local variables associated with it are added to the stack (PUSH). When the function returns, the parameters and the variables are removed ("POP") from the stack. This is why a program's stack size changes continuously while the program is running.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

The register used to access the stack is called stack pointer register. The stack pointer is a small register used to point at the stack. When we push something into the stack memory, the stack pointer increases.



When an 8051 microcontroller power up, the stack pointer contained value is 07, by default, as shown in the above figure. If we perform 'PUSH' operation, then the stack pointer address will be increased and shifted to another register. To avoid this problem, before starting the program, we have to assign a different address location to the stack pointer.

Bit Addressable RAM: 20h to 2Fh

The 8051 supports a special feature which allows access to bit variables. This is where individual memory bits in Internal RAM can be set or cleared. In all there are 128 bits numbered 00h to 7Fh. Being bit variables any one variable can have a value 0 or 1. A bit variable can be set with a command such as SETB and cleared with a command such as CLR. Example instructions are: SETB 25h ; sets the bit 25h (becomes 1) CLR 25h ; clears bit 25h (becomes 0) Note, bit 25h is actually bit b5 of Internal RAM location 24h. The Bit Addressable area of the RAM is just 16 bytes of Internal RAM located between 20h and 2Fh. So if a program writes a byte to location 20h, for example, it writes 8 bit variables, bits 00h to 07h at once. Note bit addressing can also be performed on some of the SFR registers.

Subroutine or subroutine

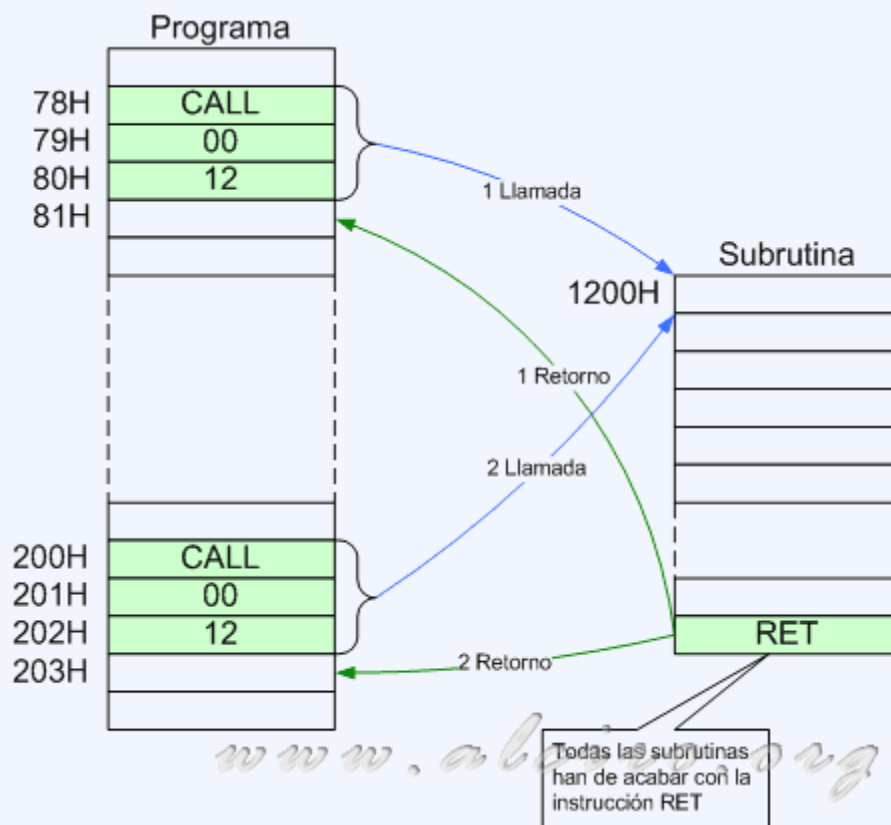
A **subroutine or subroutine** is a section separate program or part of the main program that can be called to perform a specific function. The subroutine may be required by the main program or another subroutine as many times as necessary. When you call a subroutine implementing the current program is stopped, the program counter PC (*program counter*) is loaded with the memory location of the subroutine, running up to the **RET** instruction (end of subroutine), where produce a return to the main program resumes running. The high level languages such as C, Basic subroutines are known under the name functions or procedures.

