Microcontrollers

The Intel 8051

A Microcontroller

- A Microcontroller is a Full Computer System on a chip , **designed for stand alone applications, with resources far more limited** than a GPC.

- A Microcontroller is a VLSI Integrated Circuit that contains CPU, Memory (Program Memory and Data Memory), I/O Ports (Input / Output Ports) and few other components integrated on a single chip.

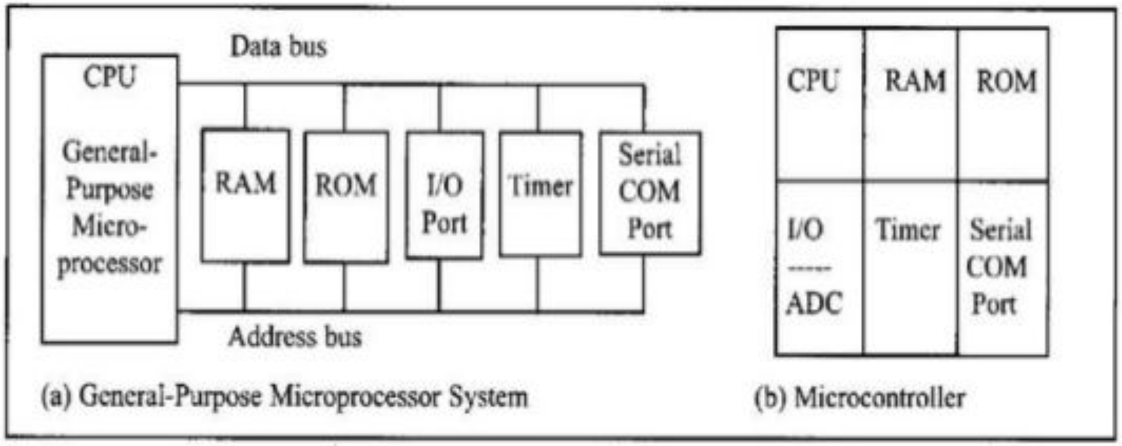
- Tiny yet powerful device that changed the face of Embedded Systems.

- Powerful and carefully chosen electronics embedded in the microcontrollers can independently or via input/output devices (switches, push buttons, etc.), **control various processes and devices such as industrial automation, electric current, temperature, engine performance etc**.

- **Very low prices** enable them to be embedded in such devices in which, until recent time it was not worthwhile to embed anything.

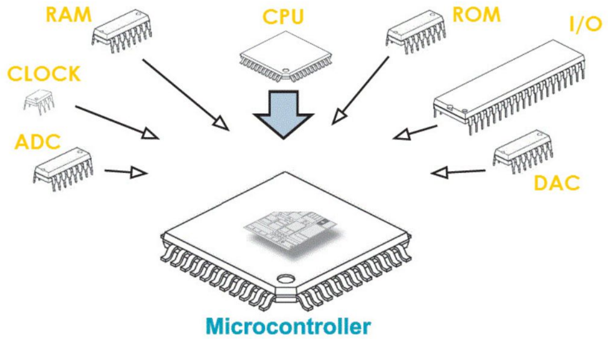
- **Prior knowledge is hardly needed for programming.**

What is a Microcontroller?



(a) General-Purpose Microprocessor System

(b) Microcontroller



A **timer module** to allow the MCU to perform tasks for certain periods of time.

A **serial I/O port** to allow data to flow between the MCU and other devices such as a PC or another MCU.

An **ADC** to allow the MCU to accept analog signals for processing.

But a microcontroller cannot do all the functions of a computer system on its own and needs other circuits to support it like I/O devices, RAM, ROM, DMA controllers, Timers, ADC, LCD drivers etc.

Use of Microcontrollers

Microcontrollers are omnipresent.

- The biggest user of Microcontrollers is probably the ***Automobiles Industry.***
- ***Consumer Electronics*** is another area which is loaded with Microcontrollers.
- Microcontrollers are also used in ***test and measurement equipment*** like Multimeters, Oscilloscopes, Function Generators, etc.
- You can also find microcontrollers near your desktop computer like Printers, Routers, Modems, Keyboards, etc.

Basic Components of a Microcontroller:

- **CPU**: which monitors and controls all processes within the microcontroller and the user cannot affect its work.

- **Program Memory (ROM)**: ROM can be built in the microcontroller or added as an external chip

- **Data Memory (RAM)**: used for temporary storing data and intermediate results created and used during the operation of the microcontrollers

- **EEPROM**: not contained in all microcontrollers. Often used to store values, created and used during operation, which must be saved after turning the power supply off.

- **SFR**: are part of RAM memory. Purpose is predefined by the manufacturer and cannot be changed thereof. Each bit of this register controls the function of one single pin.

- **PC**: an engine running the program and points to the memory address containing the next instruction to execute.

Basic Components of a Microcontroller:

- **I/O Ports**: Each microcontroller has one or more registers (called a port) connected to the microcontroller i/o pins.

- **Oscillator**: pulses generated by the oscillator enable harmonic and synchronous operation of all circuits within the microcontroller.

- **Timers/Counters, Watchdog Timer** : These are commonly 8- or 16-bit SFRs the contents of which is automatically incremented by each coming pulse.

- **Power Supply Circuit, Serial Communication, Interrupt Mechanism:**

- Most modern Microcontrollers might contain even more peripherals like **SPI** (Serial Peripheral Interface), **I2C** (Inter Integrated Circuit), **ADC** (Analog to Digital Converter), **DAC** (Digital to Analog Converter), **CAN** (Controlled Area Network), **USB** (Universal Serial Bus), and many more.

Advantages and Disadvantages

Advantages:

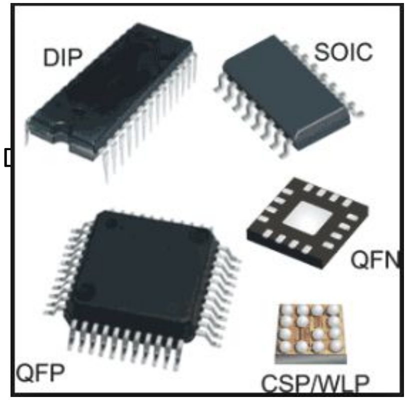
- A Microcontroller is a true device that fits the computer-on-a-chip idea.
- No need for any external interfacing of basic components like Memory, I/O Ports, etc.
- Microcontrollers doesn’t require complex operating systems as all the instructions must be written and stored in the memory. (RTOS is an exception).
- All the Input/Output Ports are programmable.
- Integration of all the essential components reduces the cost, design time and area of the product (or application).

Disadvantages:

- Microcontrollers are not known for their computation power.
- The amount of memory limits the instructions that a microcontroller can execute.
- No Operating System and hence, all the instruction must be written.

Criteria for choosing a Microcontroller:

- **Efficiency in handling task at hand:**



- Speed
- Packaging (SOIC small outline integrated chip occupies area about 30-50% less than equivalent D with 70% less thickness) (QFN quad flat no-leads, CSP chip-scale package)
- Power consumption
- Amount of ROM & RAM
- No. of I/O pins & timer on chip
- How easy to upgrade to higher performance or lower power consumption versions
- Cost per unit

- **Easiness to develop products around it:**

- Availability of assembler, debugger, emulator, technical support, support from third party vendor, etc.

- **Ready availability in needed quantities:**

Commonly used µCs and µPs in ES:

- **PIC** : first RISC microcontrollers produced by Microchip Technology in Chandler, Arizona. The acronym PIC stands for "peripheral interface controller”. Low cost, small size, and low power consumption. Key features include wide availability, low cost, ease of reprogramming with built-in EEPROM (electrically erasable programmable read-only memory), an extensive collection of free application notes, abundant development tools, and a great deal of information available on the Internet.

- **8051** (Intel and others): very powerful and easy to program. Modified Harvard architecture with separate address spaces for program memory and data memory. The 8051 features the so-called “**Boolean processor**“

- **80c196 (MCS-96)**: hardware multiply and divide, 6 addressing modes, high speed I/O, A/D, serial communications channel, up to 40 I/O ports, 8 source priority interrupt controller, PWM generator, and watchdog timer.

Commonly used µCs and µPs in ES:

- **80386 EX(Intel):** is a 32-bit microprocessor introduced in 1985 that holds the ability to carry out 32-bit operation in one cycle. It has data and address bus of 32-bit each.

- **68HC05(Motorola**): The **68HC05** (HC05 in short) is a broad family of 8-bit microcontrollers from Freescale Semiconductor (formerly Motorola Semiconductor). Like all Motorola processors they use the von Neumann architecture as well as memory-mapped I/O.

- **Z8(Zilog)**:The Zilog **Z8** is a microcontroller architecture, originally introduced in 1979. Signifying features of the architecture are up to 4,096 fast on-chip registers which may be used as accumulators, pointers, or as ordinary RAM. A 16-bit address space for between 1 KB and 64 KB of either OTP ROM or flash memory are used to store code and constants, and there is also a second 16-bit address space which may be used for large applications.

Basic 8051 Architecture

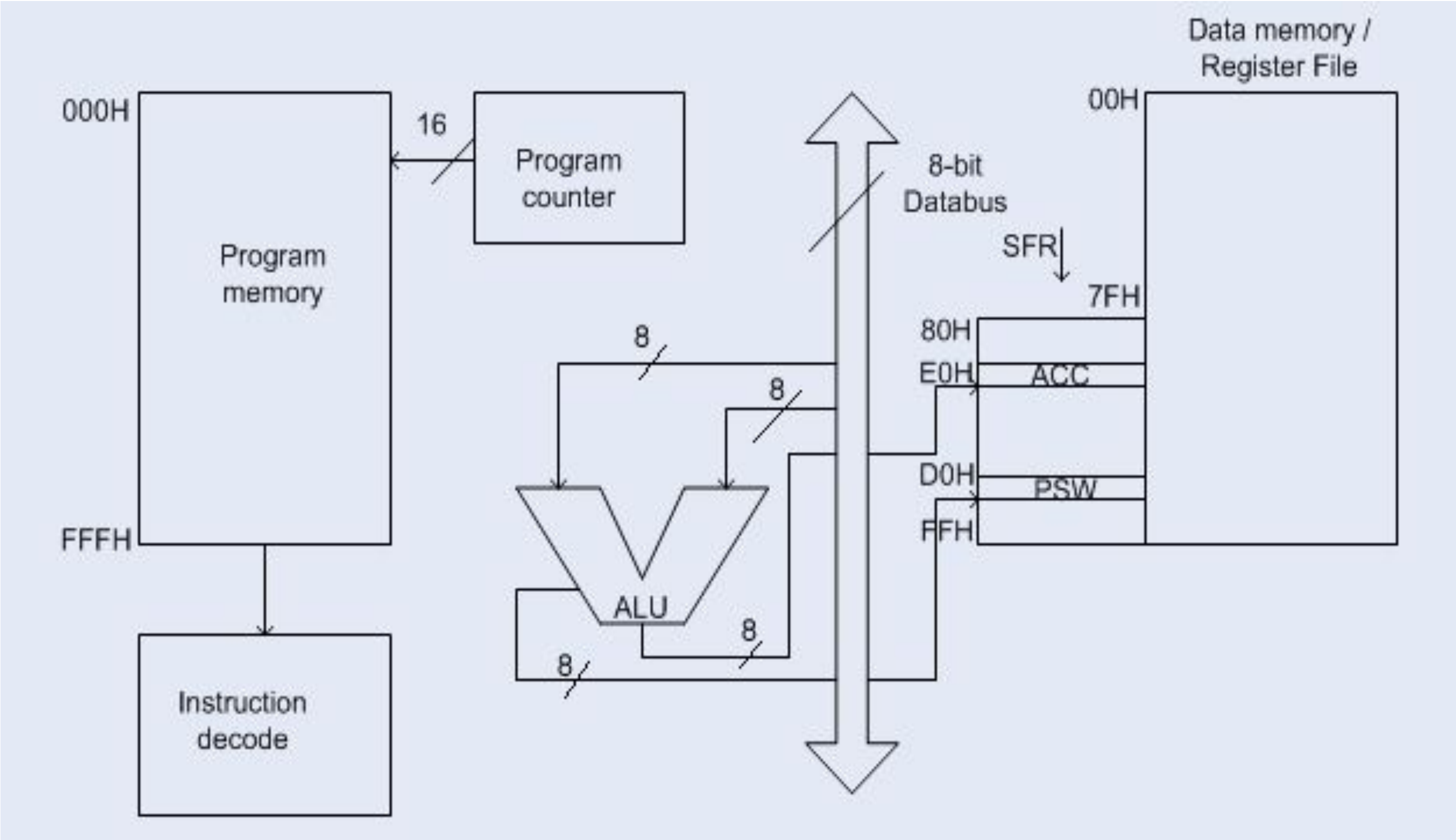
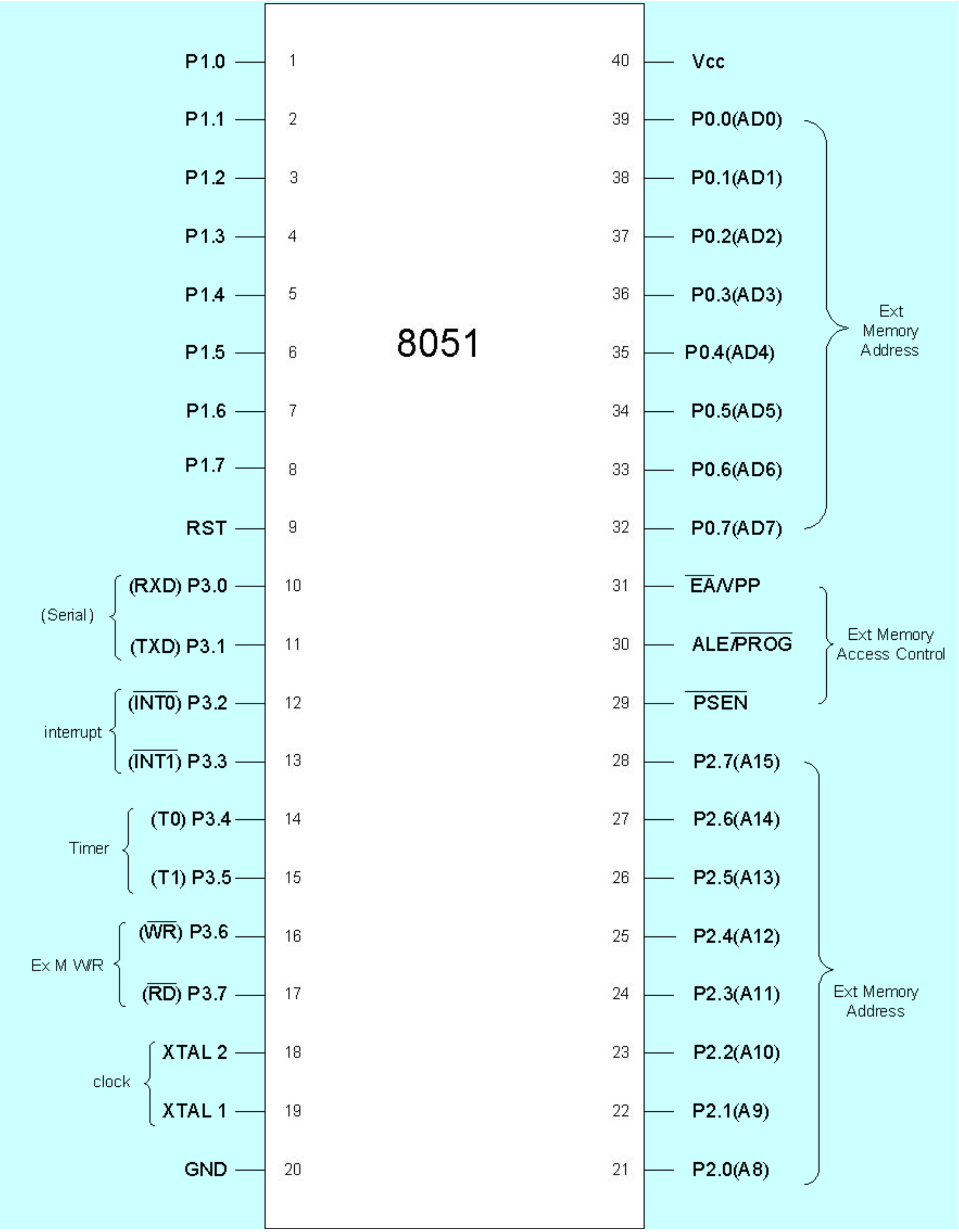


Fig 5.1 : Basic 8051 Architecture



8051

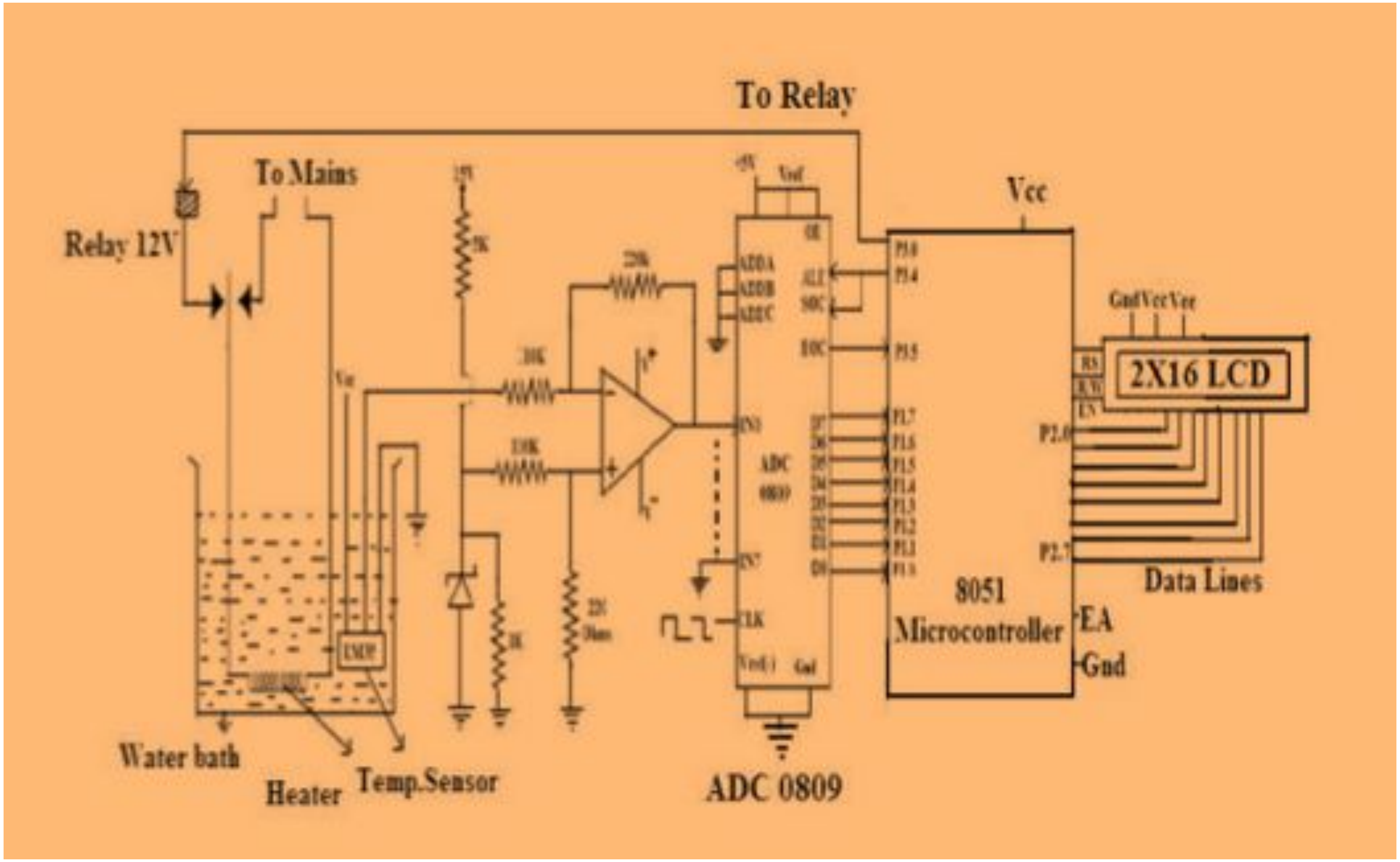
P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(Serial) (RXD) P3.0 10
(TXD) P3.1 11
interrupt ($\overline{\text{INT0}}$) P3.2 12
($\overline{\text{INT1}}$) P3.3 13
Timer (T0) P3.4 14
(T1) P3.5 15
Ex M WR ($\overline{\text{WR}}$) P3.6 16
($\overline{\text{RD}}$) P3.7 17
clock XTAL 2 18
XTAL 1 19
GND 20

40 Vcc
39 P0.0(AD0)
38 P0.1(AD1)
37 P0.2(AD2)
36 P0.3(AD3)
35 P0.4(AD4)
34 P0.5(AD5)
33 P0.6(AD6)
32 P0.7(AD7)
Ext Memory Address
31 $\overline{\text{EA}}$/VPP
30 ALE/$\overline{\text{PROG}}$
29 $\overline{\text{PSEN}}$
Ext Memory Access Control
28 P2.7(A15)
27 P2.6(A14)
26 P2.5(A13)
25 P2.4(A12)
24 P2.3(A11)
23 P2.2(A10)
22 P2.1(A9)
21 P2.0(A8)
Ext Memory Address

Basic Features:

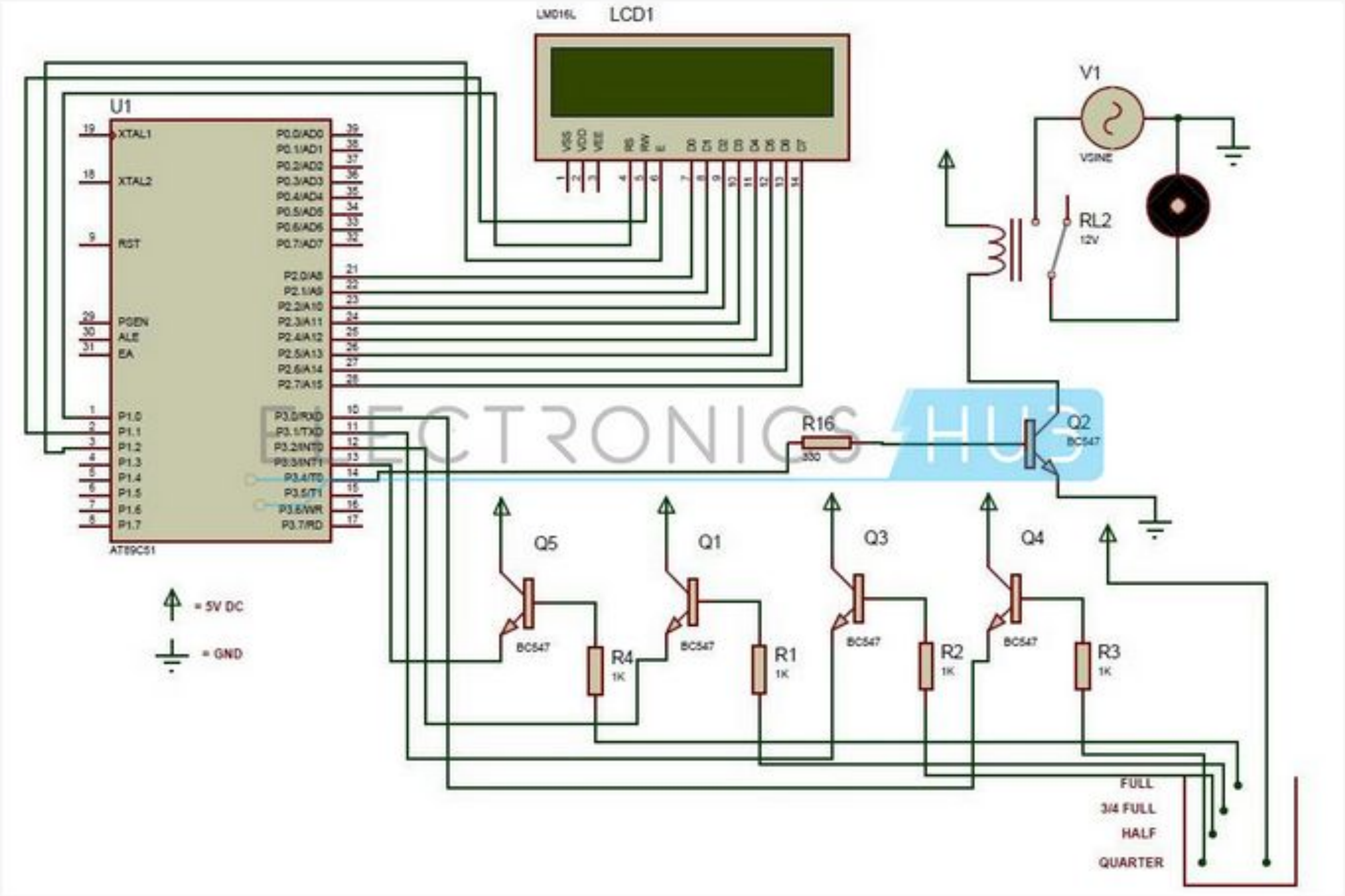
- 8-bit CPU, 8-bit ALU
- 16-bit PC, 8-bit PSW, 8-bit SP, 16-bit DP
- Internal RAM of 128bytes
- Special Function Registers (SFRs) of 128 bytes
- 32 I/O pins arranged as four 8-bit ports (P0 - P3)
- Two 16-bit timer/counters : T0 and T1
- Two external and three internal vectored interrupts
- One full duplex serial I/O
- Widely used registers: A, B, R0, R1, R2, R3, R4, R5, R6, R7, PC, DPTR (DPH, DPL)
- Microcontroller (8051 family) when powered on, wakes up at memory address 0000H.
- In the original 8051, one instruction cycle consists of 12 clock cycles. Each instruction cycle has 6 states (S_1 - S_6). Each state has two pulses (P1 and P2).
- Is able to access registers, RAM and I/O ports in bits as well as bytes

Temperature Measurement and Control



LM35:Temperature Sensor (in centigrade) : 10mV o/p for 1 degree centigrade
ADC0809: 8-i/p channel system
LM741: op-amp for signal conditioning

Water Level Controller using 8051 Circuit Diagram:

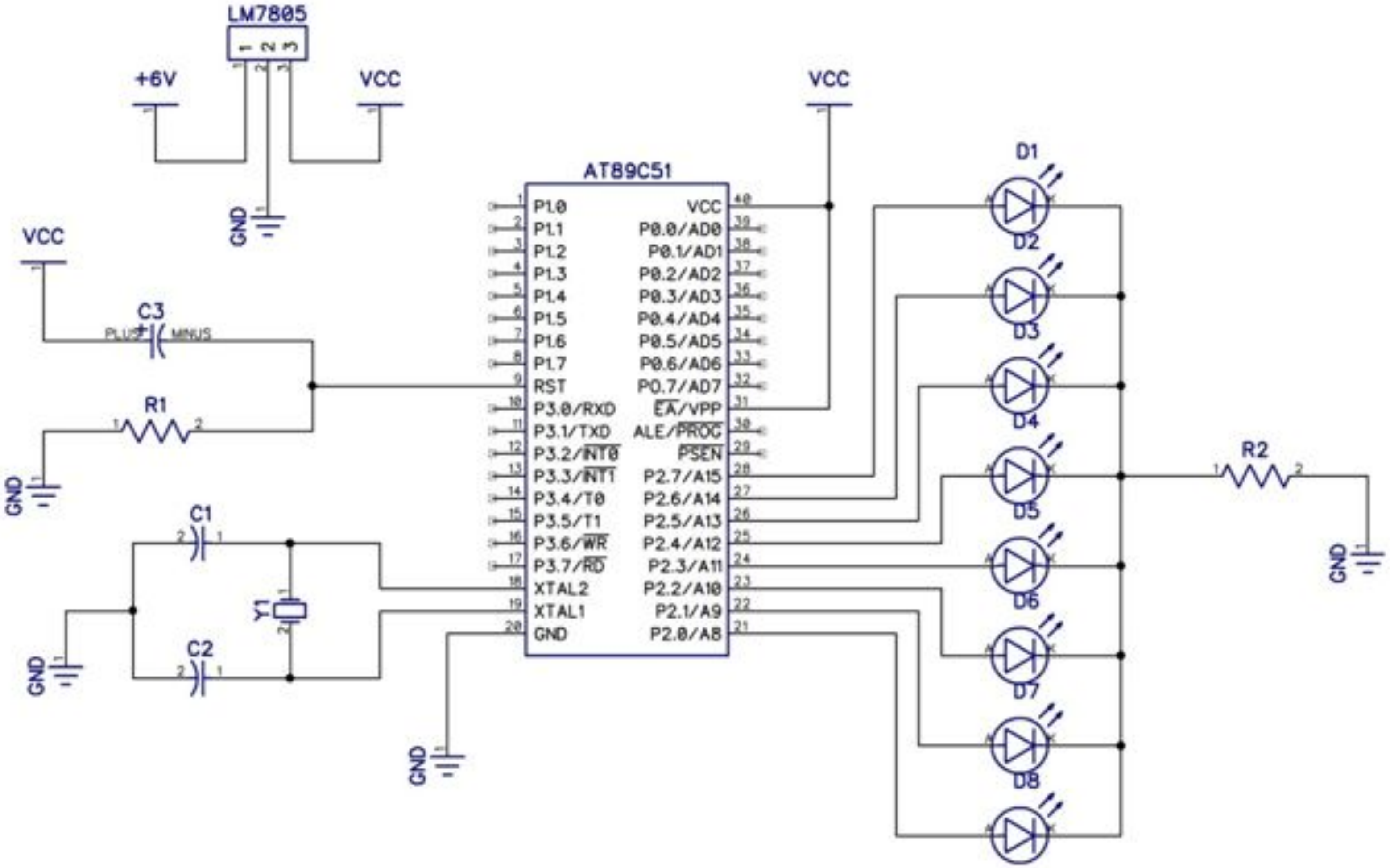


Circuit Diagram of Water Level Controller using 8051 Microcontroller

The water level probes are connected to the P3.0, P3.1, P3.2, and P3.3 through the transistors.

Port P2 connected to the data pins of LCD and control pins RS, RW and EN of LCD are connected to the P1.0, P1.1, and P1.2 respectively.

Connecting the 8051



8051 Tutorial 10 - Basic Hardware Connections - YouTube

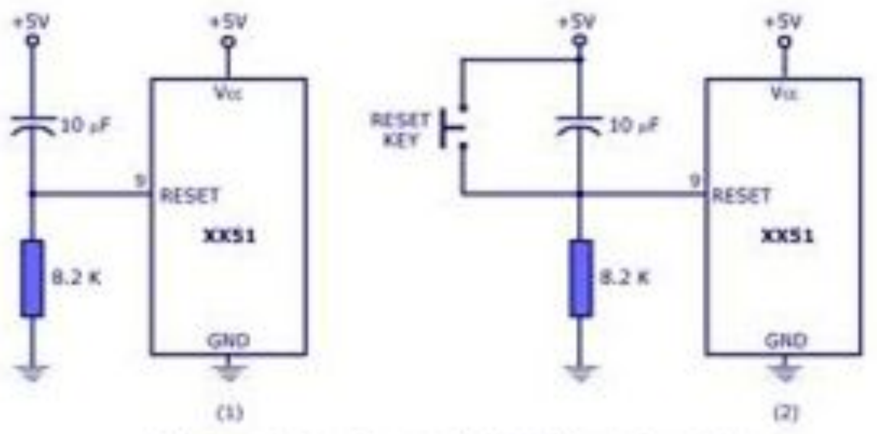
- RESET is an active HIGH input. When RESET is set to HIGH, 8051 goes back to power-on state.

- The 8051 is reset by holding the RST HIGH for at least two machine cycles and then returning it to LOW.

- After a RESET, the PC is loaded with 0000H but the content of the on-chip RAM chip is not affected.

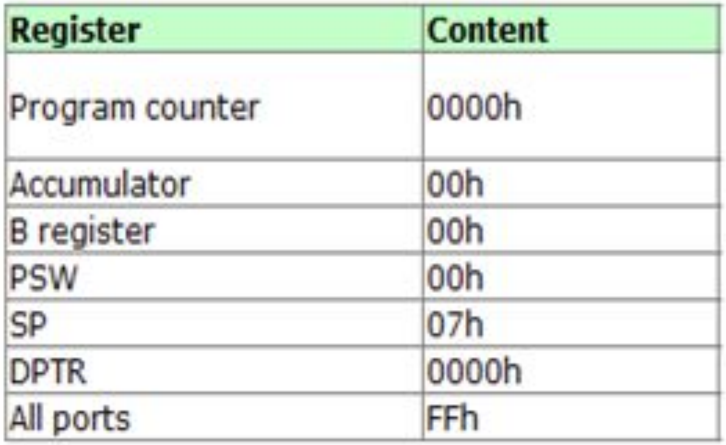
- There are two methods of RESET:
- 1. **Power on Reset:**
- Initial charging of capacitor makes RST HIGH.
- When capacitor charges fully it blocks DC and RST goes LOW.

- **2.Manual Reset:**
- Closing the switch momentarily will make RST HIGH.



(1) Power-on Reset Circuit and (2) With Manual Reset Option

www.CircuitsToday.com



Register	Content
Program counter	0000h
Accumulator	00h
B register	00h
PSW	00h
SP	07h
DPTR	0000h
All ports	FFh

- The 8051 uses the crystal oscillator to synchronize its operation.

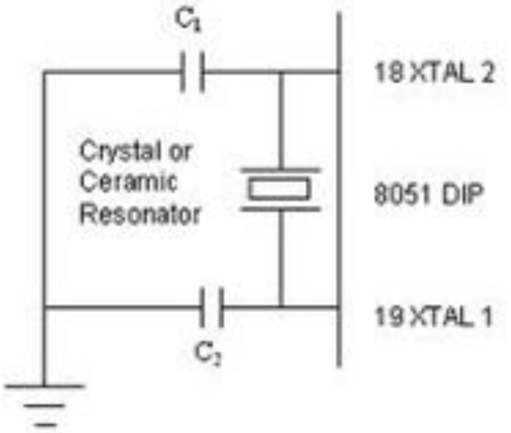
- Effectively the 8051 operates using what are called “machine cycles”.

- A single machine cycle is the minimum time in which a single 8051 instruction can be executed.

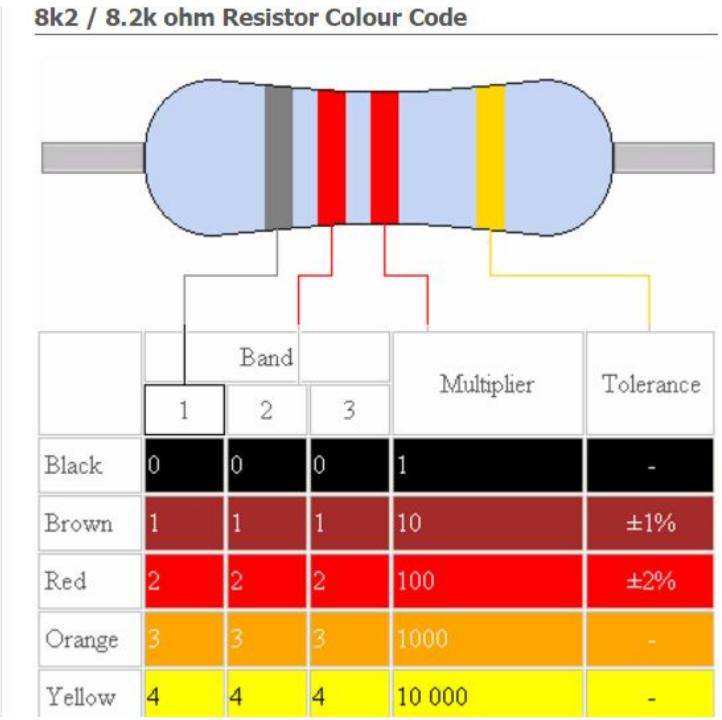
- Many instructions take multiple cycles.

- 8051 has an on-chip oscillator which needs an external crystal that decides the operating frequency of the 8051.

- This crystal (11.0592MHz) is connected to pins 18 and 19 with stabilizing capacitors.



C_1= 33pF
C_2= 33pF



8k2 / 8.2k ohm Resistor Colour Code

	Band			Multiplier	Tolerance
	1	2	3		
Black	0	0	0	1	-
Brown	1	1	1	10	±1%
Red	2	2	2	100	±2%
Orange	3	3	3	1000	-
Yellow	4	4	4	10 000	-

PC and ROM space in the 8051

- As CPU fetches opcode from program ROM, PC is incremented to point to the next instruction.

- PC being 16 bits wide can access program addresses from 0000H to FFFFH. (*all members of 8051 do not have the entire 64K bytes of on chip ROM installed)

- When powered up 8051 wakes up at 0000H. Therefore 1st opcode of program should be burned at 0000 location.

***AT89S51- ROM 4K bytes and 128 bytes RAM whereas AT89S52 has 8K bytes of ROM and 256 bytes of RAM**.*

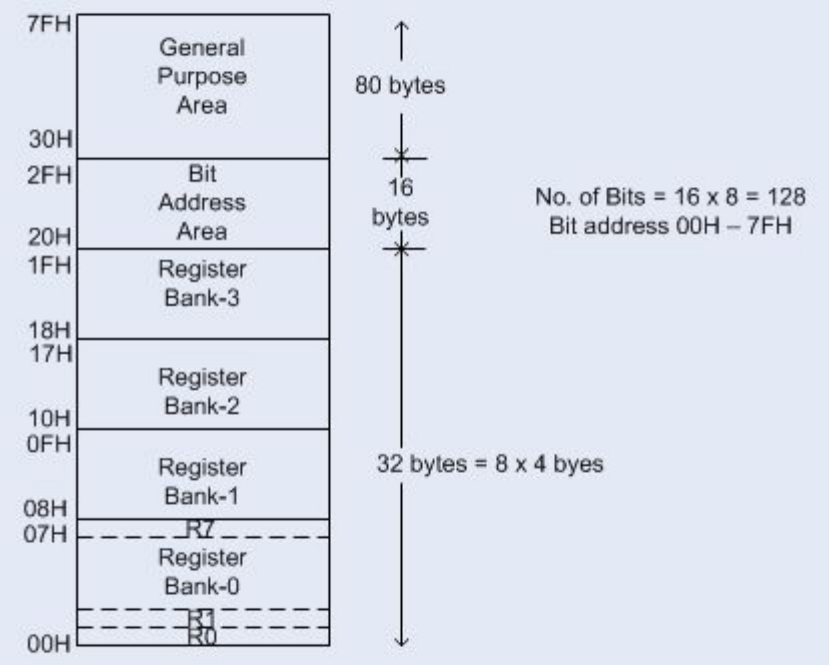
8051 Microcontroller Memory Organization| RAM And ROM Organization of 8051|8051 SFR functions Addres – YouTube

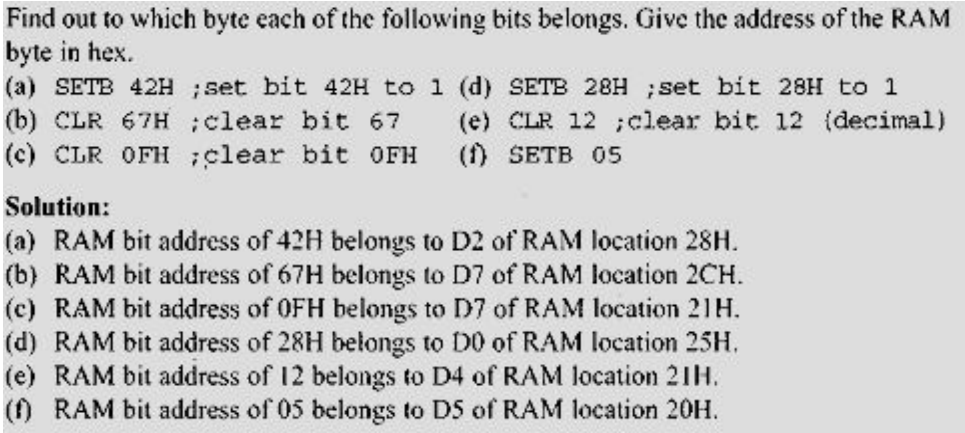
Program execution (byte by byte)

```
0000 7D25        MOV R5, #25H
0002 7F34        MOV R7, #34H
0004 7400        MOV A, #0H
0006 2D       ADD A, R5
0007 2F       ADD A, R7
0008 2412        ADD A, #12H
000A 80FE        HERE:SJMP HERE
```

RAM Memory Space Allocation in 8051

- **128 bytes of Internal RAM Structure (lower address space)**





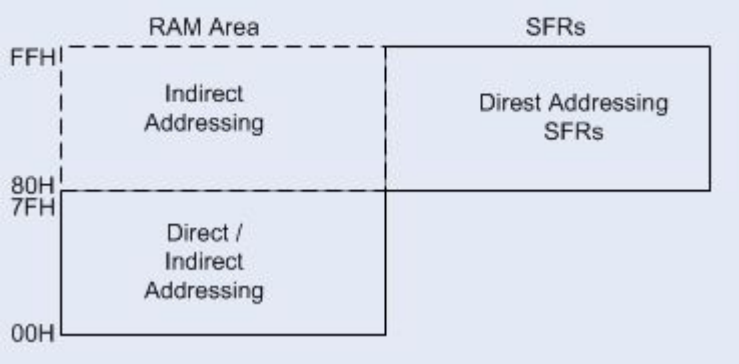
Find out to which byte each of the following bits belongs. Give the address of the RAM byte in hex.

```
(a) SETB 42H ;set bit 42H to 1 (d) SETB 28H ;set bit 28H to 1
(b) CLR 67H ;clear bit 67      (e) CLR 12 ;clear bit 12 (decimal)
(c) CLR 0FH ;clear bit 0FH     (f) SETB 05
```

Solution:

(a) RAM bit address of 42H belongs to D2 of RAM location 28H.
(b) RAM bit address of 67H belongs to D7 of RAM location 2CH.
(c) RAM bit address of 0FH belongs to D7 of RAM location 21H.
(d) RAM bit address of 28H belongs to D0 of RAM location 25H.
(e) RAM bit address of 12 belongs to D4 of RAM location 21H.
(f) RAM bit address of 05 belongs to D5 of RAM location 20H.

Internal Data Memory and Special Function Register (SFR) Map



*Upper 128 bytes of data RAM are present only in the 8052 family

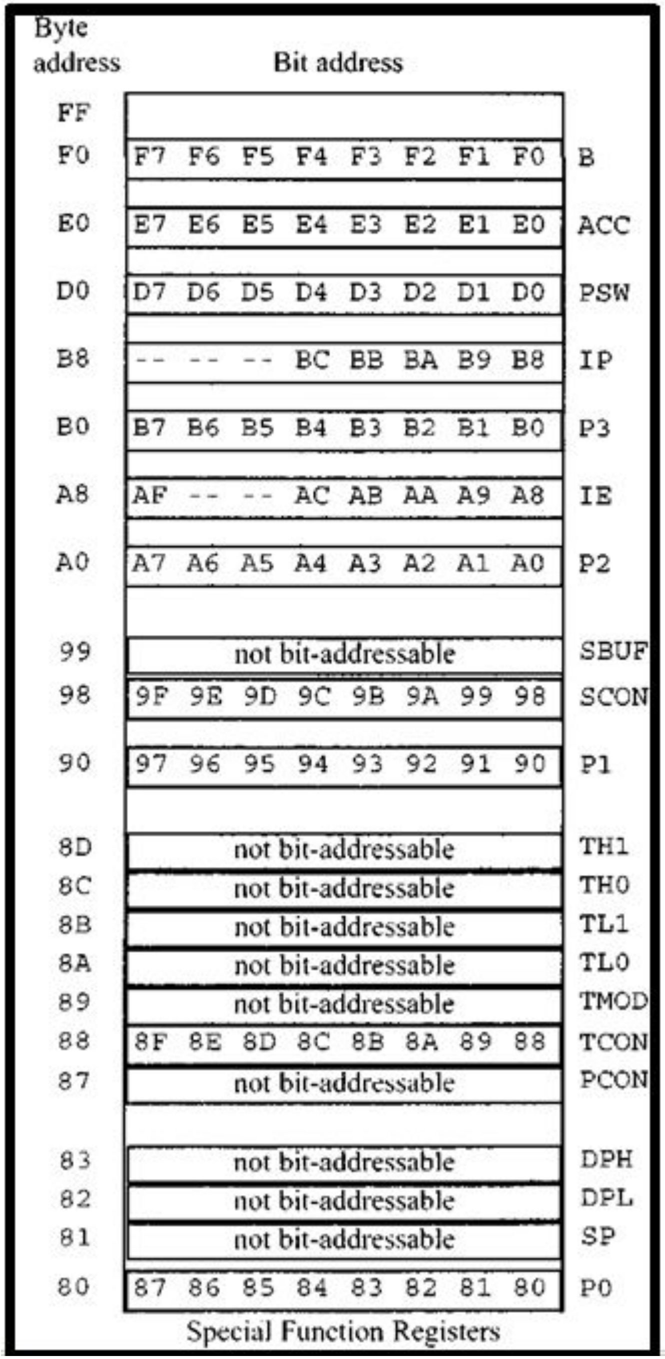
- **Register Banks:** 4 register banks, default is Bank 0. **Bank 1 uses same RAM space as stack.**

- Register banks are switched using register bank select bits of the PSW (bits D4 and D3).