In the subroutines have to take into account the following considerations:

- Perform **specific functions** and are not operating on their own.
- Are always linked to a major program or other subroutines.
- Can be called many times as necessary as it reduces the code the program to have the effect of **code reuse**.

- As good advice, it is recommended whenever possible division of subroutines or sub-program and minimize the content of the sentences in the main program. Above all, the subroutines are necessary when part of a program be executed multiple times. We will make the program easier and it takes up less space in the ROM.

If a subroutine is made up of few instructions, it may be advisable not to create it, since the call and return mechanism may make it slower execution instructions to place directly in the main program.



Subroutine or subroutine in sensamblador

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

UNIT-4

INTERFACING OF LCD TO 8051

Display units are the most important output devices in embedded projects and electronics products. 16x2 LCD is one of the most used display unit. 16x2 LCD means that there are two rows in which 16 characters can be displayed per line, and each character takes 5X7 matrix space on LCD. In this tutorial we are going to connect [16X2 LCD module](#) to the 8051 microcontroller (AT89S52). **Interfacing LCD with 8051 microcontroller** might look quite complex to newbies, but after understanding the concept it would look very simple and easy. Although it may be time taking because you need to understand and connect 16 pins of LCD to the microcontroller. So first let's understand the 16 pins of LCD module.

We can divide it in five categories, Power Pins, contrast pin, Control Pins, Data pins and Backlight pins.

Category	Pin NO.	Pin Name	Function
Power Pins	1	VSS	Ground Pin, connected to Ground
	2	VDD or Vcc	Voltage Pin +5V
Contrast Pin	3	V0 or VEE	Contrast Setting, connected to Vcc thorough a variable resistor.
Control Pins	4	RS	Register Select Pin, RS=0 Command mode, RS=1 Data mode
	5	RW	Read/ Write pin, RW=0 Write mode, RW=1 Read mode
	6	E	Enable, a high to low pulse need to enable the LCD
Data Pins	7-14	D0-D7	Data Pins, Stores the Data to be displayed on LCD or the command instructions
Backlight Pins	15	LED+ or A	To power the Backlight +5V
	16	LED- or K	Backlight Ground

All the pins are clearly understandable by their name and functions, except the control pins, so they are explained below:

RS: RS is the register select pin. We need to set it to 1, if we are sending some data to be displayed on LCD. And we will set it to 0 if we are sending some command instruction like clear the screen (hex code 01).

RW: This is Read/write pin, we will set it to 0, if we are going to write some data on LCD. And set it to 1, if we are reading from LCD module. Generally this is set to 0, because we do not have need to read data from LCD. Only one instruction "Get LCD status", need to be read some times.

E: This pin is used to enable the module when a high to low pulse is given to it. A pulse of 450 ns should be

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

given. That transition from HIGH to LOW makes the module ENABLE.

There are some preset command instructions in LCD, we have used them in our program below to prepare the LCD (in lcd_init() function). Some important command instructions are given below:

Hex Code	Command to LCD Instruction Register
0F	LCD ON, cursor ON
01	Clear display screen
02	Return home
04	Decrement cursor (shift cursor to left)
06	Increment cursor (shift cursor to right)
05	Shift display right
07	Shift display left
0E	Display ON, cursor blinking
80	Force cursor to beginning of first line
C0	Force cursor to beginning of second line
38	2 lines and 5×7 matrix
83	Cursor line 1 position 3
3C	Activate second line
08	Display OFF, cursor OFF
C1	Jump to second line, position 1
OC	Display ON, cursor OFF
C1	Jump to second line, position 1
C2	Jump to second line, position 2

Pin 3(V0) is connected to voltage (Vcc) through a variable resistor of 10k to adjust the contrast of LCD. Middle leg of the variable resistor is connected to PIN 3 and other two legs are connected to voltage supply and Ground.