- **Special Function Registers:** Accumulator, Register B, I/O Port latch registers, Stack pointer (SP), Data Pointer(DPTR), Processor Status Word (PSW) and various control registers.

- Some of these registers (B, ACC, PSW, T2CON, IE, etc.) are bit addressable. The bit address of a bit in the register is calculated as follows. Bit address of 'b' bit of register 'R' is: Address of register 'R' + b, where 0 ≤ b ≤ 7.

While all I/O ports are bit-addressable, Only registers A, B, PSW, IP, IE, ACC, SCON, and TCON are bit-addressable.



Byte address	Bit address								
FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
B8	--	--	--	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	--	--	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	not bit-addressable								SBUF
98	9F	9E	9D	9C	9B	9A	99	98	SCON
90	97	96	95	94	93	92	91	90	P1
8D	not bit-addressable								TH1
8C	not bit-addressable								TH0
8B	not bit-addressable								TL1
8A	not bit-addressable								TL0
89	not bit-addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	not bit-addressable								PCON
83	not bit-addressable								DPH
82	not bit-addressable								DPL
81	not bit-addressable								SP
80	87	86	85	84	83	82	81	80	P0

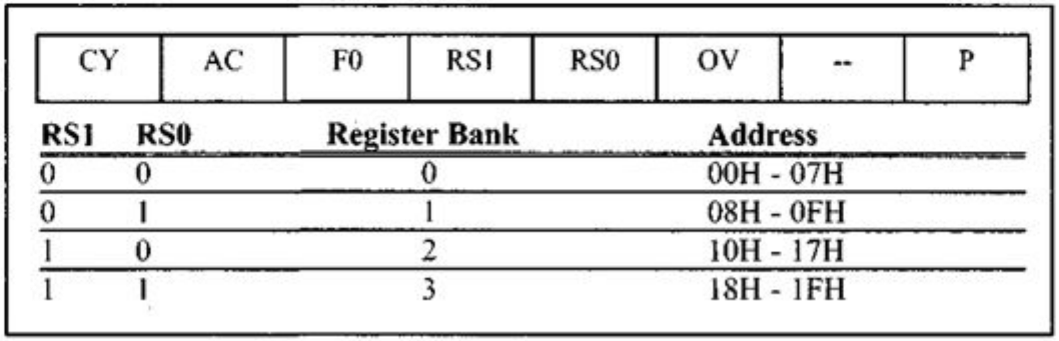
Special Function Registers

- **PSW/Flag register**(address D0H):stores important status conditions of the microcontroller. Also stores the bank select bits (RS1 & RS0) for register bank selection. 4 flags (CY, AC, P, OV), 2 for register bank select and remaining 2 are user definable.

- Upon RESET, bank 0 is selected. We can select any other banks using the bit-addressability of the PSW.

- WAP to save the Accumulator in R7 of Bank 2.

```
CLR PSW.3
SETB PSW.4
MOV R7,A
```



CY	AC	F0	RS1	RS0	OV	--	P

RS1	RS0	Register Bank	Address
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Stack in the 8051

- The stack is a section of a RAM used by the CPU to store information such as data or memory address on temporary basis. The CPU needs this storage area considering limited number of registers.

- Register bank 1 uses the same RAM space as the stack. Therefore we must either not use register bank 1, or allocate another area of RAM for the stack.

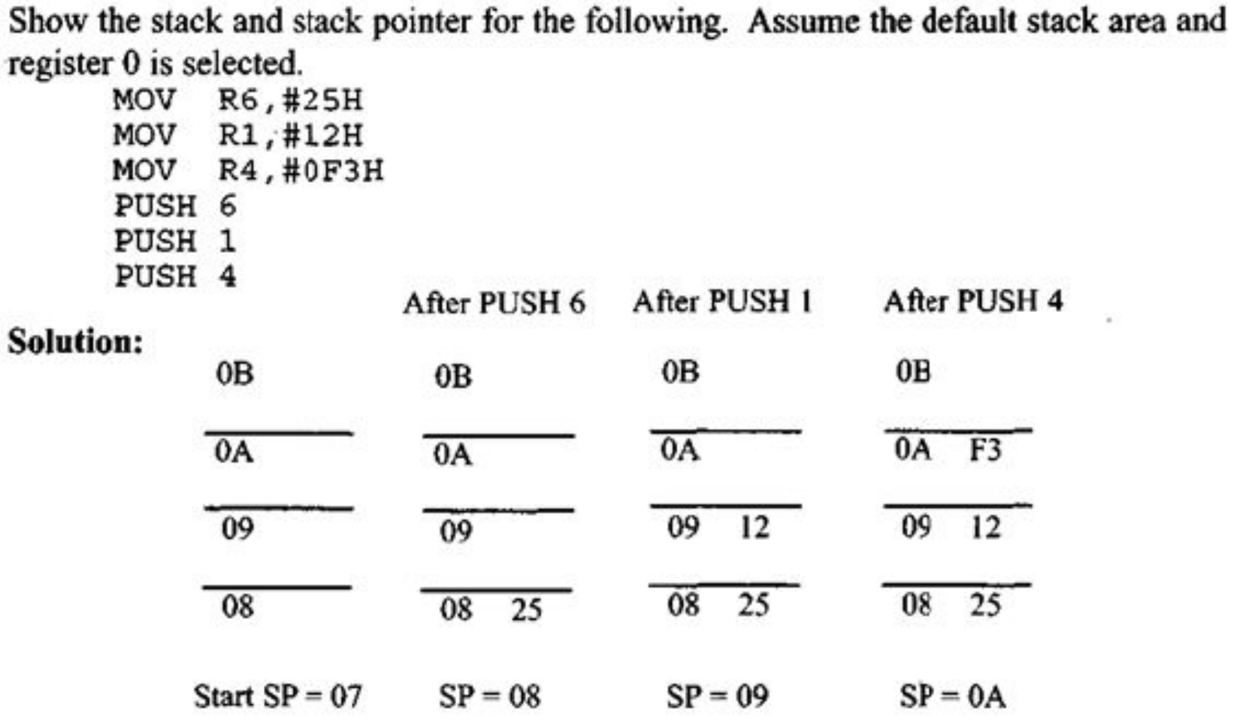
- The register used to access the stack is known as the **stack pointer register (SP)**. The stack pointer in the 8051 is 8-bits wide, and it can take a value of 00 to FFH.

- When the 8051 is initialized, the SP register contains the value 07H. This means that the RAM location 08 is the first location used for the stack.

- The storing operation of a CPU register in the stack is known as a **PUSH**, and getting the contents from the stack back into a CPU register is called a **POP**.

- In the 8051, SP points to the last used location of the stack. When data is pushed onto the stack, SP is incremented by 1. When PUSH is executed, the contents of the register are saved on the stack and SP is incremented by 1.

- With every pop operation, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is decremented once.



Show the stack and stack pointer for the following. Assume the default stack area and register 0 is selected.

```
MOV   R6,#25H
MOV   R1,#12H
MOV   R4,#0F3H
PUSH  6
PUSH  1
PUSH  4
```

Solution:

	After PUSH 6	After PUSH 1	After PUSH 4
0B	0B	0B	0B
0A	0A	0A	0A F3
09	09	09 12	09 12
08	08 25	08 25	08 25
Start SP = 07	SP = 08	SP = 09	SP = 0A

8051 common directives:

- **DB** (define byte) :define 8-bit data

- **ORG** (origin) : indicates the beginning of the address

- **EQU** (equate) : defines a constant without occupying a memory location

- **BIT** (bit): assign bit-addressable I/O and RAM locations

- **END** : indicates to the assembler the end of the source (asm) file

Example: Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a low-to-high pulse to port P1.5 to turn on a buzzer.

```
    OVEN_HOT BIT P2.3 ;assign the bit
    BUZZER BIT P1.5          ;assign the bit

HERE: JNB OVEN_HOT, HERE        ;monitor heat of oven
    ACALL DELAY         ;
    CPL BUZZER          ;sound buzzer
    ACALL DELAY
    SJMP HERE
```

Example: A switch is connected to pin PI .7 and an LED to pin P2.0. Write a program to get the status of the switch and send it to the LED.

```
    SW BIT P1.7     ;assign the bit
    LED BIT 2.0     ;assign the bit
HERE:   MOV C,SW     ;get bit from port
    MOV LED,C       ;send bit to port
    SJMP HERE       ;repeat forever
```

Example: Assume that RAM bit location 12H holds the status of whether there has been a phone call or not. If it is high, it means there has been a new call since it was checked the last time. Write a program to display “New Messages” on an LCD if bit RAM 12H is high. If it is low, the LCD should say “No New Messages”

```
     PHONEBIT BIT 12H
     MOV C,PHONEBIT          ;copy content of bit location 12H to Carry flag
     JNC NO                  ;if no carry jump to NO
     MOV DPTR, #400H         ;if yes carry load address of yes message
     LCALL DISPLAY           ;call display routine for display
     SJMP EXIT               ;exit
NO:  MOV DPTR, #420H          ;if no carry load address of no message
     LCALL DISPLAY
EXIT:

;data to be displayed on LCD
     ORG 400H
YES_MSG:DB “New Messages”,0
     ORG 420H
NO_MSG: DB “No Messages”,0
```

Time Delay for various 8051 chips.

- For the 8051 with 11.0592 MHz crystal oscillator, 1 machine cycle=reciprocal of (11.0592MHz/12)= 1.085 microsecond
- Delay subroutine consists of:

  (i) setting a counter

  (ii) a loop

```
          MOV A,#55H
AGAIN:    MOV P1,A
          AC ALL DELAY                                   CPL  A
          SJMP AGAIN

DELAY:    MOV R3,#200   ;1 m/c cycle
HERE:     DJNZ R3,HERE   ;2 m/c cycle
          RET              ;2 m/c cycle
```

Total no. of machine cycles in the delay routine=1+200x2+2=403

Total time delay= 403x1.085 microsecond=436.255 microseconds.

8051 I/O Programming:

- Total of 4 8-bit I/O ports configured as i/p ports upon reset (have FFH).

- **Port 0** : must be connected externally to 10K Ohm pull-up resistors, configured as i/p by writing all 1's, *dual role as multiplexed address and data pins when connecting 8051 to external memory*.

- Address latch enable [ALE] indicates if Port 0 has address or data. When ALE=0, it provides data D0 - D7, but when ALE = 1, it has address A0 - A7. With the help of 74LS373 latch, ALE is used for demultiplexing address and data.

- **Port 1** : can be use as an input or output. As compares to Port 0, this port does not need any pull-up resistors.

- Port 1 already has internally connected pull-up resistors. Port 1 is configured as a output port when reset.

- **Port 2 :** can also be used for both input and output port.

- To make Port 2 as an input, the Port 2 must be programmed by writing 1's to all bits.

- Port 2 has dual role. Port 2 is also designated as A8 - A15. P0 provides the lower 8 bits via A0 - A7 while Port 2 provides bits A8 - A15 of the address.

- **Port 3**: additional function of providing signals such as interrupts.
- P3.0 & P3.1 (RxD & Txd for serial comm.), P3.2 & P3.3 (external interrupts INT0 and INT1), P3.4 & P3.5 (timers 0 and 1), P3.6 & P3.7 (WR’ and RD’ for external memories)

8051 I/O ports and bit addressability

- Powerful & widely used feature of the 8051 family: access individual bits of the port without altering the rest of the bits in that port. (SETB P1.5)

- Single bit instructions: SETB, CLR, CPL, JB, JNB, JBC

- JNB and JB allows monitoring a bit and then taking decisions depending on whether it is 0 or 1.

- Carry flag can be used to save/ examine status of a single bit of port. (MOV C, Px.y)(there are instructions such as JC & JNC to check the carry flag)

- Read-Modify-Write feature of a port: (**ANL, ORL, XRL,CPL, DEC, INC, etc**)

```
          MOV P1, #55H
AGAIN:    XRL P1,#0FFH
          ACALL DELAY
          SJMP AGAIN
```

Some 8051 instructions:

- MOV, MOVC, MOVX
- SJMP and LJMP
- JNC and JC
- JNB and JB
- ACALL and LCALL
- RET
- CLR and SETB
- CPL and XRL
- DJNZ
- PUSH and POP

CALL Instructions

CALL is used to call a subroutine or method. There are two instructions − LCALL and ACALL.

- **LCALL (Long Call)**
  - LCALL - 3-byte instruction, first byte represents the opcode and second and third bytes are used to provide the address of the target subroutine.
  - LCALL can be used to call subroutines which are available within the 64K-byte address space of the 8051.

- **ACALL (Absolute Call)**
  - ACALL - 2-byte instruction, in contrast to LCALL which is 3 bytes.
  - The target address of the subroutine must be within 2K bytes because only 11 bits of the 2 bytes are used for address.

Unconditional Jump Instructions

There are two unconditional jumps in 8051 −

- **LJMP (long jump)** −
  - 3-byte instruction
  - first byte represents opcode,
  - second and third bytes represent the 16-bit address of the target location.
  - The 2-byte target address is to allow a jump to any memory location from 0000 to FFFFH.

- **SJMP (short jump)** −
  - 2-byte instruction,
  - first byte is the opcode
  - second byte is the relative address of the target location.
  - The relative address ranges from 00H to FFH which is divided into forward and backward jumps; that is, within –128 to +127 bytes of memory relative to the address of the current PC.
  - In case of forward jump, the target address can be within a space of 127 bytes from the current PC. In case of backward jump, the target address can be within –128 bytes from the current PC.

Example Programs:

- **WAP to generate a square wave(duty cycle=50%) on bit 0 of port 1.**

```
    HERE: SETB P1.0
          LCALL DELAY
          CLR P1.0
          LCALL DELAY
          SJMP HERE
```

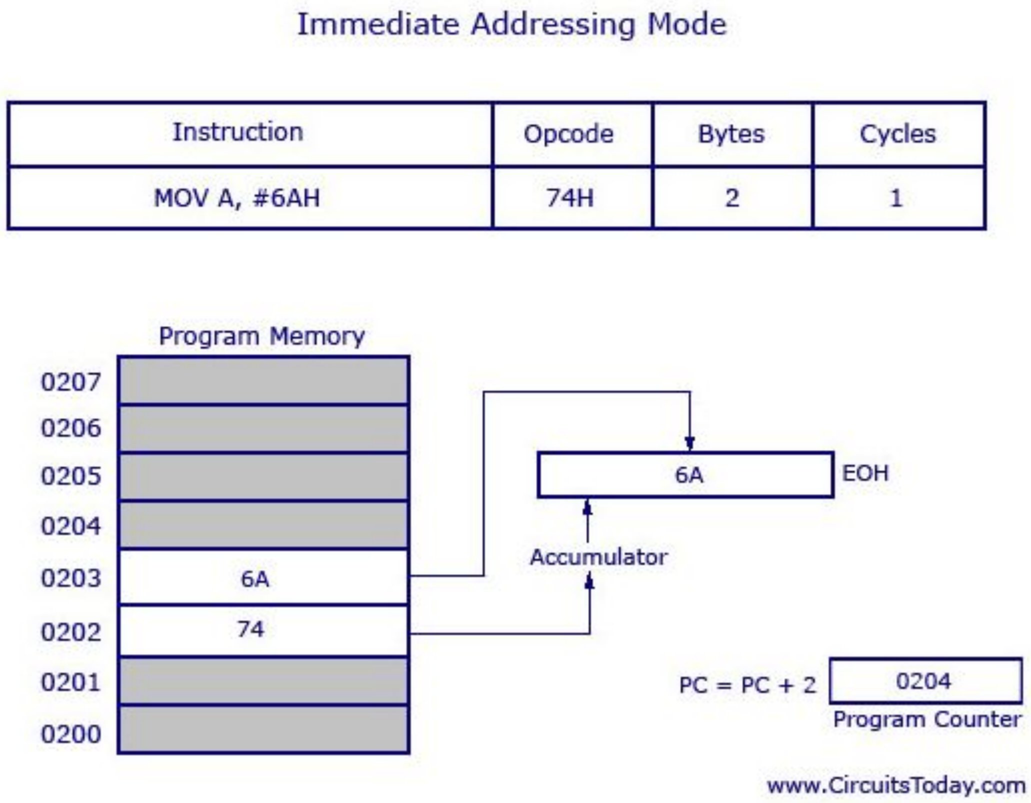
- **WAP to monitor bit 1 of port 0 continuously and when high, read in the data from port 1. At the same time send a low-to-high pulse on P0.2 (connected to an LED) to indicate that data has been read.**

```
          SETB P0.1
          MOV P1, #0FFH
AGAIN:    JNB P0.1,  AGAIN
          MOV A,P1
          CLR P0.2
          SETB P0.2
```

8051 Addressing Modes: a way to address an operand

Immediate: load information into any of the registers including DPTR

- MOV A,#25H
- MOV DPTR,#4522H
- MOV DPL,#22H



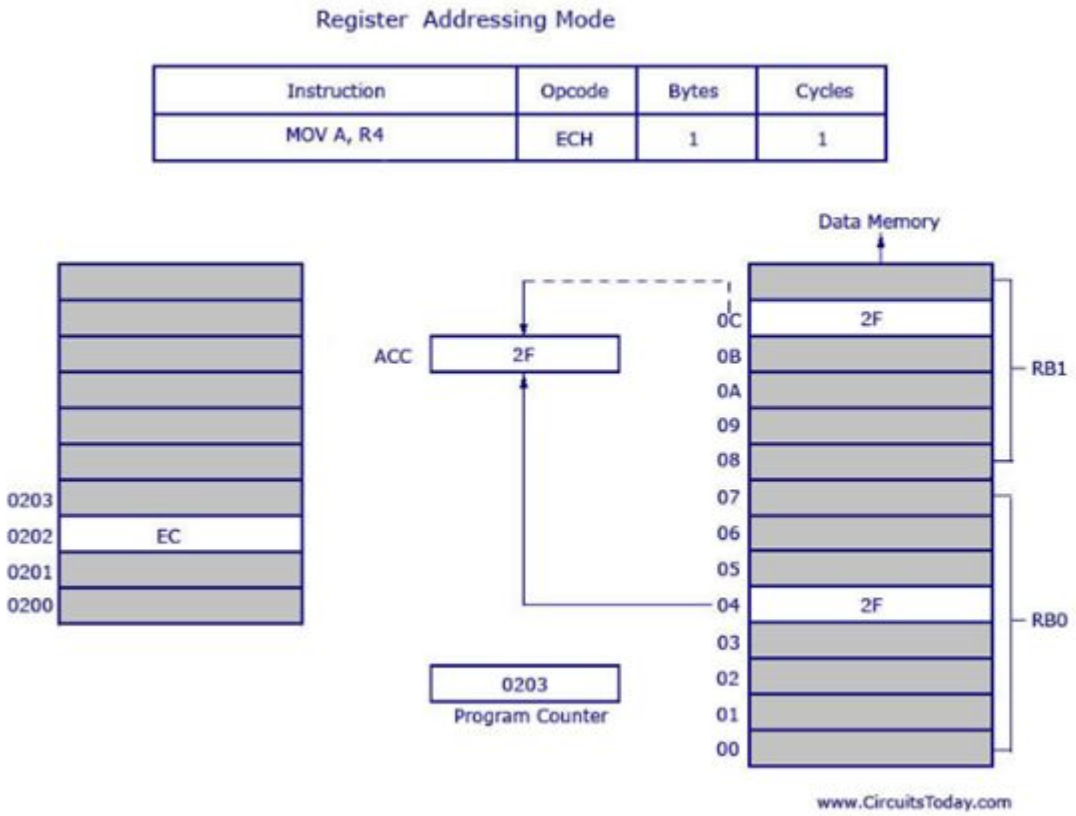
Immediate Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, #6AH	74H	2	1

www.CircuitsToday.com

Register: source and destination should be of same size

- Data is transferred to accumulator from the register (based on which register bank is selected).
- Movement of data between R_n registers is not allowed



Register Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, R4	ECH	1	1

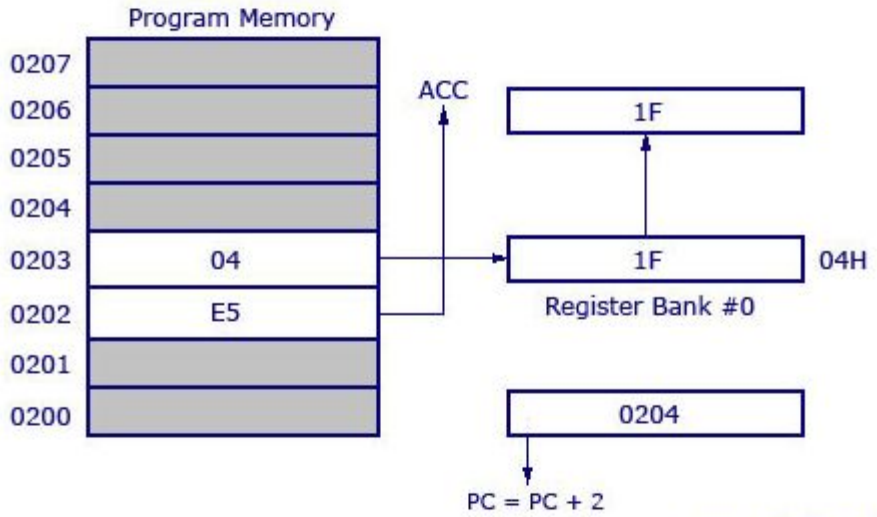
www.CircuitsToday.com

Direct: Here the address of the data (source data) is given as operand. Most often used to access RAM locations 30-7FH

- Only direct addressing mode is allowed for stack operations
- MOV R0,40H
- PUSH A is invalid

Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, 04H	E5	2	1



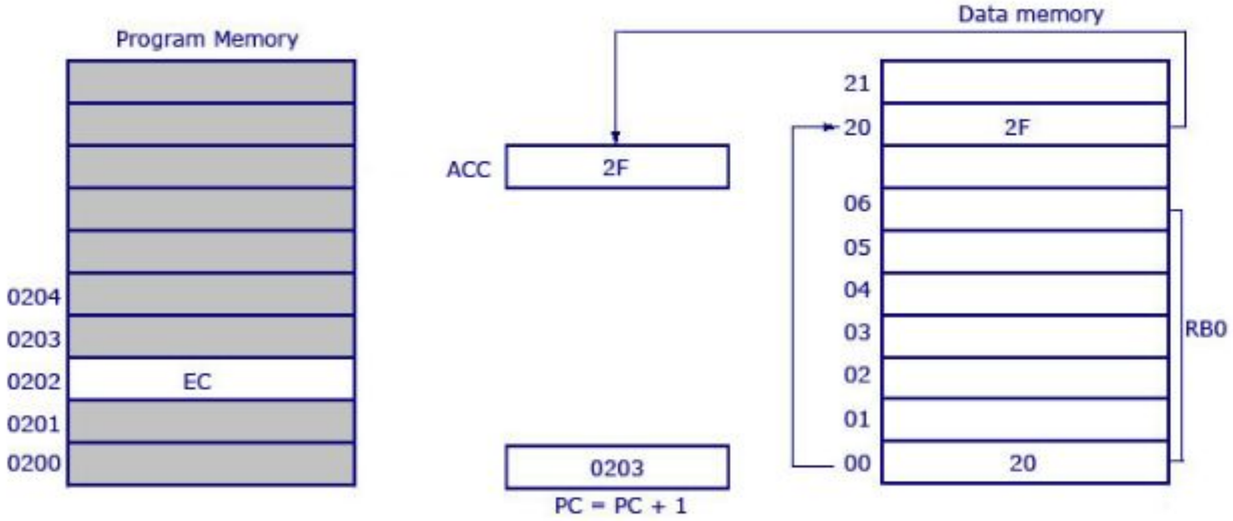
www.CircuitsToday.com

Register Indirect: register is used as a pointer to data

- Only R0, R1 are used to hold addresses of RAM locations
- For accessing externally connected RAM or on-chip ROM, 16-bit pointers are required.
- Makes data accessing dynamic
- MOV A,@R0

Register Indirect Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, @ R0	E6H	1	1



www.CircuitsToday.com

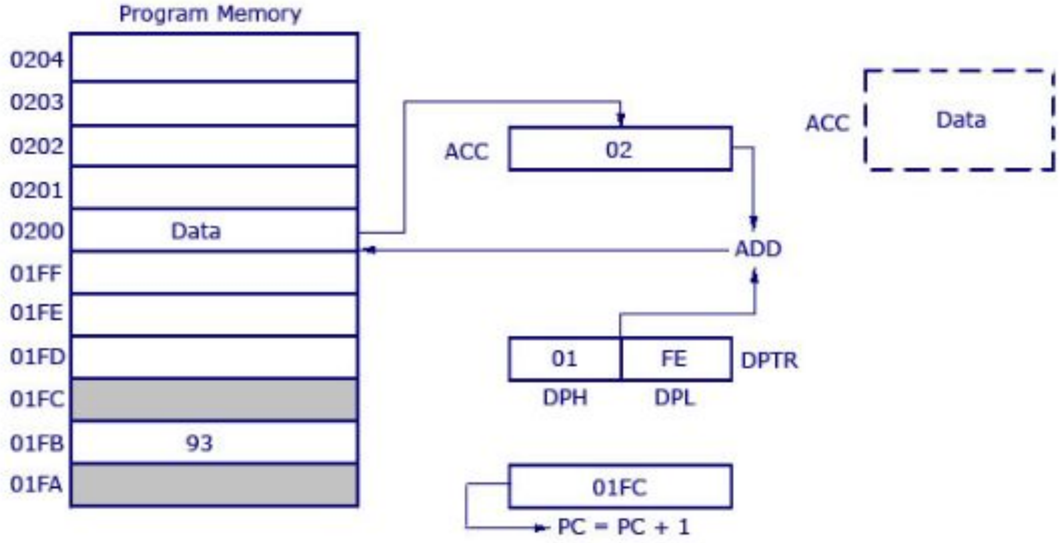
Indexed: widely used in accessing data elements of look-up table entries located in program ROM space of 8051

- MOVC A,@A+DPTR

- With Indexed Addressing Mode, the effective address of the Operand is the sum of a base register and an offset register. The Base Register can be either Data Pointer (DPTR) or Program Counter (PC) while the Offset register is the Accumulator (A).

Indexed Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOVC A,@A +DPTR	93H	1	2



Indexed Addressing Mode and the MOVX instruction

- 8051 has 64K bytes of code space ROM under control of 16-bit PC.
- MOVC is used to access a portion of this 64K byte code space.
- 8051 has another 64K bytes of memory space set aside for data storage which is external & accessed by MOVX instruction

Example Programs:

- WAP to copy values 55H into RAM locations 40H to 45H using:
- (i) direct addressing mode
  MOV A,#55H
- MOV 40H,A
- MOV 41H,A
- MOV 42H,A
- MOV 43H,A
- MOV 44H,A
- MOV 45H,A

- (ii) indirect addressing mode with loop
- MOV A, #55H
- MOV R0,#40H
- MOV R2,#05H
- AGAIN: MOV @R0,A
- INC R0
- DJNZ R2,AGAIN

Example program (indexed addressing)

- WAP to get a value from P1 and send the square of the value to P2 continuously

```
            ORG 0
            MOV DPTR,#300H
            MOV A,#0FFH
            MOV P1,A
    BACK:       MOV A,P1
            MOVC A,@A+DPTR
            MOV P2,A
            SJMP BACK

            ORG 300H
  XSQR_TABLE:
            DB 0,1,4,9,16,25,36,49,64,81
            END
```

(1) The result of a signed arithmetic operation is stored in RAM location 27H.Check if the result is positive, if so send a low value to P1.7 else send a high value to P1.7

```
MOV A,27H       ;move content of RAM location 27H to A
     JNB ACC.7, POS       ;check msb of acc for sign bit
     SETB P1.7       ;if sign bit is high, it is a –ve no.,so set P1.7          SJMP NEXT
POS:         CLR P1.7
NEXT:        NOP
     END
```

Try:

(2) WAP to get the status of a switch connected to P1.7 and glow an LED connected to P2.0 to indicate the status.

Arithmetic & Logical Instructions:

- Arithmetic: ADD, ADC, DA, SUBB, MUL, DIV

- Logical: ANL, ORL, XRL, CPL, CJNE, RR, RL, RRC, RLC

- The DA instruction: Decimal adjust for addition

  : Only operand is Accumulator

  : Adds 6 to the lower nibble or higher nibble if needed, otherwise will leave the result alone

  :Works only after ADD not INC

- The rotate instructions for data serialization

Example Program

(3) P1 is an i/p port connected to a temperature sensor. WAP to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following:

If T=75, A=75; if T<75, R1=T; if T>75, R2=T

```
   MOV P1,#0FFH      ;P1 as i/p port
   MOV A,P1       ;read P1
   CJNE A,#75, OVER ;jump if A not = 75
   SJMP EXIT      ;A=75 exit
OVER:JNC NEXT        ;CY=0 => A>75
   MOV R1,A       ;CY=1 => A<75
   SJMP EXIT
NEXT: MOV R2,A
EXIT:…….
```

Rotate instruction & Data Serialization

- Rotate instructions work on the Accumulator
- 2 types: (i) simple rotation (RR,RL) (ii) rotation through carry(RRC,RLC)
- 2 ways to transfer a data byte serially:
- (i) using serial port
- (ii)transfer 1 bit at a time and control sequence of data and spaces in between them.

```
RRC A
MOV P1.3,C
```

Try:

- (4) WAP to bring in data in serial form (say, P0.0) and send it out in parallel form (say,P1)

- (5) WAP to take data from P0.0 to P0.3 and display it on 4 LEDs connected to P1.4 to 1.7 (use the SWAP instruction which works only on the Accumulator)

Solution to (4)

```
        MOV R0,#08H       ;set R0 as counter for 8 bits
        SETB P0.0       ;configure P0.0 as i/p
BACK:   MOV C, P0.0       ;??
        RRC A           ;??
        DJNZ R0, BACK     ;??
        MOV P1,A        ;??
        END
```

Solution to (5)

```
        MOV P0,#0FFH      ;make P0 an i/p port
        MOV A, P0
        ANL A, #0FH
        SWAP A          ;swap nibbles
        MOV P1,A
```

Checksum byte in ROM

- To ensure data integrity of ROM contents

- Detects any corruption of ROM contents

- Uses an extra byte called a checksum byte, tagged at the end of a series of bytes of data

- Checksum is calculated by adding all the bytes and dropping the carries and then taking 2’s complement

- While checking , add all the bytes including checksum, drop carry, result should be 0. Non zero result implies data has been corrupted.

Accessing External Memory:

- Access to external program memory uses the o/p signal PSEN’ (Program store enable) as the read strobe. It should be connected to the OE pin of ROM.

- Access to external data memory uses RD’ and WR’(alternate function of P3.7 and P3.6).

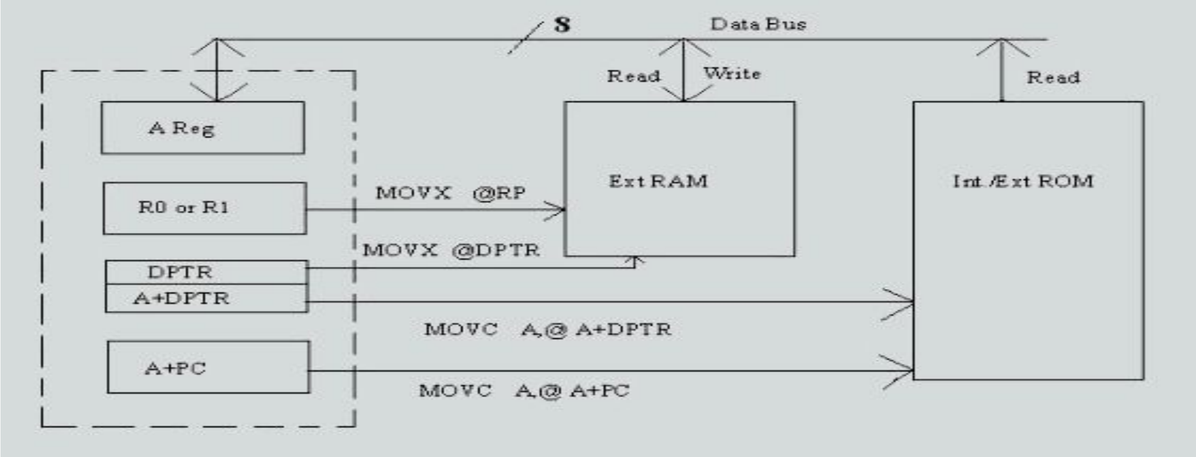
- For external program memory, always 16 bit address is used, access to external data memory can be either 8-bit address or 16-bit address.

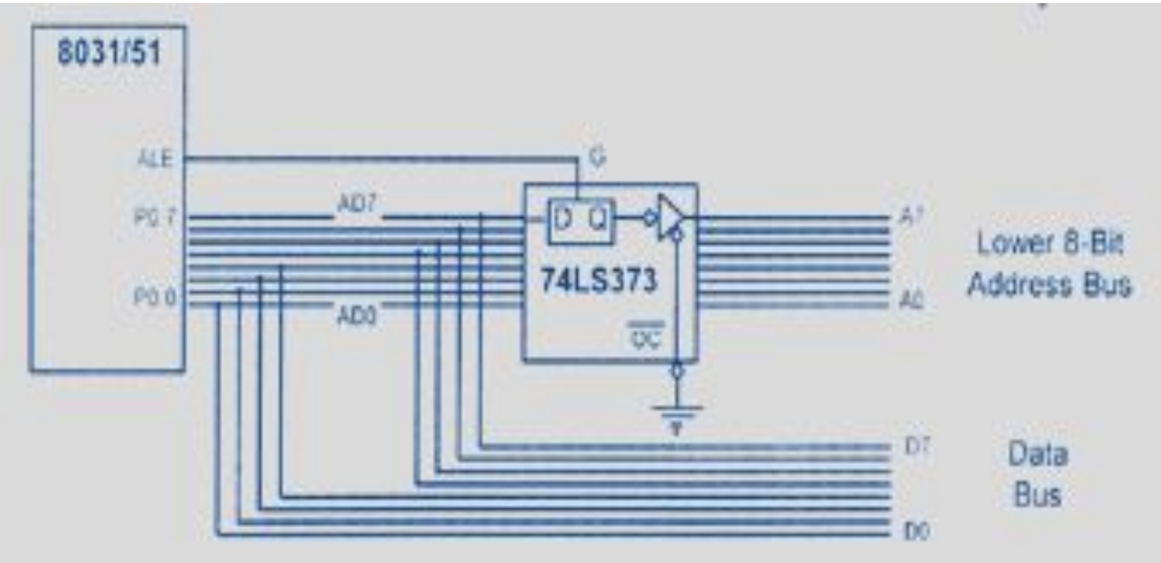
- The address is sent though P0 (A0-A7)& P1 (A8-A15). P0 is also used to send data (D0-D7).(Uses ALE pin to select between address and data)

- External program memory is accessed under the following condition, whenever EA’ is low, or whenever PC contains a number higher than 0FFFH (for 8051) or 1FFF (for 8052).

- External program memory can be not only used to store the code, but also for lookup tables of various functions required for a particular application and can be accessed by the MOVC instruction.

Schematic diagram of external memory access

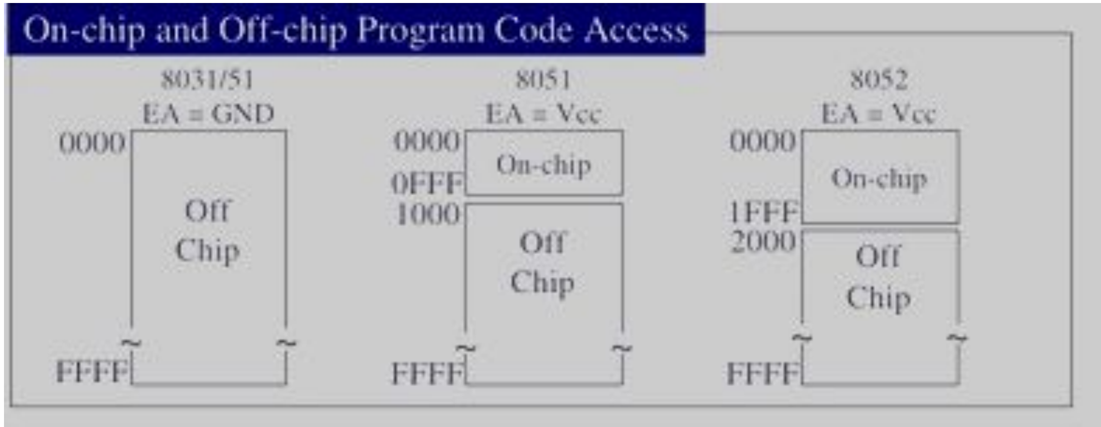




On-chip & Off-chip ROM

- Both on-chip & off-chip ROM can be used at the same time. On-chip ROM can be used for boot code & external ROM(using NVRAM) can be used for user program.

- Here, EA=VCC, upon reset 8051 executes on-chip program first, then it switches to external ROM.



Data Memory Space

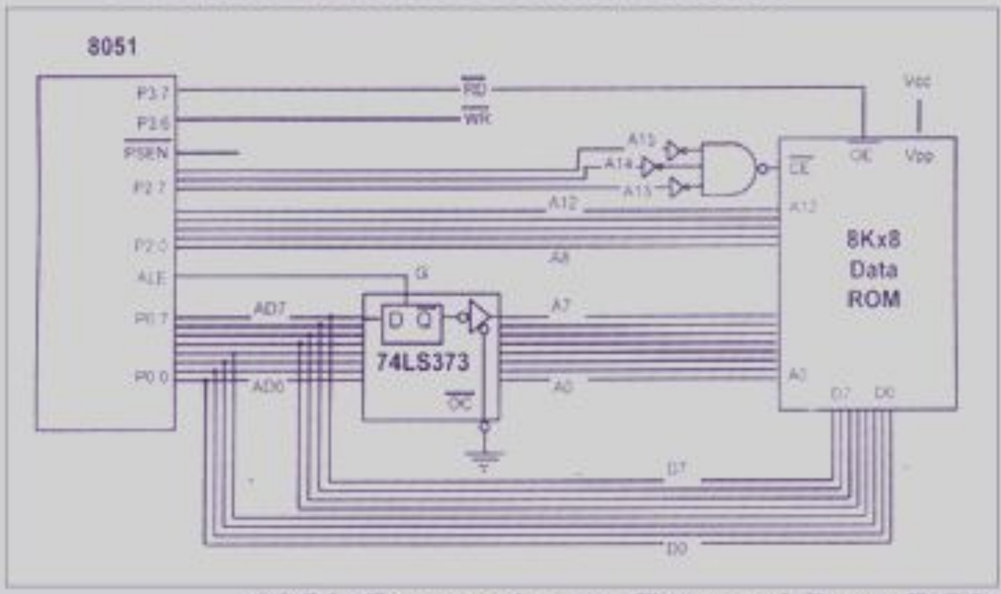
- 8051 family has 64Kbytes of Code Space (16bit PC) and also 64Kbytes of Data Space(16bit DPTR, MOVX instruction).

- For external ROM containing data, RD signal is used. For ROM containing program code, PSEN signal is used to fetch the code.

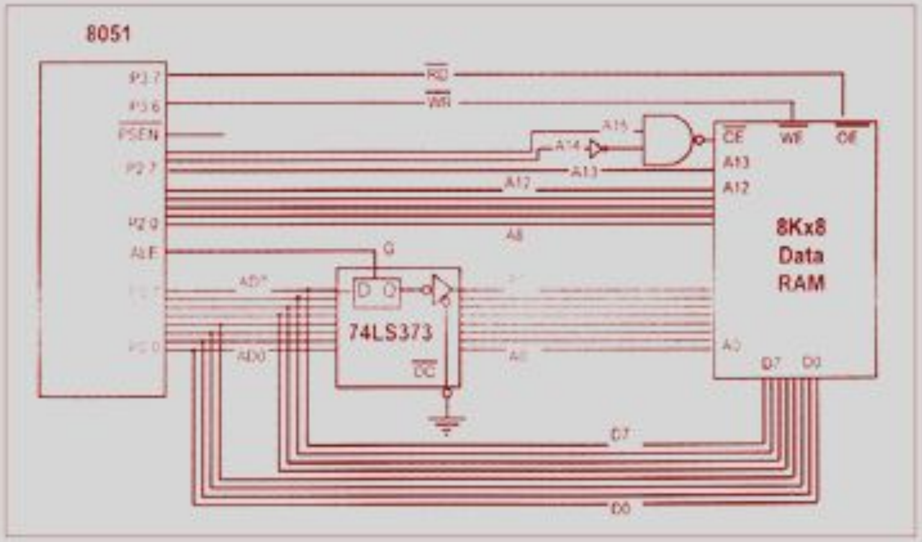
- To connect the 8051 to external SRAM both RD and WR signals are used.

- MOVC A, @A+DPTR gets data in the code space. MOVX A, @DPTR gets data in the data space of the microcontroller.

- MOVX @DPTR, A writes data to the external RAM.



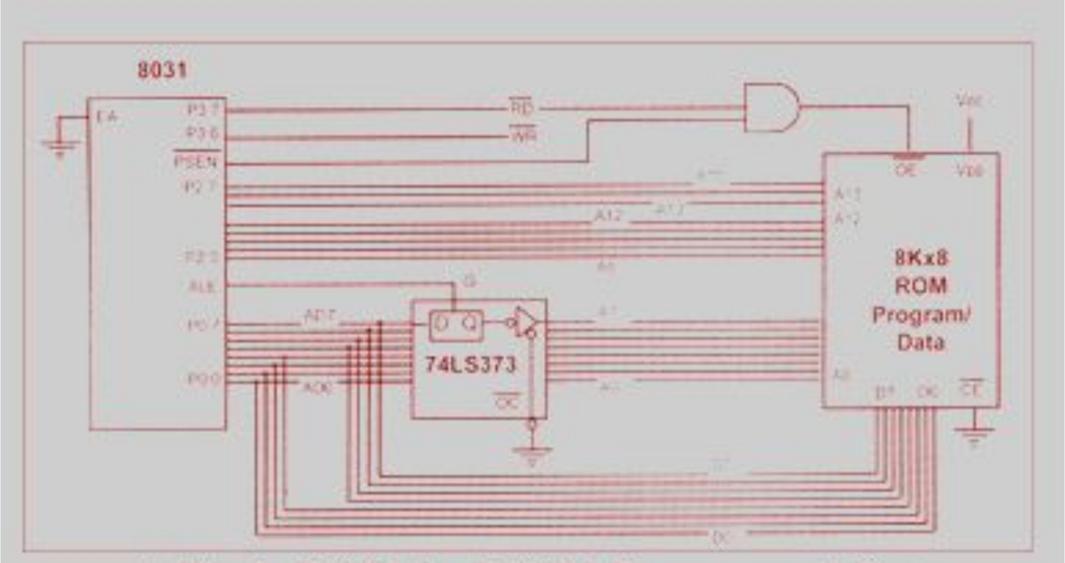
8051 Connection to External Data ROM



8051 Connection to External Data RAM

Single ROM for both program & data

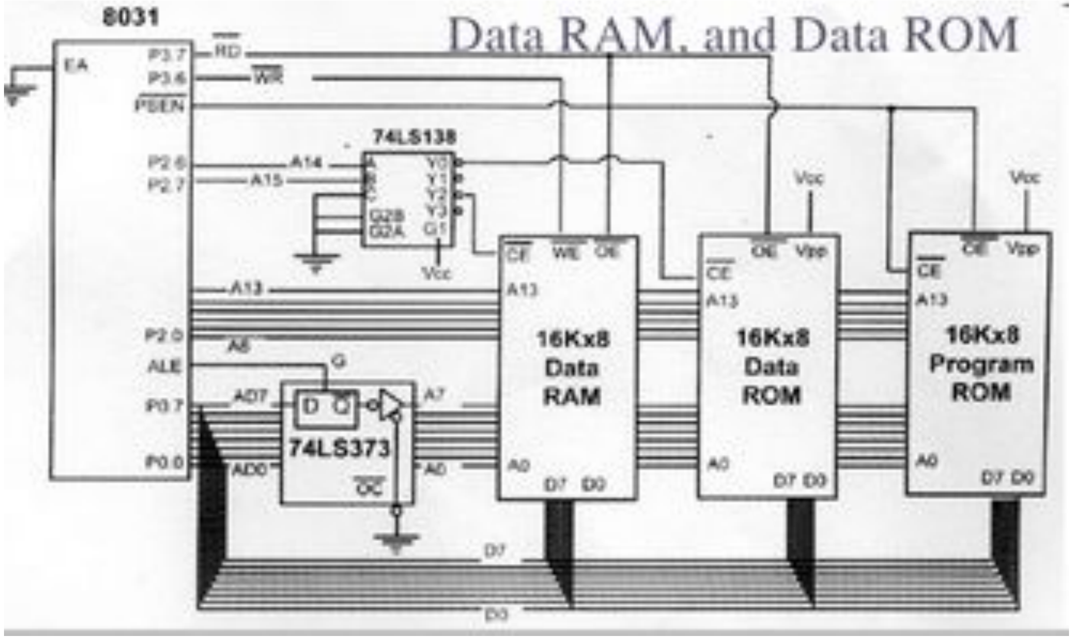
- Single 64Kx8 external ROM chip to be used for both program code and data storage.
- 0000H-7FFFH : program code;D000H-FFFFH: data
- PSEN: is used to access external code space
- RD: is used to access external data space
- AND gate is used with PSEN & RD as i/ps and the o/p is connected to the OE pin of the program/data ROM.



A Single ROM for BOTH Program and Data

Memory Design Problems:

- Assume that we need an 8031 system with 16KB of program space, 16KB of data ROM starting at 0000H, and 16KB of NVRAM starting at 8000H. Show the design using a 74LS138 for the address decoder.



Find the address space of the memory chips in the above design.

For the Data RAM, A15 & A14 =10 for it to be selected.

Address space=1000 0000 0000 0000 to 1011 1111 1111 1111

For the Data ROM, A15 & A14=00 for it to be selected.

Address Space=0000 0000 0000 0000 to 0011 1111 1111 1111

For the Program ROM, A15 & A14 are don't cares.

Interfacing Large External Memory

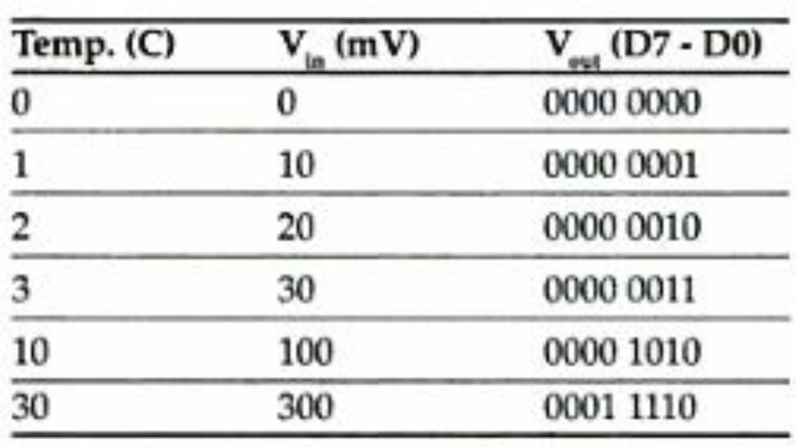
- 8051 can support only 64K bytes of external data memory because the DPTR is a 16 bit register

- If large external memory is required, A0-A15 of 8051 are connected to the address pins of the external memory and the rest of the address bits are provided through some port pins.

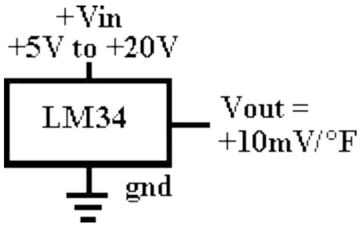
- Example: External Memory is a 256K byte NV-RAM.
  - Address bits reqd.=18 (A0-A17). A0-A15 are provided through A0-A15 of the 8051. Remaining 2 bits through say, P1.0 & P1.1
  - Also P1.2 is connected to the chip select line of the NV-RAM
  - The 256K byte memory chip is broken into 4 blocks of 64K bytes each.
  - Block Address Space:
    - 00000H-0FFFFH (P1.1 & P1.0=00)
    - 10000H-1FFFFH(P1.1 & P1.0=01)
    - 20000H-2FFFFH(P1.1 & P1.0=10)
    - 30000H-3FFFFH(P1.1 & P1.0=11)

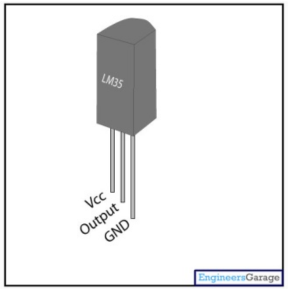
8051 connection to ADC0848 and LM34/LM35

- ADC0848 has 8-bit resolution with a max. of 256 steps
- LM34/35 produces 10mv for every degree of temperature change
- We can condition Vin of ADC0848 to produce a Vout of 2560mV (2.56V) for full scale o/p, therefore Vref=2.56V



Temp. (C)	V_{in} (mV)	V_{out} (D7 - D0)
0	0	0000 0000
1	10	0000 0001
2	20	0000 0010
3	30	0000 0011
10	100	0000 1010
30	300	0001 1110





8051 connection to ADC0848 and LM34/LM35

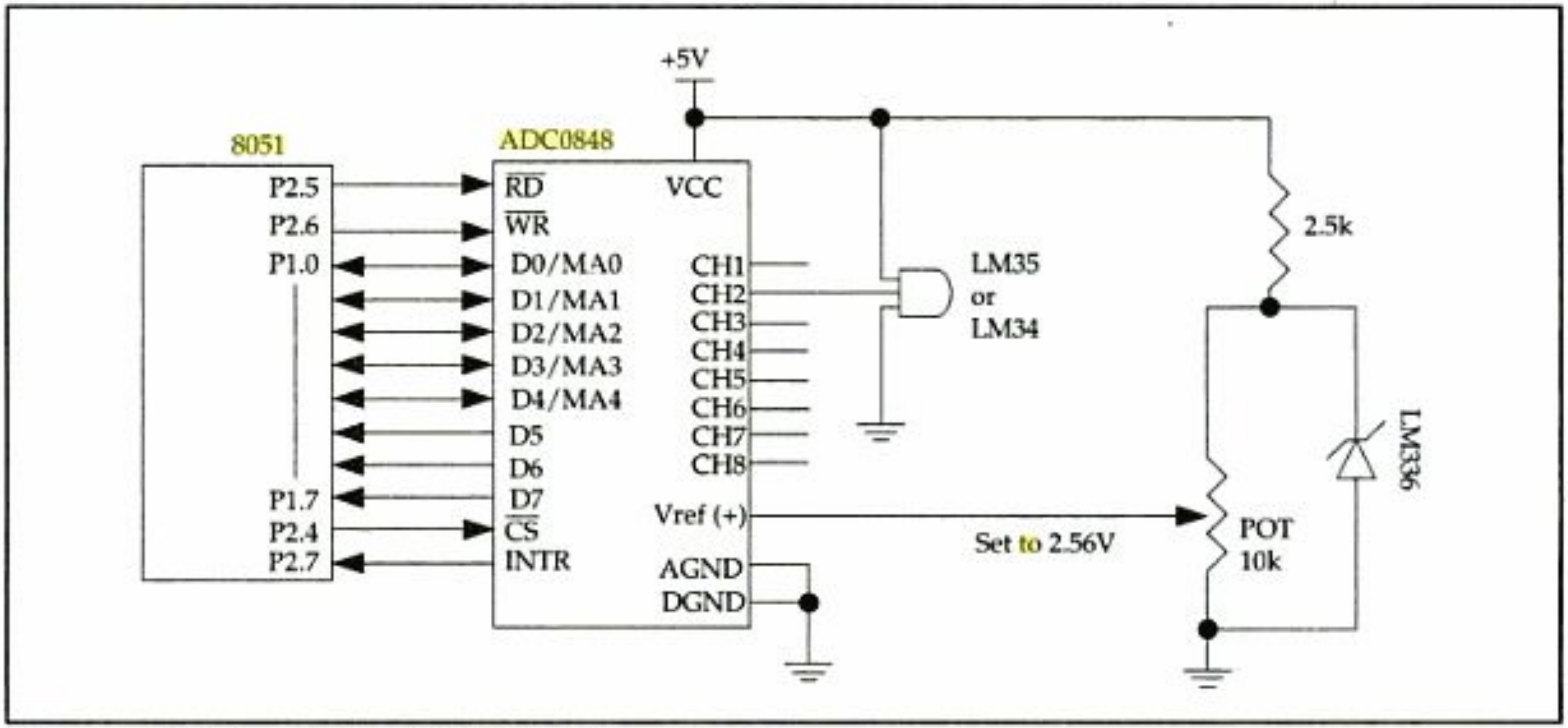


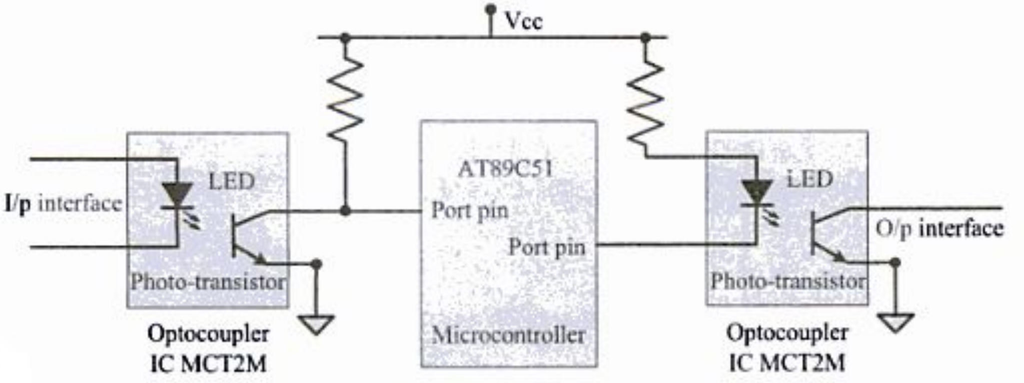
Figure 8051 Connection to ADC0848 and Temperature Sensor

Assembly code for reading and displaying temperature

```
 RD BIT P2.5
 WR BIT P2.6          ;start-conversion
 INTR BIT P2.7        ;end-of-conversion
 MYDATA EQU P1            ;P1.0-P1.7=D0-D7 OF ADC0848
 MOV P1,#0FFH             ;P1 is i/p port
 SETB INTR
BACK: CLR WR
 SETB WR              ;L-to-H to start conversion
HERE:JB INTR, HERE        ;wait for end of conversion
 CLR RD
 MOV A,MYDATA
 ACALL CONVERSION         ;hex to ascii
 ACALL DATA_DISPLAY       ;display data
 SETB RD              ;make RD=1 for next round
 SJMP BACK
```

The I/O System

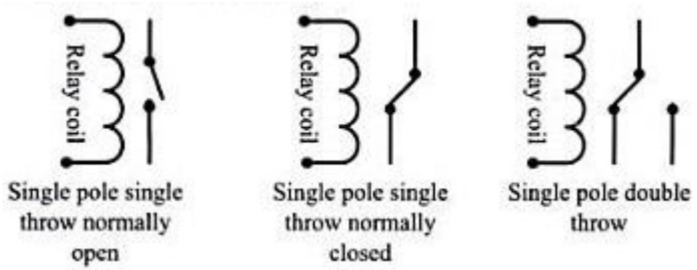
- Facilitates interaction of the ES with external world
- Interaction may happen through sensors & actuators connected to the I/O ports of the ES
- Interfacing may be direct or through signal conditioning and translating systems like ADCs, optocouplers, etc
- **LEDs**: can be used as an indicator for the status of various conditions
- 7-segment LED displays for displaying alphanumeric characters
- **Optocouplers**: is a solid state device to isolate two parts of a circuit.
- It is a device that uses an optical transmission path to transfer a signal between elements of a circuit, while keeping them electrically isolated.
- Combines an LED and a photo transistor in a single housing
- Can be used for suppressing interference in data communication, circuit isolation, high voltage separation, etc



Optocoupler in Input and Output circuit

I/O system (contd…)

- **Relay**:
  - an electro mechanical device which acts as dynamic path selectors for signals and power.
  - Converts magnetic flux created by an electrical signal into a mechanical force which cause the switch to open or close.
  - Consists of insulated wire on metal core & a metal armature with one or more contacts
  - Relay is controlled using a relay driver ckt connected to port pin of processor/controller.
  - Special relays (not bulky) called Reed relays are available for ES requiring switching of low voltage DC signals

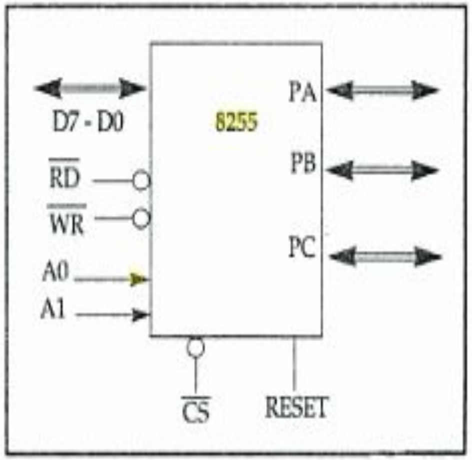

Relay coil
Single pole single throw normally open
Relay coil
Single pole single throw normally closed
Relay coil
Single pole double throw

- **Piezo** buzzer:
  - Piezo electric device for generating audio indications in embedded application
  - 2 types: self driving & external driving
  - Can be interfaced directly to controller port pin
- **Push Button switch**: push-to-make and push-to-break
- **Keypad**:
- **Stepper motors**: produces discrete rotation in response to applied dc voltage

The 8255 PPI

- 40 pin DIP chip having 3 separately accessible ports (8-bit) namely ports A,B and C.
- Individual ports can be programmed to behave as i/p or o/p ports and can be changed dynamically.

PA0-PA7: can be programmed as all i/p or all o/p or all bits as bidirectional input/output.
PB0-PB7: can be programmed as all i/p or all o/p but not as bidirectional
PC0-PC7: can be all i/p or all o/p. can be split as CU & CL. Each can be used for i/p or o/p.
RD' and WR': RD' & WR' lines of 8051 are connected to these lines
D0-D7: data pins of 8255 are connected to data pins of 8051
RESET: clear control register, all ports initialized as i/p ports
A0, A1 and CS': CS' selects the entire chip and A0 and A1 select specific ports.



Mode Selection:

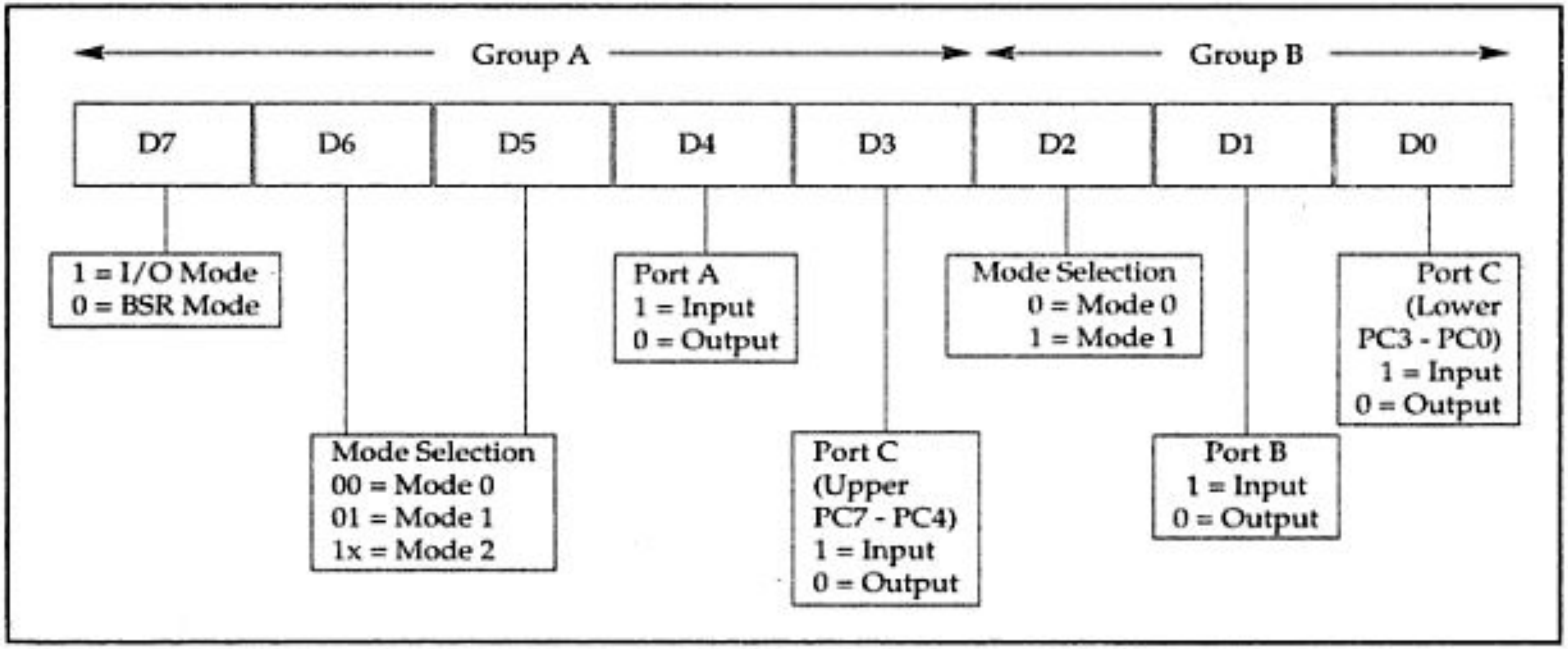
- **Control Register** must be programmed to select the operation mode of the 3 ports. Its content is called the control word. It is divided into 2 blocks, group A control (controls port A and port C upper) & group B control (controls port B & port C lower).

- **Mode 0** (Basic I/O mode or simple I/O): ports are used for simple I/O operations. No handshaking is used.

- **Mode 1**: Port A & port B can be used as i/p or o/p port in handshake mode, and port A uses 3 bits and port B uses 3 bits of port C to generate or accept handshake signals. Remaining 2 bits are used as general purpose I/O

- **Mode 2**: Port A can be configured as an 8-bit bi-directional port and 5 bits of port C are used as handshaking signals. Port B can be used either in mode 0 or in mode 1.

- **BSR Mode**: Only the individual bits of port C can be programmed.



***8255 control word format(I/O mode)**

D7=0:port C operates in BSR mode. Any of the 8 bits of port C is selected by using D3, D2 & D1 and it is set/ reset by D0. These can be used to generate strobe signals for controlling external devices.

D7=1:selects I/O functions, bits D6-D0 determines I/O function operation in the 3 modes, mode 0, mode 1 and mode 2.

TRY: Find the control word for the following configurations:

(a) All the ports A, B, C are o/p ports(mode 0)

(b) PA=in, PB=out, PCL=out and PCH=out.

I/O Addressing

- 8255 can be interfaced with the processor by 2 methods: Isolated I/O or Memory Mapped I/O.

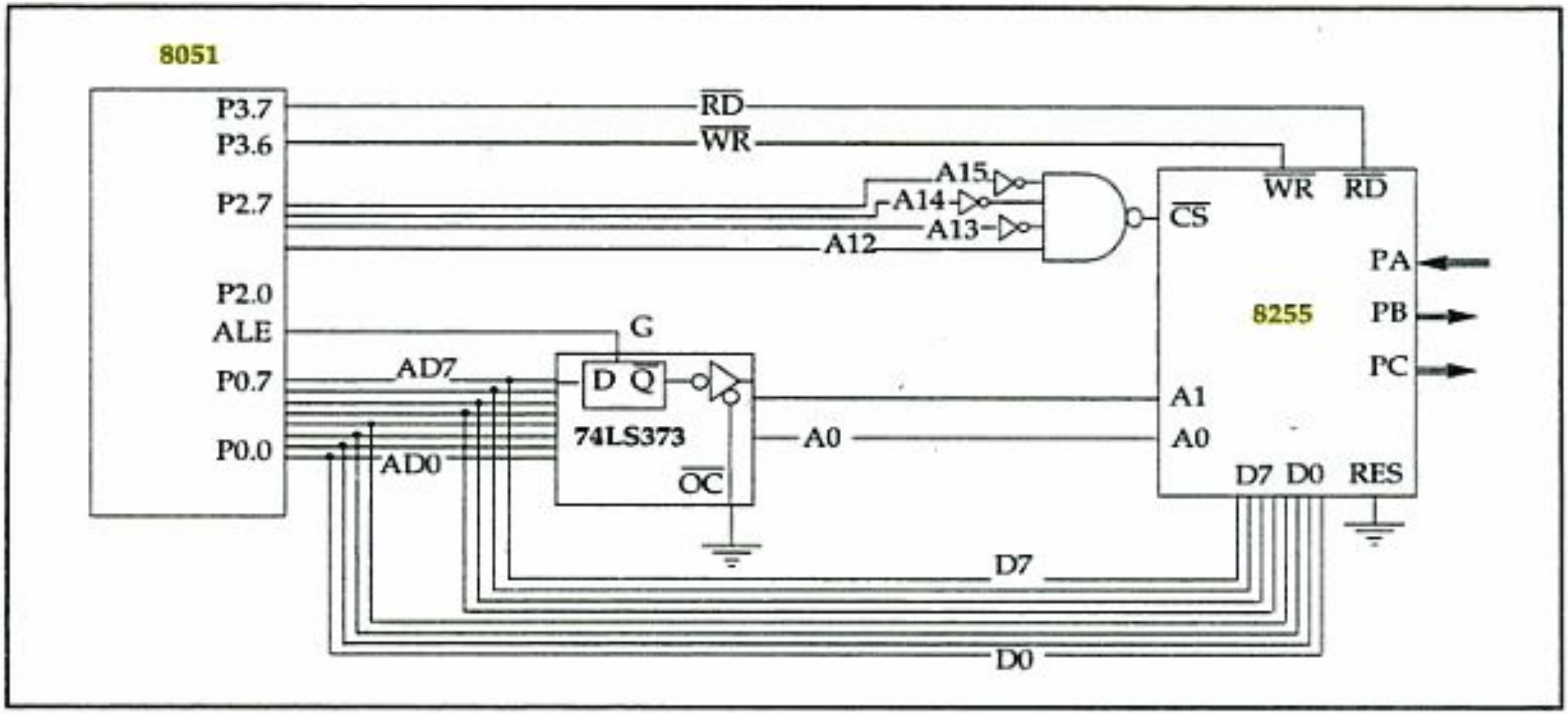
- Intel processors 8085, 8086 use both methods, 8051 has no separate I/O instructions and hence uses only memory mapped I/O for interfacing peripherals. (we use memory space to access the I/O devices)

- Therefore the interfacing of 8255 to 8051 is same as that of interfacing RAM memory (using MOVX).

- 8255 is programmed in any of the 4 modes by sending the control word to the control register.

- I/O port mapping requires finding port addresses for ports A, B, C and the control register.