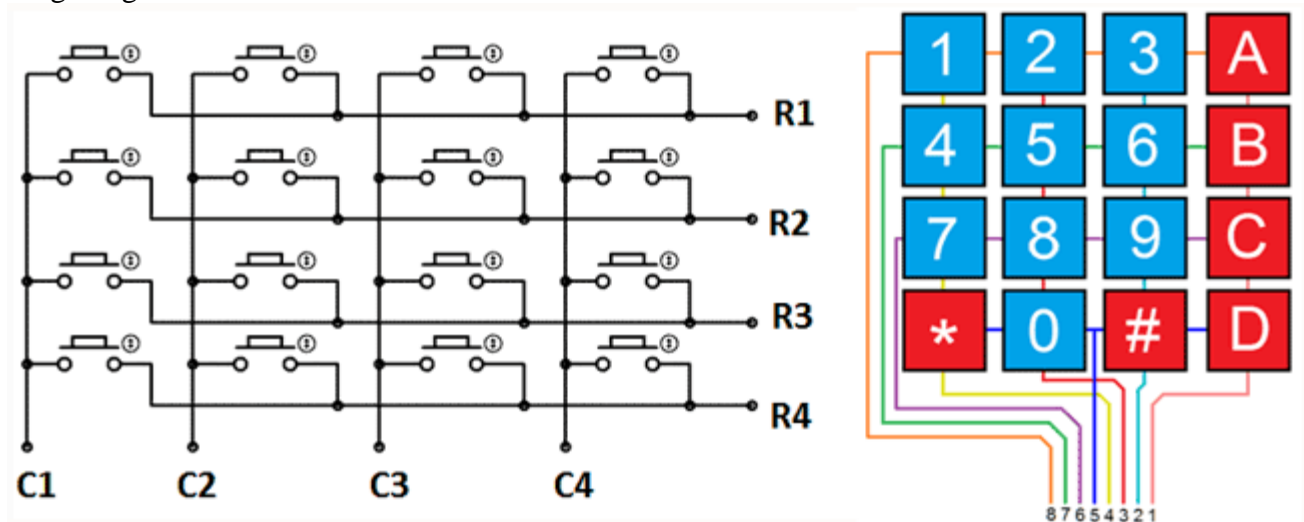
(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

INTERFACING OF 8051 WITH KEYBOARD

Keypads are widely used input devices being used in various electronics and embedded projects. They are used to take inputs in the form of numbers and alphabets, and feed the same into system for further processing. In this tutorial we are going to **interface a 4x4 matrix keypad with 8051 microcontroller**.