8051 connection to 8255



(i) Find the I/O port addresses assigned to ports A, B, C and the control register
(ii) Find the control byte for PA=in, PB=out and PC=out
(iii) WAP to get data from PA and send it to both ports B and C
(IV) WAP to generate a square wave at bit 0 of port C

Solution:

(i) Address for PA=1000H
Address for PB=1001H
Address for PC=1002H
Address for control register=1003H

(ii) Control word=10010000=90H

```
(iii) MOV A,#90H
      MOV DPTR,#1003H
      MOVX @DPTR,A
      MOV DPTR,#1000H
      MOVX A,@DPTR
      INC DPTR
      MOVX @DPTR,A
      INC DPTR
      MOVX @DPTR,A
```

- (iv) To have the square wave at PC0, PCL should be configured as an o/p port. All other ports may be i/p or o/p. Hence control word=80H.

```
        MOV A,#80H
        MOV DPTR,#1003H
        MOVX @DPTR,A
        MOV DPTR,#1002H
START:  MOV A,#01H
        MOVX @DPTR,A
        ACALL DELAY
        MOV A,#00H
        MOVX @DPTR,A
        ACALL DELAY
        SJMP START
        END
```

Interfacing I/O Devices with the 8051 using the 8255 PPI

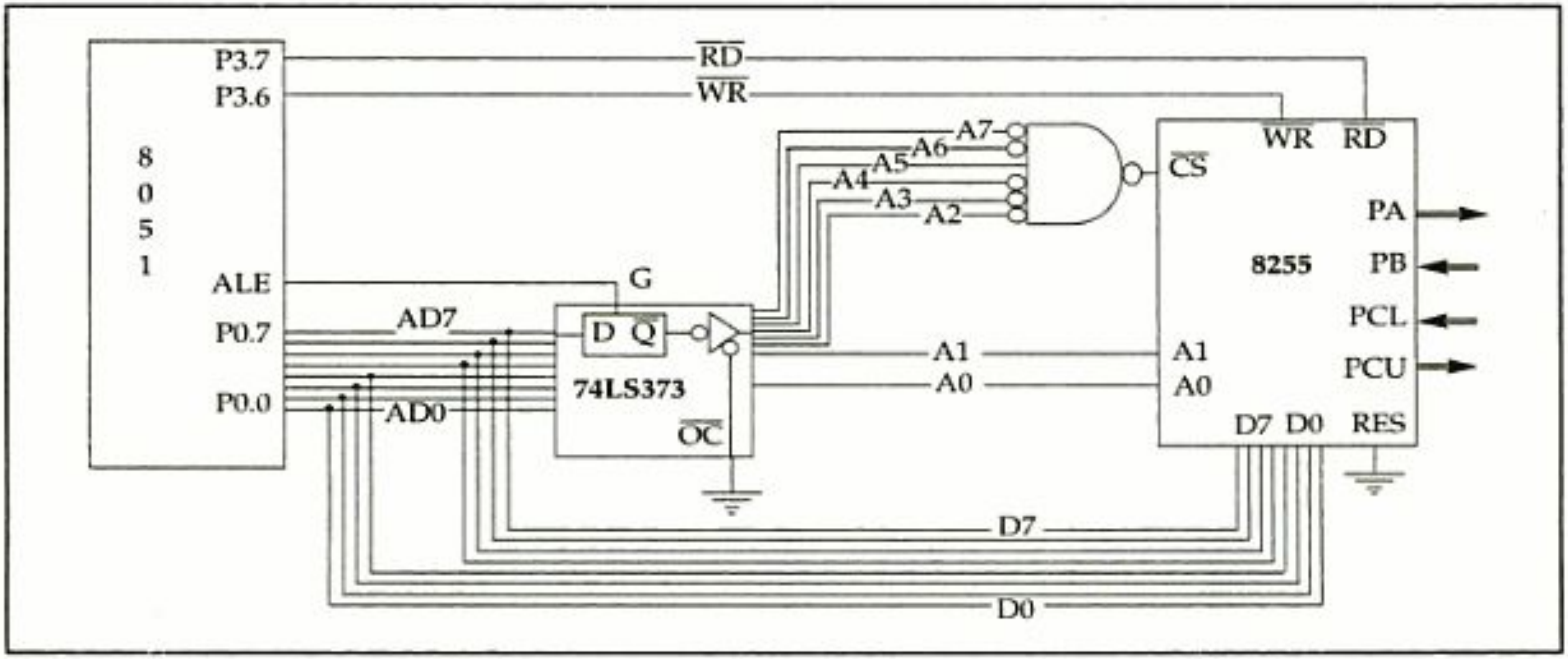
- I/O devices are interfaced using ports A, B, C

- If the peripheral is an i/p or o/p device, the port is defined as an i/p or o/p port.

- If the I/O device is of 8bit then ports A and B can be used, if they are of 4 bits then port C upper or lower can be used.

- **Example: Interface 4 push button switches to PC0-PC3 and display the key value by 4 LEDs connected to PC4-PC7. What will be the control word?**

<u>Problem</u>: A switch SW is connected to PC0. If SW=1 , data received from port A is to be transferred to port B. If SW=0, data received from port B is to be transferred to port A. WAP for this to be done continuously.



Solution:

- When SW=1, PA=in & PB=out & PCL=in. Control word=91H
- When SW=0, PA=out, PB=in &PCL=in. Control word =83H
- From fig. addresses for PA, PB, PC & control register are 20, 21, 22 & 23H resp.

```
CWD1      EQU 91H
CWD2      EQU 83H
PA    EQU     20H
PB    EQU     21H
PC    EQU     22H
CREG      EQU     23H
```

```
        MOV A,#CWD1
        MOV R0,#CREG
        MOVX @R0,A
RPT:    MOV R0,#PC
        MOVX A,@R0
        RRC A
        JC THERE
HERE:           MOV A,#CWD2
        MOV R0,#CREG
        MOVX @R0,A
        MOV R0,#PB
        MOVX A,@R0
        MOV R0,#PA
        MOVX @R0,A
        SJMP RPT
THERE:          MOV R0,#PA
        MOVX A,@R0
        MOV R0,#PB
        MOVX @R0,A
        SJMP RPT
        END
```

- **Interface two 8255 PPIs with the 8051 microcontroller such that port A of 8255(1) is selected for address 2000H and port A of 8255(2) is selected for address 4000H.**

	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
8255(1):	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	PA
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	PB
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	PC
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	CREG
8255(2):	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PA
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	PB
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	PC
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	CREG

A1 & A0 are directly connected to A1 & A0 of both8255(1) & 8255(2)

Y1 o/p line of decoder is connected to CS’ of 8255(1) & Y2 to CS’ of 8255(2)

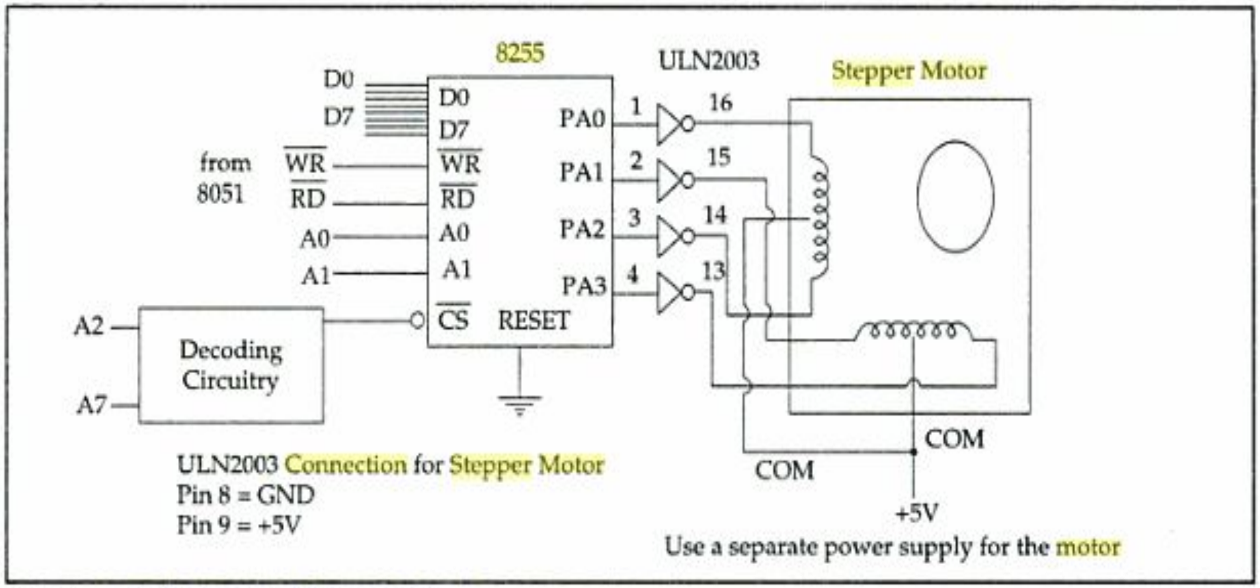


Figure 8255 Connection to Stepper Motor

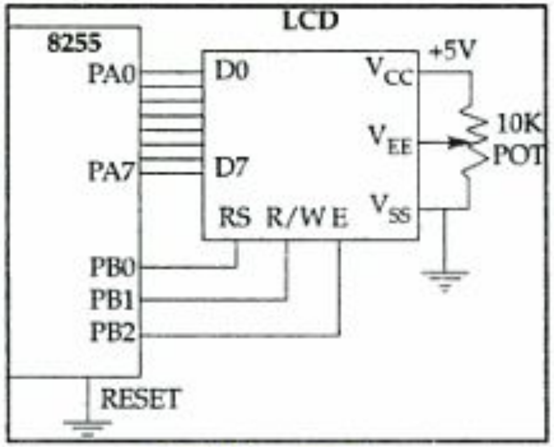


Figure LCD Connection

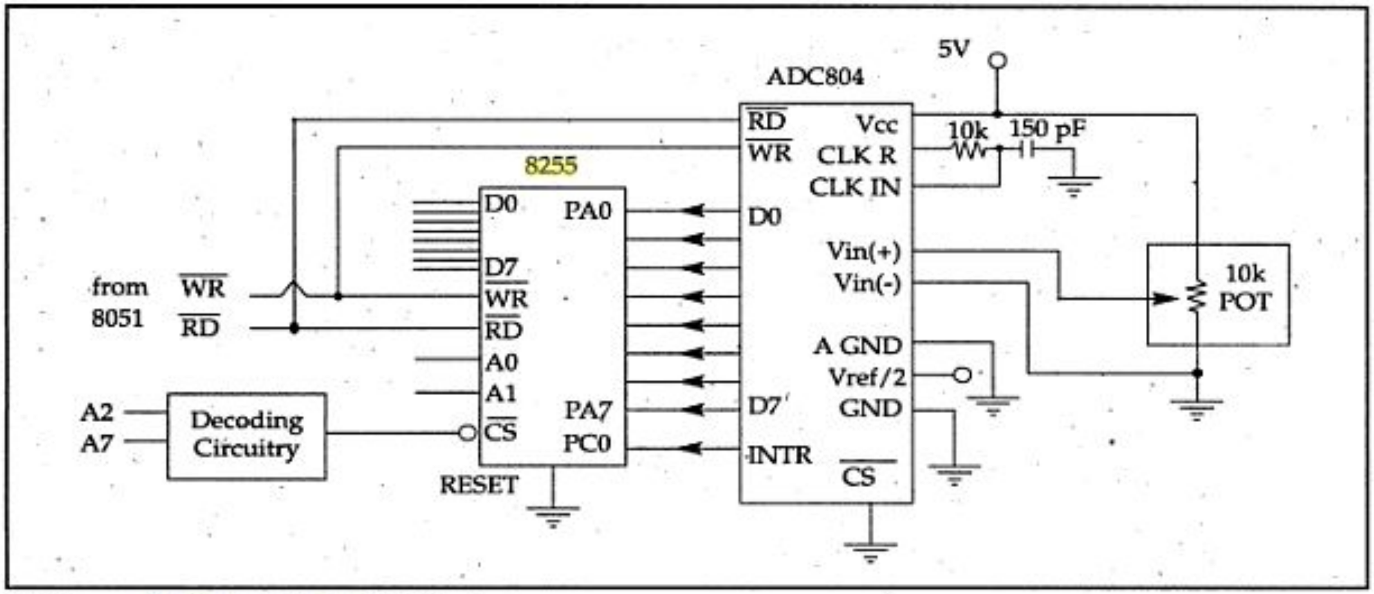


Figure 8255 Connection to ADC804

Questions

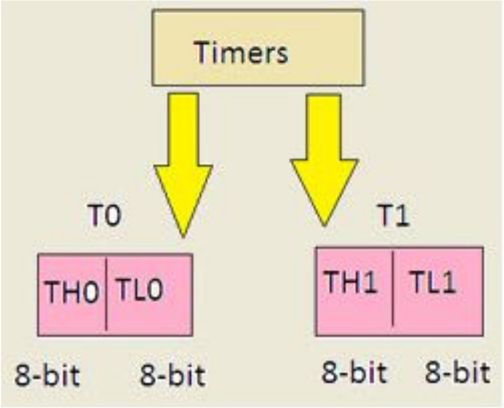
- How many timers/counters does the 8051 microcontroller have?
- What is the basic difference between a timer and a counter?
- What is the clock source for the 8051 timer?
- What is the clock source for the 8051 counter?
- Name some SFRs of the timer/counter.
- What are the 4 modes of 8051 timer?
- How do you start and stop the 8051 timer?
- What is the function of the C/T bit in the TMOD register?
- When is the Timer Flag (TF bit of the TCON register) set?

Timers/ Counters:

- 8051 has two 16-bit programmable UP timers/counters (T0 and T1), which can be configured to operate either as timers or as event counters.

- The timer content is available in four 8-bit special function registers, viz, TL0, TH0, TL1 and TH1 respectively.

- In the "timer" function mode, the counter is incremented in every machine cycle. Thus, one can think of it as counting machine cycles.



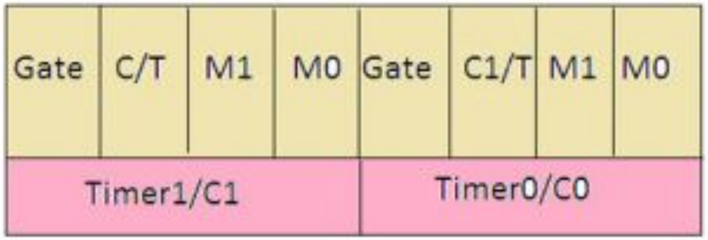
- One m/c cycle consists of 12 clock pulses. Hence the clock rate is 1/12 th of the oscillator frequency.

- In the "counter" function mode, the register is incremented in response to a 1 to 0 transition at its corresponding external input pin (T0 or T1).

- It requires 2 machine cycles to detect a high to low transition. Hence maximum count rate is 1/24 th of the oscillator frequency.

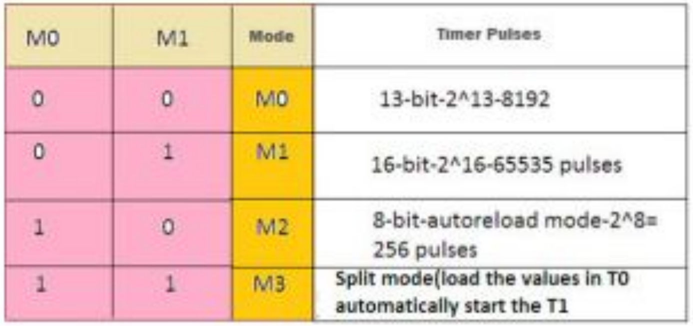
Timer Mode control (TMOD) Special Function Register:

- address of this SFR is 89H
- 8-bit register for selecting timer/counter and mode
- Lower 4-bits for controlling timer 0 and upper 4-bits for timer 1.
- **Gate bit:** set to one ('1') by the program to enable the interrupt to start/stop the timer. Timer/counter is enabled only while the INTX pin is high and the TRX control bit is set. When cleared, timer is enabled when TRX control bit is set.
- **C/T' bit:** selection of Timer or Counter
- **M0, M1 bits:** mode select bits (mode 0,1,2,3)
- The start and stop of timer is controlled by software by way of the TR bits (TR0 & TR1) –SETB TR0, CLR TR0
- The hardware way of starting and stopping the timer by an external source is achieved by making Gate=1 in the TMOD register.



Gate	C/T	M1	M0	Gate	C1/T	M1	M0
Timer1/C1				Timer0/C0			

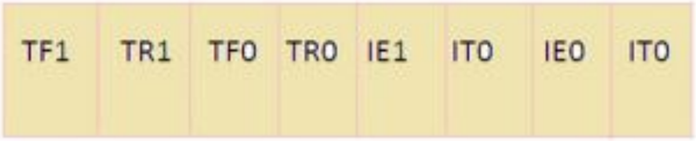
Timer Mode Control (TMOD)



M0	M1	Mode	Timer Pulses
0	0	M0	13-bit-2^13-8192
0	1	M1	16-bit-2^16-65535 pulses
1	0	M2	8-bit-autoreload mode-2^8= 256 pulses
1	1	M3	Split mode(load the values in T0 automatically start the T1

Timer control (TCON) Special Function Register:

- bit addressable, address is 88H
- **TF1 :** Timer1 overflow flag is set when timer rolls from all 1s to 0s, is cleared when processor vectors to execute ISR located at address 001BH.
- **TR1 :** Timer1 run control bit. Set to 1 to start the timer / counter.
- **TF0 :** Timer0 overflow flag. (Similar to TF1)
- **TR0 :** Timer0 run control bit.



TF1	TR1	TF0	TR0	IE1	IT0	IE0	IT0

Timer Control Register (TCON)

- **IE1 :** Interrupt1 edge flag. Set by h/w when an external interrupt edge is detected, cleared when interrupt is processed.
- **IE0 :** Interrupt0 edge flag. (Similar to IE1)
- **IT1 :** Interrupt1 type control bit. Set/cleared by s/w to specify falling edge/low level triggered external interrupt.
- **IT0 :** Interrupt0 type control bit. (Similar to IT1)

Timer Modes:

- **Timer Mode 0:** timer is used as a 13-bit UP counter. The lower 5 bits of TLX and 8 bits of THX are used for the 13 bit count. When the counter rolls over from all 1's to all 0's, TFX flag is set and an interrupt is generated.

- **Timer Mode 1:** similar to mode-0 except for the fact that the Timer operates in 16-bit mode.

- **Timer Mode 2 (Auto-Reload mode):** Counting is performed in TLX while THX stores a constant value. In this mode when the timer overflows i.e. TLX becomes FFH, it is fed with the value stored in THX.

- **Timer Mode 3:** Timer 1 in mode-3 simply holds its count. Timer 0 in mode-3 establishes TL0 and TH0 as two separate counters.

Mode 1 programming

- 16-bit timer, allows values 0000H to FFFFH to be loaded into TL & TH.

- After TL & TH are loaded with an initial value, the timer is started by using SETB TR0/TR1.

- The timer once started starts counting up till it reaches its maximum value of FFFFH.

- When it rolls over from FFFF to 0000, it sets high a flag bit called TF which can be monitored.

- When TF is set, one option is to stop the timer using CLR TR0/TR1.

- In order to repeat the process, TH & TL must be reloaded with the original value and TF must be reset to 0.

Steps to program in Mode 1

To generate a time delay using the timer in mode 1, the following steps are required:

1.Load TMOD indicating which timer is to be used in which mode.

2. Load TL & TH with initial count.

3. Start the timer.

4. Monitor TF with the ‘JNB TFx, target’ instruction. Get out of the loop when TF is high.

5. Stop the timer.

6. Clear TF for the next round.

7. Go back to step 2 to load TL & TH again.

Finding values to be loaded into the timer

- Assuming we know the amount of time delay we need, how to find the values required to be loaded into TL & TH.
- Assume XTAL=11.0592MHz, T=1.085us
- n=desired time delay/T
- Perform 65536-n
- yyxx=hexadecimal equivalent of n
- Set TL=xx and TH=yy

Analyze the following program:

```
        MOV  TMOD, #01              ;TIMER 0, MODE 1
HERE:       MOV  TL0, #0F2H         ;TL0=F2H
        MOV  TH0, #0FFH             ;TH0=FFH
        CPL P1.5            ;toggle P1.5
        ACALL DELAY             ;call DELAY subroutine
        SJMP HERE           ;reload TL and TH
;**********Delay using Timer 0**********
DELAY:
        SETB TR0            ;start the timer
AGAIN:      JNB TF0, AGAIN          ;monitor TF0 till it rolls over
        CLR TR0             ;stop timer 0
        CLR TF0             ;reset timer 0 overflow flag
        RET             ;return to main program
```

Timer counts up from FFF2H to FFFFH
FFF2(TF=0) ⍰ FFF3(TF=0)⍰FFF4(TF=0)⍰FFF5(TF=0) ⍰ ………… ⍰ FFFF(TF=0)⍰ 0000(TF=1)

Assuming XTAL=11.0592MHz, Timer works with 1/12th of XTAL,i.e., timer frequency=921.6kHz.

Each clock has a period T=1/921.6kHz=1.085us, No. of counts(FFF2 to FFFF and then to 0000)=14

Half of the pulse =14 x 1.085us=15.19us, Total delay generated by the timer=2 x 15.19us=30.38us

What value do we need to load into the timer's registers if we want to have a time delay of 5ms? Show the program for Timer 0 to create a pulse width of 5ms on P2.3. Assume XTAL=11.0592MHz.

- Since XTAL=11.0592MHz, timer counts up every 1.085us. (implies we need to make a 5ms pulse out of 1.085us intervals.)
- Therefore, we need 5ms/1.085us=4608 clocks.
- Value to be loaded into timer 0 registers=65536-4608=60928=EE00H

```
        CLR P2.3                ; clear P2.3
        MOV TMOD, # 01          ;Timer 0, mode 1
HERE:       MOV TL0, #0                 ;load TL0
        MOV TH0,#0EEH           ;load TH0
        SETB P2.3               ;set P2.3
        SETB TR0                ;start timer 0
AGAIN:      JNB TF0, AGAIN              ;monitor TF0
        CLR P2.3                ;clear P2.3
        CLR TR0                 ;stop timer 0
        CLR TF0                 ;clear TF0
        RET                     ;return to main
```

Try: Generate square waves with an ON time of 3ms and an OFF time of 10ms on all pins of port 0. Assume XTAL=22MHz. Use Timer 0 in Mode 1.

Maximum delay possible

- For XTAL=11.0592MHz
- T=Reciprocal of (1/12th of 11.0592MHz)=1.085us
- Maximum delay possible is when TL=TH=0
- Count= FFFF-0000+1=65536
- **Delay=65536 x 1.085us=71.12ms**

- For XTAL=22MHz
- T=Reciprocal of (1/12th of 22MHz)=0.546us
- Maximum delay possible is when TL=TH=0
- Count= FFFF-0000+1=65536
- **Delay=65536 x 0.546us=35.78ms**

WAP to generate a pulse train 2s period on P2.4. Use Timer 1 in mode 1. Assume XTAL=22MHz.

- For a time period of 2s, half a period will be 1s

- Maximum delay using Timer 1 in mode 1 with XTAL=22MHz=35.78ms

- However if this delay is repeated 28 times we get an approx delay of 1s.

- 1s/35.78ms=27.95=28(approx)

- Therefore we need to run the delay loop 28 times.

```
        MOV TMOD, #10H     ;timer 1 mode 1
REPT:       MOV R0,#28     ;counter for multiple loops
        CPL P2.4
BACK:       MOV TL1,#00H
        MOV TH1,#00H
        SETB TR1
AGAIN:      JNB TF1,AGAIN
        CLR TR1
        CLR TF1
        DJNZ R0, BACK
        SJMP REPT
```

Mode 2 Programming

- 8-bit timer which allows values 00H to FFH to be loaded into the timer’s TH register.