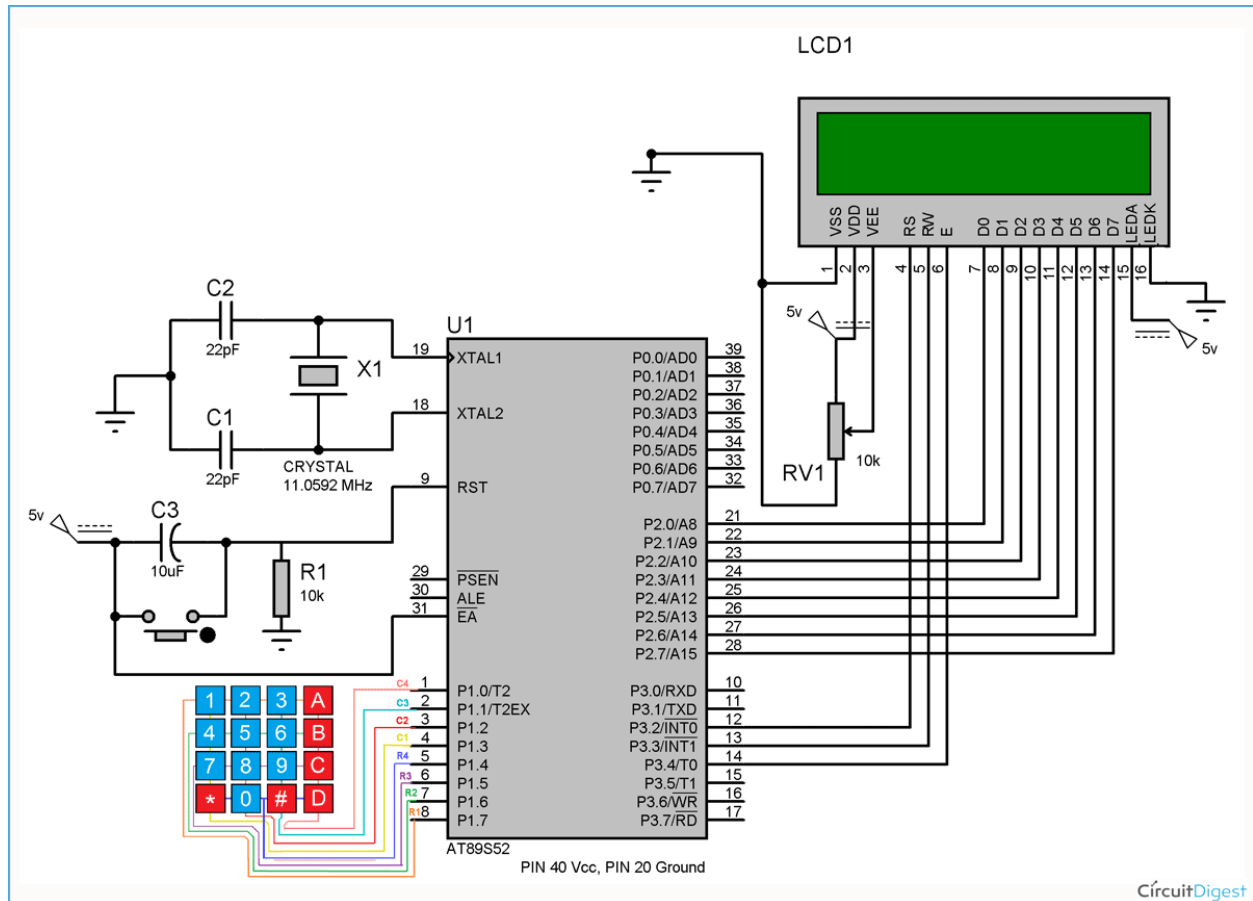
4X4 Matrix Keypad

Before we interface the keypad with microcontroller, first we need to understand how it works. Matrix keypad consists of set of Push buttons, which are interconnected. Like in our case we are using 4X4 matrix keypad, in which there are 4 push buttons in each of four rows. And the terminals of the push buttons are connected according to diagram. In first row, one terminal of all the 4 push buttons are connected together and another terminal of 4 push buttons are representing each of 4 columns, same goes for each row. So we are getting 8 terminals to connect with a microcontroller.



(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

Interfacing keypad with 8051 microcontroller (AT89S52)



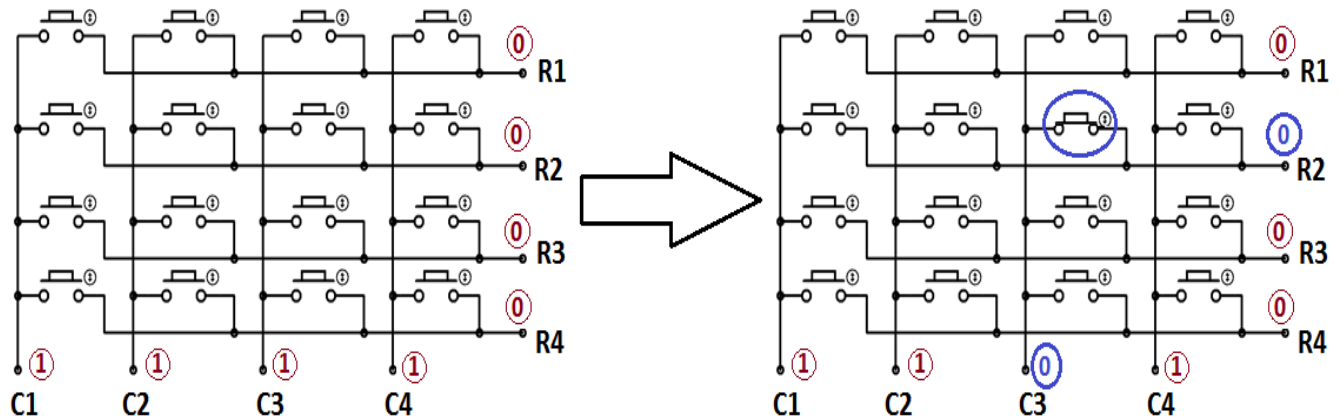
First we need to interface a LCD module to display the data which will be feed through KEYPAD, so please go through “[LCD Interfacing with 8051 Microcontroller](#)” article before interfacing KEYPAD.

As shown in above circuit diagram, to interface Keypad, we need to connect 8 terminals of the keypad to any port (8 pins) of the microcontroller. Like we have connected keypad terminals to Port 1 of 8051. Whenever any button is pressed we need to get the location of the button, means the corresponding ROW an COLUMN no. Once we get the location of the button, we can print the character accordingly.

Now the question is how to get the location of the pressed button? I am going to explain this in below steps and also want you to look at the code:

1. First we have made all the Rows to Logic level 0 and all the columns to Logic level 1.
2. Whenever we press a button, column and row corresponding to that button gets shorted and makes the corresponding column to logic level 0. Because that column becomes connected (shorted) to the row, which is at Logic level 0. So we get the column no. See main() function.

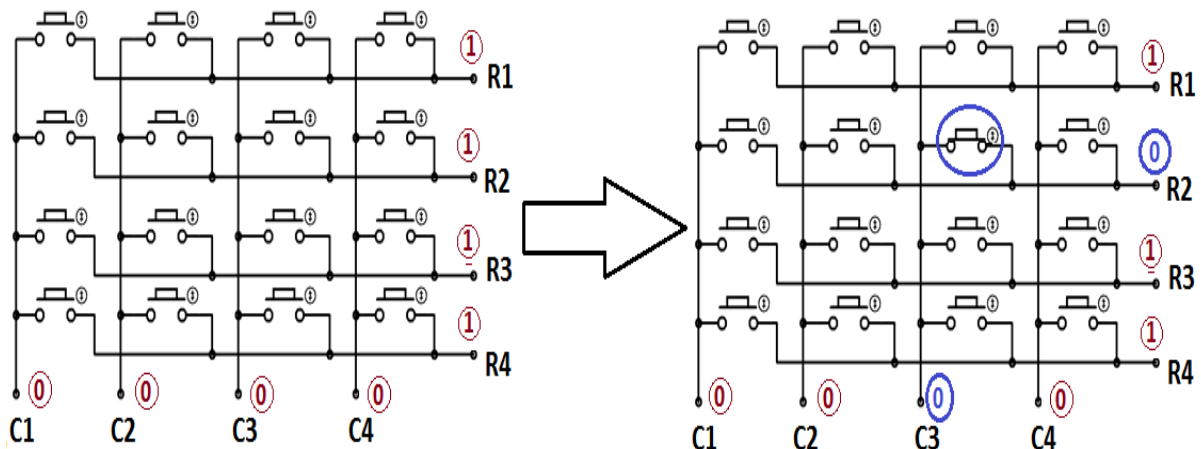
(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)



FINDING COLUMN No.

3. Now we need to find the Row no., so we have created four functions corresponding to each column. Like if any button of column one is pressed, we call function row_finder1(), to find the row no.

4. In row_finder1() function, we reversed the logic levels, means now all the Rows are 1 and columns are 0. Now Row of the pressed button should be 0 because it has become connected (shorted) to the column whose button is pressed, and all the columns are at 0 logic. So we have scanned all rows for 0.



FINDING ROW No.

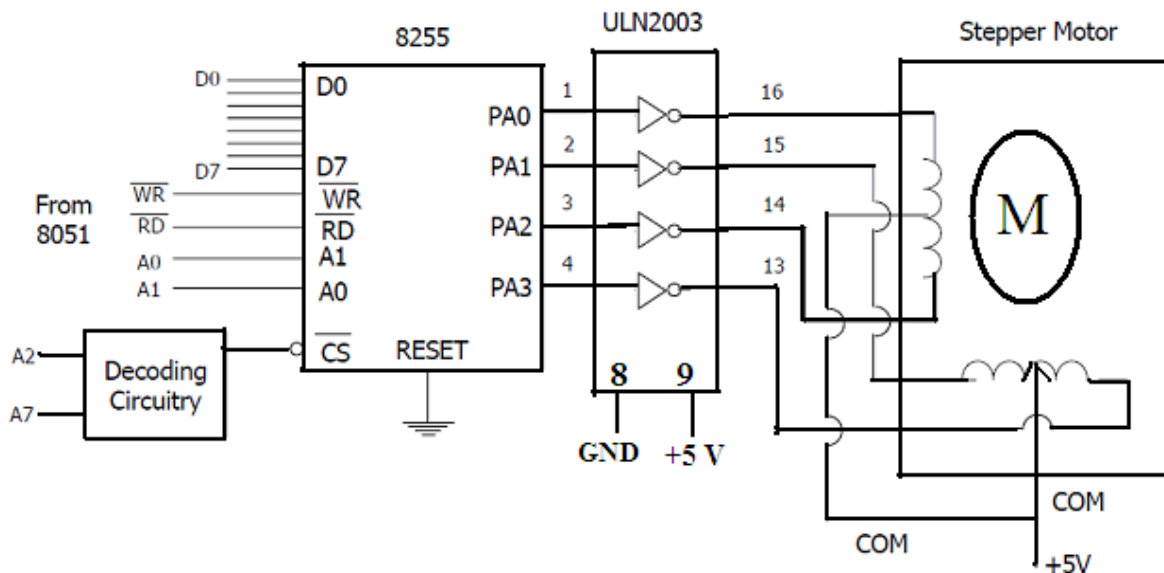
5. So whenever we find the Row at logic 0, means that is the row of pressed button. So now we have column no (got in step 2) and row no., and we can print no. of that button using lcd_data function.