- After TH is loaded, a copy of it is given to TL and the timer is started.

- Timer starts counting by incrementing TL register and finally reaches maximum value of FFH when it rolls over to 00H.

- While doing so it sets TF1 or TF0.

- At the same time TL is reloaded automatically with the original value stored in TH.

- To repeat the process, simply clear TF1/TF0.

Steps to program in Mode 2

1. Load TMOD indicating which timer is to be used in which mode.

2. Load TH with initial count

3. Start timer

4. Monitor TF to see when it is high. When high, get out of the loop.

5. Clear TF

6. Go back to step 4 since mode-2 is auto-reload.

Assuming XTAL=11.0592MHz, find:
(a) frequency of square wave generated on P1.0 in the given program,
(b) smallest frequency achievable in this program and the TH value required.

```
        MOV TMOD, #20H
        MOV TH1,#5
        SETB TR1
BACK:     JNB TF1, BACK
        CPL P1.0
        CLR TF1
        SJMP BACK
```

Note:
count=255-5+1=251,

delay=251x1.085us=272.33us
This is half of the pulse (say the low part)

Therefore, total period(50% duty cycle)=2x272.33us=544.67us and frequency=1/544.67=1.83 kHz

To get smallest frequency, we need largest T and that is achieved when TH=00.
T=2x256x1.085us=555.5us and frequency=1.8kHz

Counter Programming

- When the Timer/Counter of the 8051 microcontroller is used as a timer, the 8051’s crystal is used as the source of the frequency.

- However when it is used as a counter, it is a pulse outside the 8051 that increments the TL and TH registers.

- To use it as a counter, the C/T bit in the TMOD register has to be set to 1.

- In this case the counter counts up as pulses are fed from pins 14 or P3.4 (T0, i.e., Timer 0 input) and 15 or P3.5 (T1, i.e., Timer 1 input).

Design a counter to count the pulses of an input signal fed at pin P3.4. Assume XTAL=22MHz

```
            ORG 0000H
RPT:        MOV TMOD,#15H
            SETB P3.4
            MOV TL0,#00H
            MOV TH0,#00H
            SETB TR0
            MOV R0,#28
AGAIN:          MOV TL1,#00H
            MOV TH1,#00H
            SETB TR1
BACK:           JNB TF1,BACK
            CLR TF1
            CLR TR1
            DJNZ R0,AGAIN
            MOV A, TL0
            MOV P2,A
            MOV A, TH0
            MOV P1,A
            SJMP RPT
            END
```

A switch SW is connected to P1.2. Write an 8051 C program to monitor SW and generate a signal with a frequency of 500Hz when SW=0 and a frequency of 750Hz when SW=1, on P1.7. [Use Timer 0 in Mode 1, Assume XTAL=11.0592MHz]

```
sbit mybit=P1^7;
sbit SW=P1^2;
void T0M1Delay(unsigned char);
void main(void)
{
SW=1;
while(1)
    {
    mybit=~mybit;
    if(SW==0)
        T0M1Delay(0);
    else
        T0M1Delay(1);
    }
}
```

```
void T0M1Delay(unsigned char c)
{
  TMOD=0x01;
  if(c==0)
  {
          TL0=0x67;
          TH0=0xFC;
  }
  else
  {
          TL0=0x9A;
          TH0=0xFD;
  }
  TR0=1;
  while(TF0==0);
  TR0=0;
  TF0=0;
}
```

Note: FC67H=64615, 65536-64615=921
921x1.085us=999.285us and 1/(2x999.285us)=500Hz

FD9A=64922, 65536-64922=614
614x1.085us=666.19us and 1/(2x666.19us)=750Hz

TRY

- Assume that a 60Hz external clock is being fed into pin T0(P3.4). Write a C/Assembly program for counter 0 in mode 2 (8-bit auto reload) to display the seconds and minutes on P1 and P2 respectively.

***For Timer 0, Mode 2, TMOD=06H**

```
void disp_time(unsigned char);
void main()
{
        unsigned char val;
        T0=1;
        TMOD=0x06;
        TH0=-60;
        while(1)
        {
                do
                {
                        TR0=1;
                        val=TL0;
                        disp_time(val);
                }
                while(TF0==0);
                TR0=0;
                TF0=0;
        }
}
```

```
void disp_time(unsigned char val)
{
        unsigned char sec, min;
        min=val/60;
        sec=val%60;
        P1=sec;
        P2=min;
}
```

Interrupts

- A single microcontroller can serve several devices either via polling or interrupts.
- **Polling:**
  - uc continuously monitors status of a device and when status condition is met provides service to that device.
  - It can provide service to several devices.
  - Cannot provide priority since it checks all devices in a round robin fashion.
  - Wastes a lot of time polling devices that do not require service from the uc.
- **Interrupt:**
  - Each device can get the attention of the uc based on the priority assigned to it.
  - When using interrupts, a uc can ignore a device request for service-masking.
  - Does not tie down the uc with the task of monitoring the devices.
  - For every interrupt there is an ISR, address of which is stored in IVT.

Steps in executing an Interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

- Finishes current instruction execution & saves address of next instruction on stack

- Saves current status of all interrupts internally (not on stack)

- Jumps to fixed memory location called IVT that holds addresses of ISRs

- Gets address of ISR from IVT and jumps to it. Executes ISR till last instruction RETI.

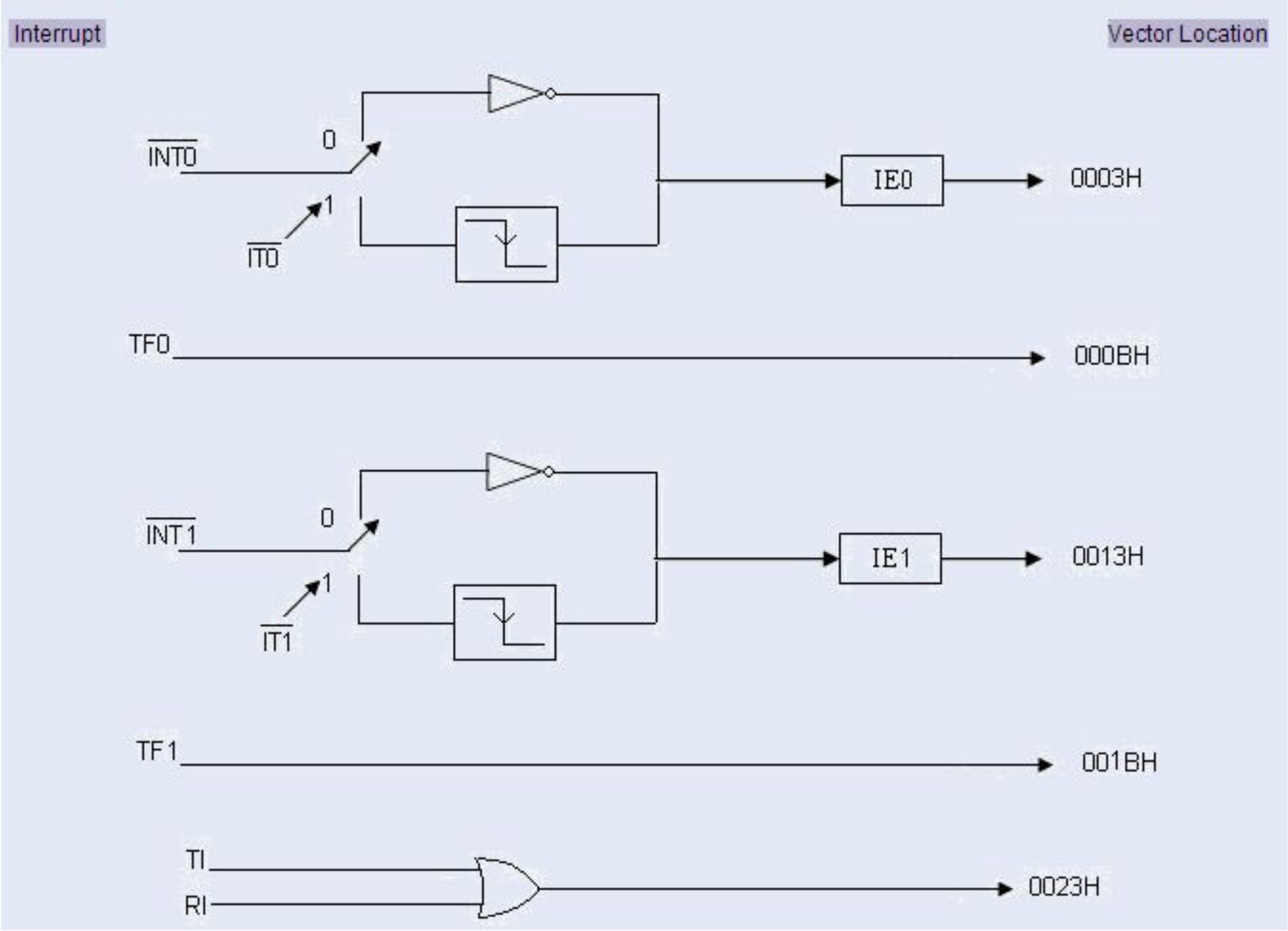
- Upon executing RETI, returns to the place where it was interrupted. This it does by retrieving the address from the stack.

The 6 interrupts of 8051

- **Reset:** when reset pin is activated 8051 jumps to address location 0000H.
- 2 interrupts for Timer0 and Timer 1, 000BH & 001BH in IVT.
- 2 interrupts (P3.2 & P3.3) for external hardware interrupts INT0 & INT1, 0003H & 0013H in IVT.
- Serial communication has a single interrupt for both receive & transmit, IVT location 0023H.
- If the ISR is larger than the allotted 8 bytes of memory space it is redirected to some other location using LJMP.

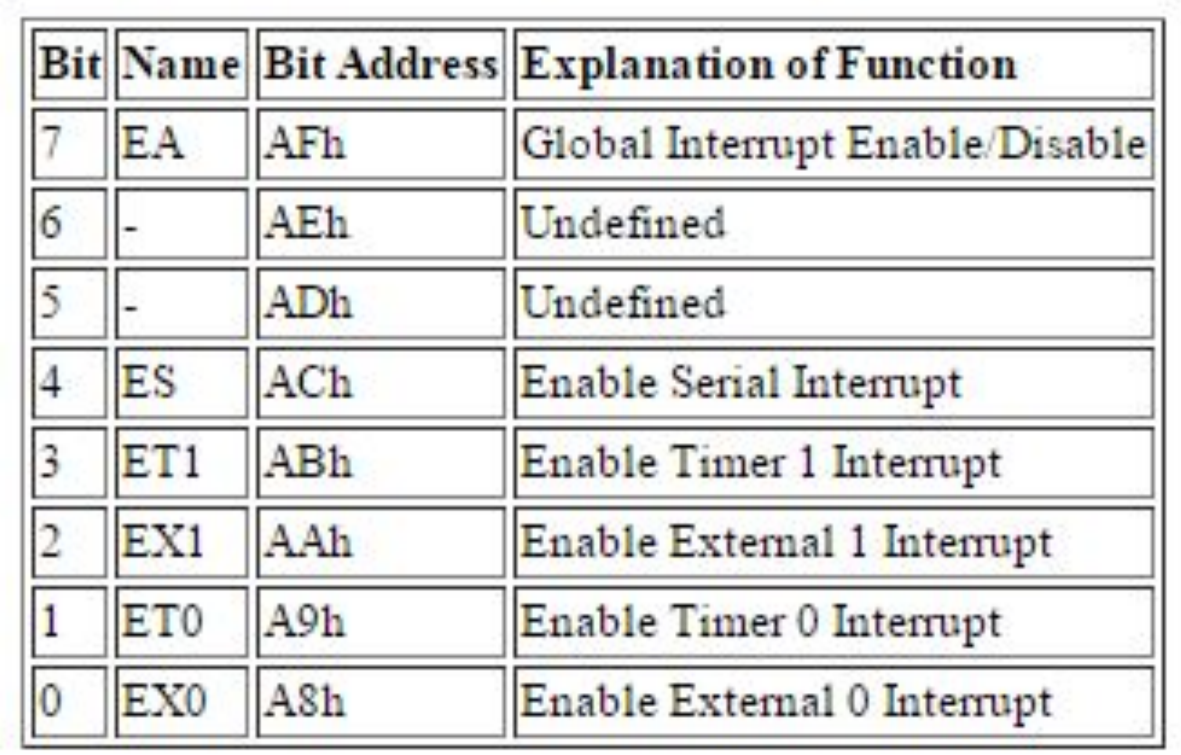
Interrupts:

- 8051 provides 5 vectored interrupts in addition to **RESET (0000H)**. They are:
- INT0,TF0, INT1, TF1, RI/TI (Each of these interrupts can be individually enabled or disabled by 'setting' or 'clearing' the corresponding bit in the IE SFR.
- 3 ROM locations for RESET. Put LJMP as the 1st instruction in our program to redirect processor away from IVT.



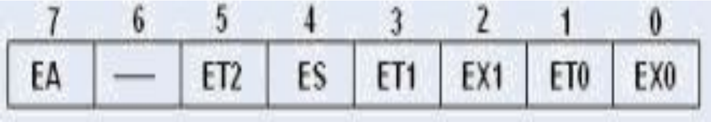
Enabling & Disabling of Interrupts

- All interrupts are disabled on reset
- Bit addressable register IE (address A8H) is responsible for enabling & disabling of interrupts.



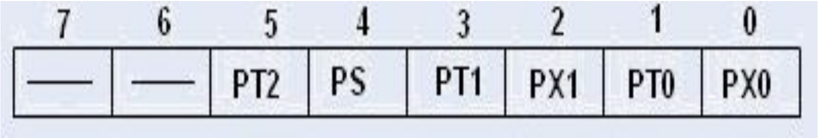
Bit	Name	Bit Address	Explanation of Function
7	EA	AFh	Global Interrupt Enable/Disable
6	-	AEh	Undefined
5	-	ADh	Undefined
4	ES	ACh	Enable Serial Interrupt
3	ET1	ABh	Enable Timer 1 Interrupt
2	EX1	AAh	Enable External 1 Interrupt
1	ET0	A9h	Enable Timer 0 Interrupt
0	EX0	A8h	Enable External 0 Interrupt

- Interrupt Enable Register (IE): address A8H



7	6	5	4	3	2	1	0
EA	—	ET2	ES	ET1	EX1	ET0	EX0

- Interrupt Priority Register (IP):



7	6	5	4	3	2	1	0
—	—	PT2	PS	PT1	PX1	PT0	PX0

Priority Level Structure: Each interrupt source can be programmed to have one of the two priority levels by setting (high priority) or clearing (low priority) a bit in the IP Register.

A low priority interrupt can itself be interrupted by a high priority interrupt, but not by another low priority interrupt.

If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served.

If the requests of the same priority level are received simultaneously, an internal polling sequence determines which request is to be serviced.

Thus, within each priority level, there is a second priority level determined by the polling sequence

Interrupt Handling:

- The interrupt flags are sampled at P2 of S5 of every instruction cycle (6 states with 2 pulses each).

- The samples are polled during the next machine cycle (or instruction cycle).

- If one of the flags was set at P2 of S5 of the preceding instruction cycle, the polling detects it and the interrupt process generates a long call (LCALL) to the appropriate vector location of the interrupt.

- The LCALL is generated provided this hardware generated LCALL is not blocked by any one of the following conditions:
  - An interrupt of equal or higher priority level is already in progress.
  - The current polling cycle is not the final cycle in the execution of the instruction in progress.
  - The instruction in progress is RETI or any write to IE or IP registers.

- **Reset** is a non-maskable interrupt. A reset is accomplished by holding the RST pin high for at least two machine cycles.

Programming Timer Interrupts

- TF is raised when the timer rolls over and needs to be monitored

```
SETB TR0          ;start timer
JNB TF0, AGAIN   ; check timer overflow
```

- If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised and the microcontroller is interrupted in whatever it is doing & jumps to the IVT to execute the ISR.

WAP to generate 2 square waves one of 5KHz on P1.3 & other of 25KHz on P2.3 (X-TAL=22MHz)

- For a freq of 5KHz,T=1/f=1/5KHz=0.2ms
- Half of it is high & half is low=0.1ms=100us
- For x-tal freq 22MHz, 1 m/c cycle=0.546us
- 100us/0.546us=183.15cycles
- 256-184=72=48H

- For a freq of 25KHz,T=1/f=1/25KHz=0.04ms
- Half of it is high & half is low=0.02ms=20us
- 20us/0.546us=36.63=37
- 256-37=219=DBH

```
ORG 0000H
LJMP MAIN
;ISR for Timer 0
ORG 000BH
CPL P1.3
RETI
;ISR for Timer 1
ORG 001BH
CPL P2.3
RETI
;Main program for initialization
ORG 0030H
MAIN:   MOV TMOD, #22H     ; both timers in mode 2 auto-reload
        MOV IE,#8AH        ; enable timer 0 & timer 1 interrupts
        MOV TH0,#48H
        MOV TH1,#0DBH
        SETB TR0
        SETB TR1
WAIT:   SJMP WAIT
        END
```

Programming External Hardware Interrupts

- External Interrupts INT0 (P3.2) and INT1 (P3.3). Enabled and disabled using register IE.

- IVT locations **0003H** and **0013H** are set aside for **INT0** and **INT1**.

- They can be activated in two ways:

  Level Triggered(default mode): INT0 & INT1 are normally high and if a low level signal is applied to them it triggers the interrupt.

  The low level signal at the INT pin has to be removed before execution of the last instruction of the ISR i.e., RETI, otherwise another interrupt will be generated.

  Edge Triggered: To make the interrupts edge triggered, the **TCON** register has to be programmed.

  IT0 & IT1, bits D0 & D2 of the TCON register, referred to as TCON.0 & TCON.2 should be made high for edge triggering. Upon reset they are both 0 indicating level triggering.

  example: SETB TCON.2 makes INT1 an edge triggered interrupt so that when a High-to-Low signal is applied to P3.3, microcontroller will be interrupted and forced to jump to location 0013H in the IVT to service the ISR.

Sampling the Interrupts

- After the hardware interrupts in the IE register are enabled, controller keeps sampling the INTn pin for a low level signal once every machine cycle.

- For the interrupt to be recognized the pin has to be kept low till the start of execution of the ISR.

- The pin has to be brought back to high before the execution of the RETI instruction.

- To ensure activation of the h/w interrupt at the INTn pin, the duration of the low level signal should be around 4 m/c cycles and no more.

- In edge triggered interrupts the external source must be held high for at least 1 m/c cycle and then held low for at least 1 m/c cycle to ensure recognition by microcontroller.

- The falling edge is latched and held by the TCON register.(TCON.1 or IE0 & TCON.3 or IE1 hold the latched falling edge of pins INT0 & INT1.)

- **Two switches are connected to P3.2 (INT0) and P3.3(INT1).When a switch is pressed, corresponding line goes low.**

- WAP to:
- Light all LEDs connected to Port 0, if the 1st switch is pressed
- Light all LEDs connected to Port 1, if the 2nd switch is pressed.

```
;Assume X-TAL=22MHz
;Pin 3.2 is for interrupt 0 & Pin3.3 is for interrupt 1
;upon wake-up go to main
      ORG 0000H
      LJMP MAIN
;ISR for INT0
      ORG 0003H
      MOV R0,#255
      LED1: MOV P0,#0FFH
        DJNZ R0,LED1
        RETI
;ISR for INT1
      ORG 0013H
      MOV R0,#255
      LED2: MOV P2,#0FFH
        DJNZ R0,LED2
        RETI
;main program for initialization
      ORG 0030H
      MAIN:MOV IE,#85H;enable INT0 & INT1
      HERE:  SJMP HERE
        END
```

TCON register

- IT0 (TCON.0)& IT1(TCON.2):sets the low-level or edge triggered mode for the external h/w interrupts INT0 & INT1.
- IE0(TCON.1) & IE1(TCON.3): they are used to latch the high-to-low edge transition on INT0 & INT1 pins.
- TR0(TCON.4) & TR1(TCON.6): are used to start or stop timers.
- TF0(TCON.5) & TF1(TCON.7):indicates roll-over of timers.

Serial Communication Interrupt

- TI is raised when the last bit of the framed data, i.e., the stop bit is sent indicating that the SBUF register is ready to transfer the next byte

- RI is raised when the entire frame including the stop bit is received.

- In the polling method we wait for the RI or TI flag to be raised whereas in the interrupt method we are notified when the 8051 has received a byte or when it has finished sending a byte.

- In the 8051 one interrupt is used for both receive and transmit.

- If the interrupt bit in the IE register is enabled, when RI or TI is raised 8051 gets interrupted and jumps to location 0023H to execute its ISR. In the ISR we need to check which flag has been raised.

- In serial communication, RI/TI flag must be cleared by programmer using s/w instruction whereas with external and timer interrupts it is the job of the 8051 to clear the interrupt flags.

WAP(in C) using interrupts which will do the following:

(a) receive data serially and send it to P0

(b)Read P1, transmit serially and give a copy to P2

(c) Make Timer 0 generate a square wave of 5KHz on P0.1

```
sbit SQWAVE= P0^1;
void Timer0()
{
SQWAVE=~SQWAVE;
}
void Serial0()
{
If(TI==1)
    {
     TI=0;
     }
Else
    {
    P0=SBUF;
    RI=0;
    }
}
```

```
void main()       //XTAL=11.0592Mhz
{
unsigned char x;
P1=0xFF;          //conf.as i/p port
TMOD=0x22;        //mode 2 for both timers
TH1=0xF6;         //baudrate=4800
SCON=0x50;        //8-bit data, 1 stop bit, REN enabled
TH0=0xA4;         //T=200us,delay=100us
IE=0x92;          //enable interrupts
TR1=1;            //start Timer 1
TR0=1;            //start Timer 0
while(1)
    {
    X=P1;
    SBUF=x;
    P2=x;
    }
}
```

Interrupt Priority

- Interrupts Priority upon reset:

Highest to Lowest Priority

External Interrupt 0 (INT0)
Timer Interrupt 0 (TF0)
External Interrupt 1 (INT1)
Timer Interrupt 1 (TF1)
Serial Communication (RI+TI)
Timer 2 (8052 only) (TF2)

***Note: interrupt priority can be set with the IP register**

- **IP.7, IP.6: reserved,**
- **IP.5:Timer 2 interrupt priority bit(8052 only),**
- **IP.4:Serial port interrupt priority bit,**
- **IP.3:Timer 1 interrupt priority bit,**
- **IP.2:Ext.interrupt 1 priority bit,**
- **IP.1: Timer 0 interrupt priority bit,**
- **IP.0:Ext.interrupt 0 priority bit**

Serial Communication

- Is used for transferring data between 2 systems located at distances of hundreds of feet to millions of miles apart.