Same procedure follows for every button press, and we are using while(1), to continuously check, whether button is pressed or not.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

INTERFACING WITH STEPPER MOTOR

A stepper motor is a device that translates electrical pulses into mechanical movement. The stepper motor rotates in steps in response to the applied signals. It is used in applications such as disk drives, dot matrix printers, plotters and robotics. It is mainly used for position control. Stepper motors have a permanent magnet called rotor (also called the shaft) surrounded by a stator. There are also steppers called variable reluctance stepper motors that do not have a PM rotor. The most common stepper motors have four stator windings that are paired with a center-tapped. This type of stepper motor is commonly referred to as a four-phase or unipolar stepper motor. The center tap allows a change of current direction in each of two coils when a winding is grounded, thereby resulting in a polarity change of the stator.

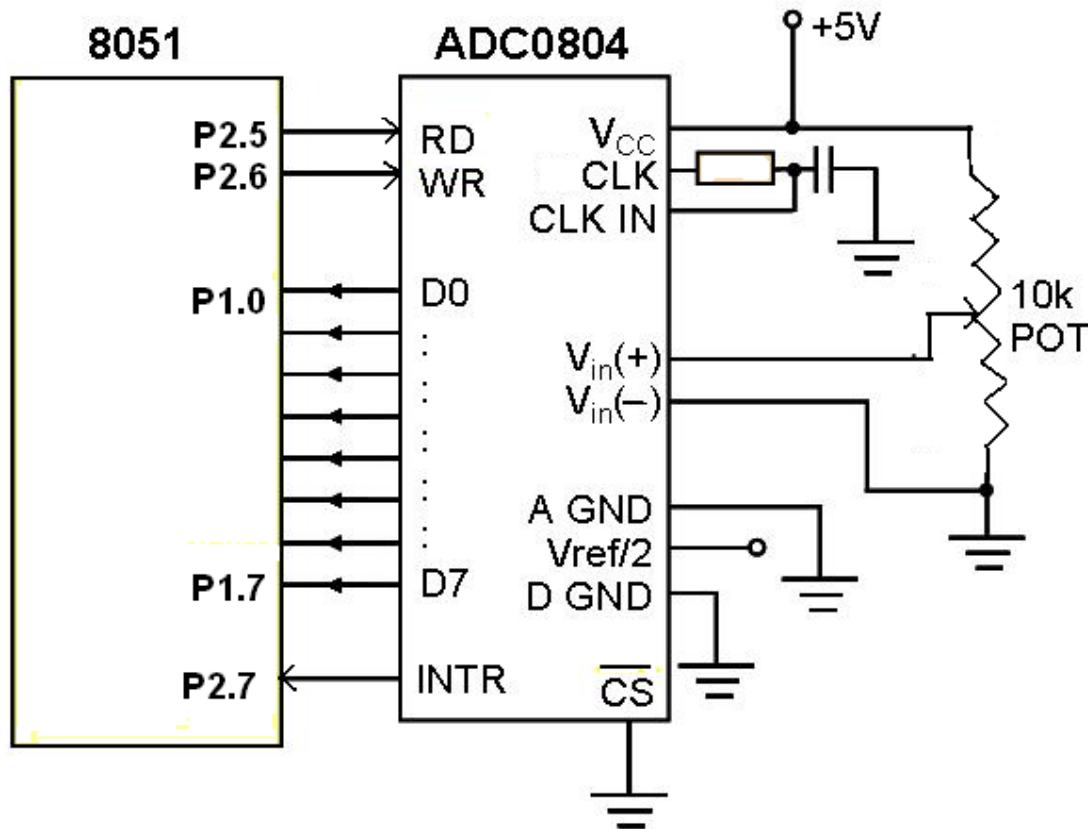


Interfacing of ADC 0804 to 8051 Microcontroller :

ADC 0804 is a single channel analog to digital converter i.e., it can take only one analog signal. ADC 0804 has 8 bit resolution. The higher resolution ADC gives smaller step size. Step size is smallest change that can be measured by an ADC. For an ADC with resolution of 8 bits, the step size is 19.53mV (5V/255). The time taken by the ADC to convert analog data into digital form depends on the frequency of clock source. The conversion time of ADC 0804 is around 110us. To use the internal clock a capacitor and resistor are used as shown in the circuit. The input to the ADC is given from a regulated power supply and a 10K potentiometer

The 8051 Microcontroller is used to provide the control signals to the ADC. CS(chip select) pin of ADC is directly connected to ground. The pin P1.1, P1.0 and P1.2 are connected to the pin WR, RD and INTR of the ADC respectively. When the input voltage from the preset is varied the output of ADC varies also varies.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)



From the circuit it is clear that the ADC interfaced directly to the microcontroller. The Port1 is used as an input port which receives the digital data from the ADC. Port pins P2.5 and P2.6 are used for SOC and EOC operation. When the conversion is over the ADC will send an interrupt signal to the microcontroller through the pin P2.7. Now the Microcontroller receives digital data through the Port1. This data after conversion to decimal data is displayed on the LCD module.

INTERFACING DAC -8051 MICROCONTROLLER

The DAC 0800 is a simple monolithic 8-bit D/A converter. It has fast settling time of 100ns. It can be directly interfaced to TTL, CMOS, PMOS and others. It operates at 4.5V to +18V supply. 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.

The interfacing circuit is shown below. port 1(8 bits of the microcontroller is connected to the input data lines of DAC-08. The reference current is determined by the resistor R_1 and the reference voltage V_{ref} . The

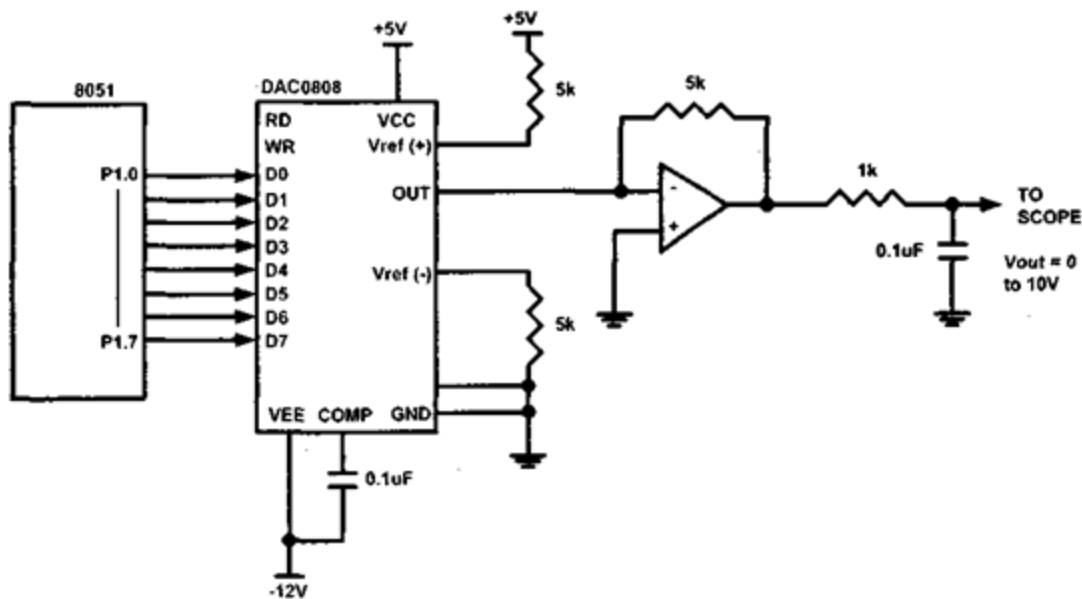
(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

resistor R_2 is generally equal to R_1 to match the input impedance of reference source. The output (taken from pin number 4) is observed either on a digital multimeter or on a cathode ray oscilloscope.