- Serial data communication uses 2 methods, synchronous (block of data) and asynchronous(single byte of data).

- Programs for serial communication can be tedious. Special ICs such as UART, USART, etc are available by manufacturers.

8051 Serial Port Programming

- 8051 may transfer and receive data serially at different baud rates.

- The baud rate is programmable with the help of Timer 1 in mode-2 (auto-reload).

- 8051 divides the XTAL frequency by 12 to get the machine cycle frequency. 8051's serial communication UART circuitry divides this machine cycle frequency by 32 to be used by the Timer 1 to set the baud rate. (11.0592MHz/12/32=28800Hz)

- For a baud rate of 9600, the TH1 register has to be loaded with a value of '-3' in decimal or 'FD' in hex. [28800/3=9600, timer counts from FD to FF to 00]

- **SBUF:**
  - 8-bit register holds the byte of data to be transferred or receives the data byte.
- **SCON(serial control):**
  - 8-bit register to program the stop bit, start bit, data bits for data framing etc.
- **SM0, SM1 (0,1)**
  - Serial Mode-1, 8-bit data, 1 start bit and 1 stop bit.
- **SM2:**
  - Enables multiprocessing capability of 8051. We keep it 0.
- **REN/SCON.4 (receive enable):**
  - When high allows 8051 to receive data on the RxD pin.
- **TB8:**
  - It is used for serial modes 2 and 3. We make it 0.
- **RB8:**
  - In serial mode 1 it gets a copy of the stop bit when an 8-bit data is received. Rarely used. We make it 0.
- **TI(Receive Interrupt):**
  - When the 8051 finishes the transfer of the 8-bit character, it raises the TI flag to indicate that it is ready to transfer another byte. The TI bit is raised at the beginning of the stop bit.
- **RI(Transmit Interrupt):**
  - When 8051 receives data via RxD, it gets rid of the start & stop bits and places the byte in the SBUF register. Then it raises the RI flag to indicate that a byte has been received and should be picked up before it is lost. RI is raised halfway through the stop bit.

Steps for programming the 8051 to transfer/ receive data serially

1. TMOD register is loaded with the value 20H indicating the use of Timer 1 in mode 2, to set the baudrate.

2. TH1 is loaded with a suitable value to set the baudrate for serial data transfer. (FD for baudrate=9600)

3. SCON register is loaded with 50H, indicating serial mode 1, where 8-bit data is framed with 1 start bit and 1 stop bit.

4. TR1 is set to 1 to start Timer 1.

5. TI/RI is cleared (CLR TI/RI).

6. Character byte to be transferred serially is written into SBUF register. (Skip for receive operation)

7. TI/RI flag is monitored to check if transfer/reception is complete. For reception, when RI is raised SBUF has the byte. Its contents are saved.

8. To transfer/receive next character go to step 5.

Importance of TI and RI flags

- During transfer of byte in SBUF register, start bit is transferred followed by the 8 bits, one at a time and then the stop bit is transferred.

- During transfer of the stop bit, 8051 raises the TI flag indicating that the last character was transmitted and it is ready to transfer the next character.

- By monitoring the TI flag we make sure that the SBUF register is not overloaded.

- After SBUF is loaded with a new byte the TI flag should be forced to 0.

- Similarly when receiving a data byte, the bits are received bit by bit after the start bit and when the last bit is received the byte is placed in SBUF.

- While receiving the stop bit 8051 makes RI=1 indicating that the byte has been received and should be saved.

- After copying the contents of SBUF to a safe place, the RI flag is forced to 0. Failure to do so causes loss of received character.

Doubling the baud rate in the 8051

- 2 ways to increase the baud rate of 8051:
  - Use a higher frequency crystal. (Not feasible)
  - Change a bit (SMOD) in the in the PCON register. When 8051 is powered ON, SMOD bit is 0. It can be set to 1 by software to double the baud rate.

Write a C Program for the 8051 to transfer the letter ‘A’ serially at 4800 baud rate continuously. Use 8-bit data and 1 stop bit.

```
void main(void)
   {
   TMOD=0x20;       //use Timer 1 in mode-2
   TH1=0xFA;        //4800 baud rate
   SCON=0x50;       //01010000
   TR1=1;        //start timer
   while(1)
     {
     SBUF=‘A’;      //write letter A in SBUF
     while(TI==0);     //monitor TI flag
     TI=0;       //clear TI flag
     }
   }
```

Analog to Digital Converters

- 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 ADC to translate the analog signals to digital numbers, so microcontroller can read them.

ADC 0804

- ADC804 IC is a successive approximation analog-to-digital converter.

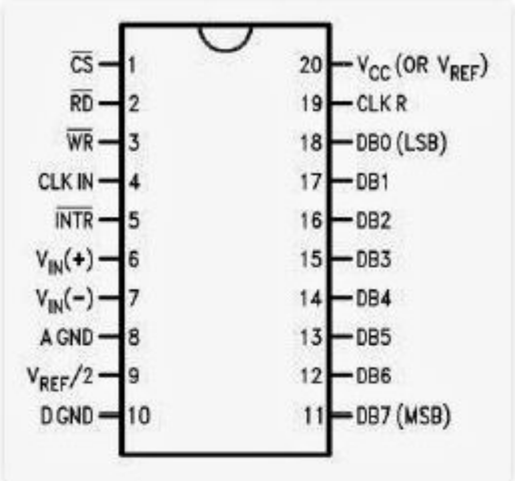
- It works with +5 volts and has a resolution of 8 bits

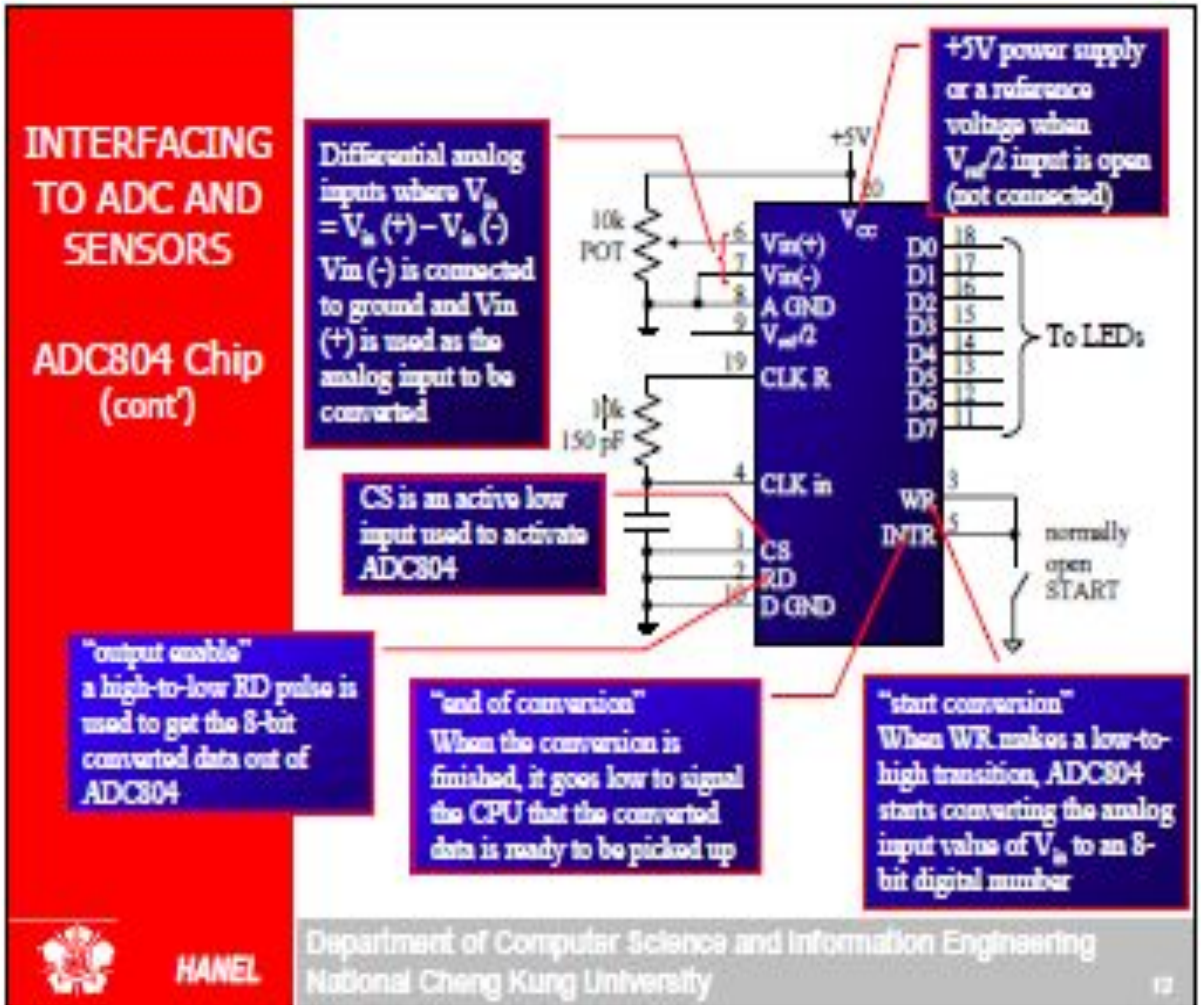
- A major factor in judging an ADC is Conversion time.

- Conversion time is defined as the time it takes for the ADC to convert the analog input to a digital (binary) number.

- In ADC0804 conversion time varies depending on the clocking signals applied to CLK R and CLK IN pins, but it cannot be faster than 110 µs.

Pin Description





INTERFACING TO ADC AND SENSORS

ADC804 Chip (cont')

+5V power supply or a reference voltage when $V_{ref}/2$ input is open (not connected)

Differential analog inputs where $V_{in} = V_{in}(+) - V_{in}(-)$ Vin (-) is connected to ground and Vin (+) is used as the analog input to be converted

CS is an active low input used to activate ADC804

"output enable" a high-to-low RD pulse is used to get the 8-bit converted data out of ADC804

"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

"start conversion" When WR makes a low-to-high transition, ADC804 starts converting the analog input value of V_{in} to an 8-bit digital number

Department of Computer Science and Information Engineering
National Cheng Kung University

CLK IN and CLK R

- CLK IN is an input pin connected to an external clock source
- 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=1/(1.1RC)
- Typical values are R = 10K ohms and C = 150 pF
- We get f = 606 kHz and the conversion time is 110 µs

Vref/2

- 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.

D0-D7

- Digital data output pins which are tri-state buffered
- The converted data is accessed only when CS =0 and RD is forced low
- To calculate the output voltage, use the following formula, Dout=Vin/stepsize
- Dout = digital data output (in decimal), Vin = analog voltage, and step size (resolution) is the smallest change

Analog ground and digital ground

- Analog ground is connected to the ground of the analog Vin
- Digital ground is connected to the ground of the Vcc pin
- 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

CS

- Active low input used to activate the ADC 0804 chip.
- To access the ADC0804 chip this pin must be kept low.

RD(Read)

- 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. When CS=0, if a high-to-low pulse is applied to the RD pin, the 8-bit digital output shows up at the D0-D7 data pins.
- The RD pin is also referred to as the Output Enable pin.

WR(start conversion)

- Active low input used to inform the ADC0804 to start the conversion process.
- If CS=0 when WR makes a low-to-high transition, the ADC0804 starts converting the analog input value of Vin to an 8-bit digital number.
- Amount of time it takes to convert varies depending on the CLK IN and CLK R values.
- When the data conversion is complete, the INTR pin is forced low by the ADC0804.

INTR

- An output and is active low
- It is normally high and when the conversion is complete, it goes low to signal the CPU that the converted data is ready to be picked up.
- After INTR goes low, we make the CS=0, and send a high-to-low pulse to the RD pin to get the data out of the ADC.

Vin(+) and Vin(-)

- Differential analog inputs where Vin=Vin(+)-Vin(-). Often Vin(-) is connected to ground and Vin(+) is used as the analog input to be converted to digital.

Vcc and Vref/2

- Vcc is the +5 supply voltage. It is used as the reference voltage when Vref/2 is kept open.
- Vref/2 is connected to (say) 2V if the Vref is to be kept at 4V.

Steps for conversion

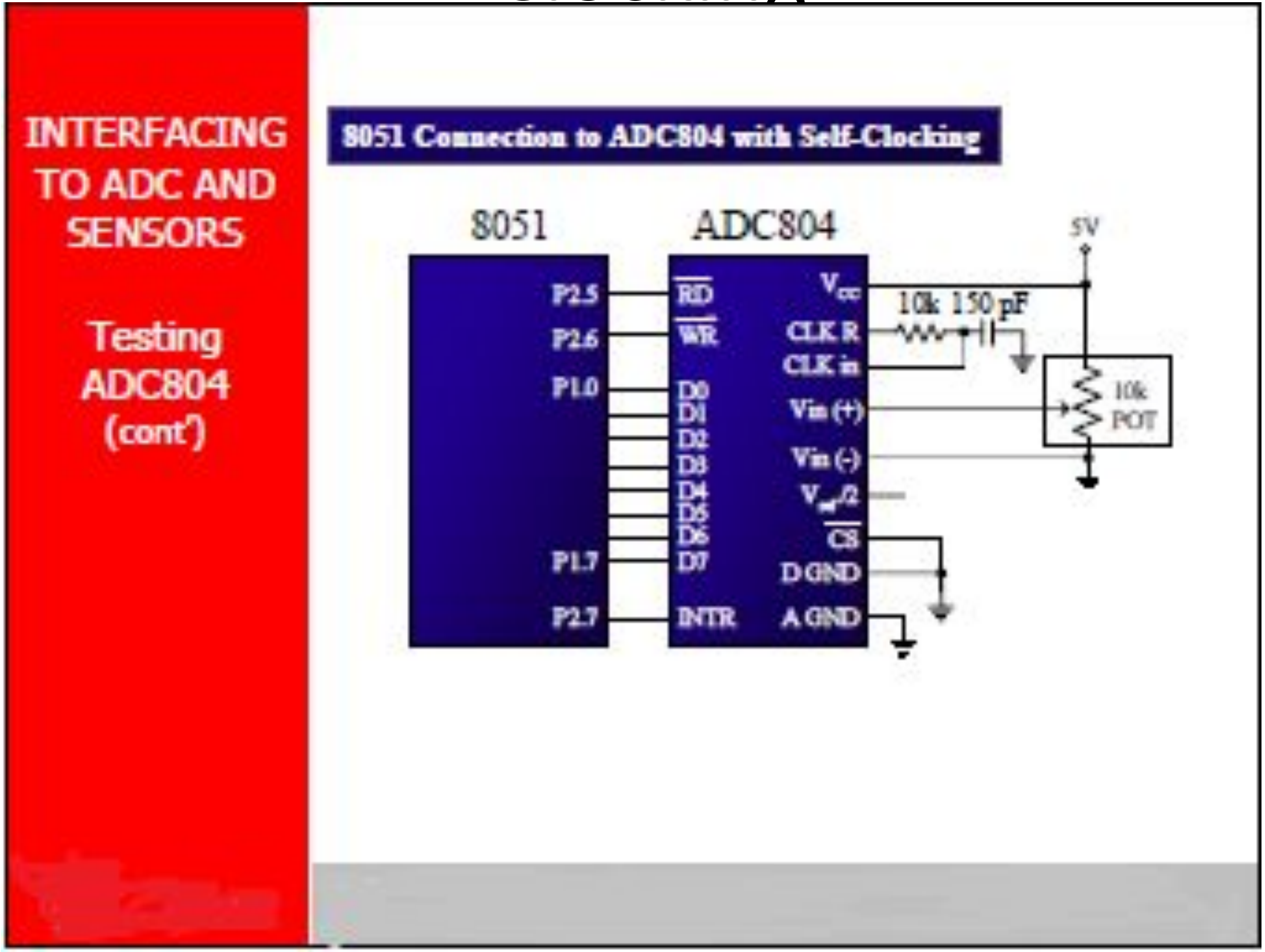
- The following steps must be followed for data conversion by the ADC804 chip:
  - Make CS = 0, send a low-to-high pulse to pin WR to start conversion
  - Keep monitoring the INTR pin
  - If INTR is low, the conversion is finished
  - If the INTR is high, keep polling until it goes low
  - After the INTR has become low, make CS = 0 and send a high-to-low pulse to the RD pin to get the data out of the ADC804.

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 =INTR (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
SETB P2.5
SETB P2.7
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
```

```
void main()
{
unsigned char value;
P1=0xFF;
P2.7=1;
P2.5=1;
P2.6=1;
while(1)
{
P2.6=0;
P2.6=1
while(P2.7==1);
P2.5=0;
value=P1;
Convert_Display(value);
P2.5=1;
}
}
```

8051 Connection to ADC0804 with Self Clocking

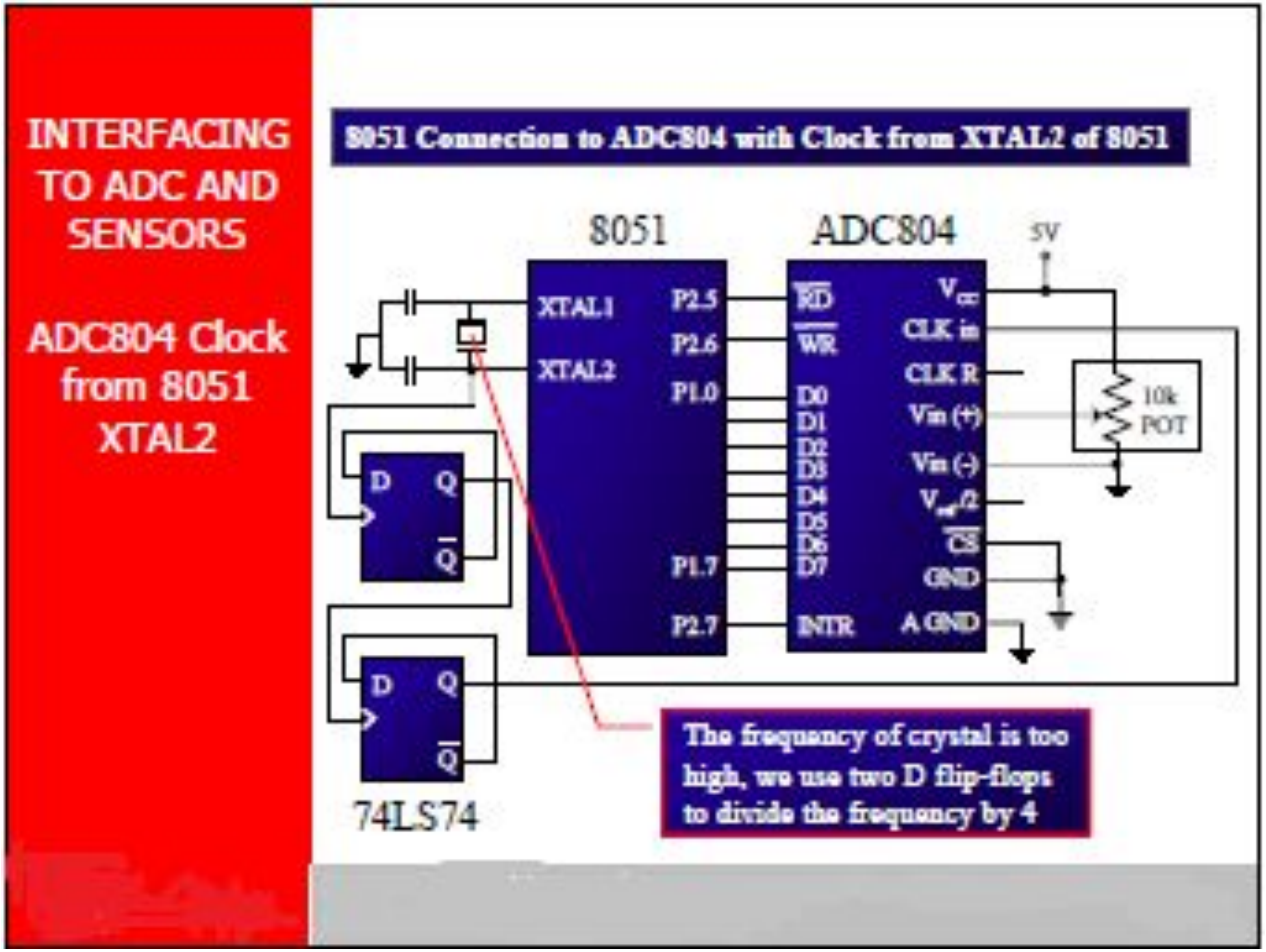


INTERFACING TO ADC AND SENSORS

Testing ADC804 (cont')

8051 Connection to ADC804 with Self-Clocking

8051 Connection to ADC0804 with Clock from XTAL osc of 8051



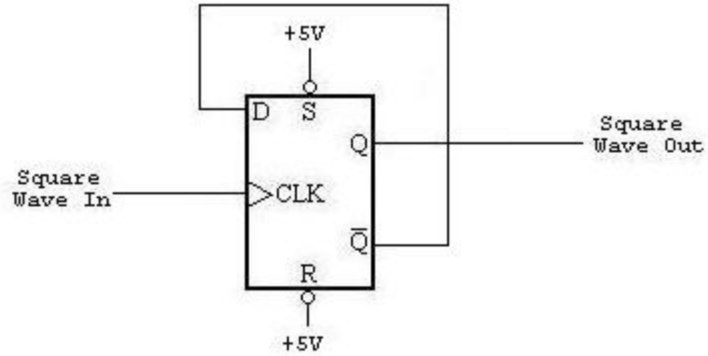
INTERFACING TO ADC AND SENSORS

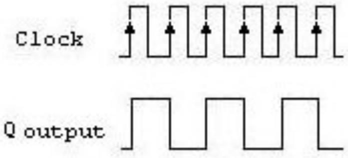
ADC804 Clock from 8051 XTAL2

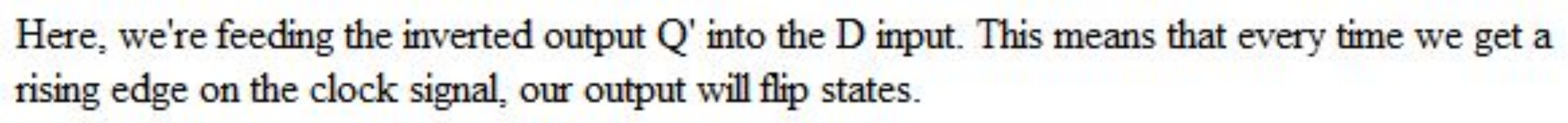
8051 Connection to ADC804 with Clock from XTAL2 of 8051

The frequency of crystal is too high, we use two D flip-flops to divide the frequency by 4

D Flip Flop as a Divide-by-2 counter






Here, we're feeding the inverted output Q' into the D input. This means that every time we get a rising edge on the clock signal, our output will flip states.

Serial Programming