The output current I_o is calculated as follows:

$$I_o = V_{ref}/R_1[A_0/2 + A_1/4 + A_2/8 + \dots + A_7/256]$$

The output voltage V_o is obtained as follows: $V_o = I_o * R_1$



4 MEMORY MAPPED I/O INTERFACING

In memory-mapped I/O, each input or output device is treated as if it is a memory location. The MEMR and MEMW control signals are used to activate the devices. Each input or output device is identified by unique 16-bit address, similar to 16-bit address assigned to memory location. All memory related instruction like LDA 2000H, LDAX B, MOV A, M can be used.

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

Since the I/O devices use some of the memory address space of 8085, the maximum memory capacity is lesser than 64 KB in this method.

Ex: Interface an 8-bit DIP switch with the 8085 using logic gates such that the address assigned to it is F0F0H.

Since a 16-bit address has to be assigned to a DIP switch, the memory-mapped I/O technique must be used. Using LDA F0F0H instruction, the data from the 8-bit DIP switch can be transferred to the accumulator. The steps involved are:

- i. The address F0F0H is placed in the address bus A0 – A15.
- ii. The MEMR signal is made low for some time.
- iii. The data in the data bus is read and stored in the accumulator.

Fig. 22 shows the interfacing diagram.

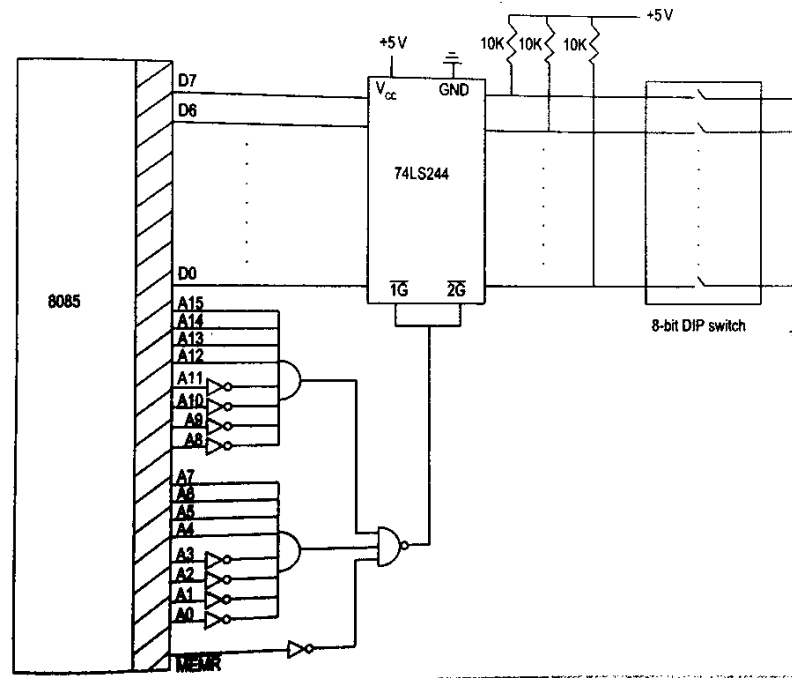


Fig. 22 Interfacing 8-bit DIP switch with 8085

When 8085 executes the instruction LDA F0F0H, it places the address F0F0H in the address lines A0 – A15 as:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	= F0F0H

The address lines are connected to AND gates. The output of these gates along with MEMR signal are connected to a NAND gate, so that when the address F0F0H is placed in the address bus and MEMR = 0 its output becomes 0, thereby enabling the buffer 74LS244. The data from the DIP switch is placed in the 8085 data bus. The 8085 reads the data from

(Prepared By: Mr. Ashok Saini , Assistant Professor , ECE)

the data bus and stores it in the accumulator.

DESIGNING USING MICROCONTROLLER

For this whole unit follow this link

<http://www.faadooengineers.com/online-study/post/ece/embedded-system/258/designing-music-box-with-microcontroller>

FOR THE TOPIC

Interpreter, compiler, high level language and intel hex format object files follow the link

<http://www.faadooengineers.com/online-study/post/eee/embedded-system-design/1068/interpreter-and-compiler>