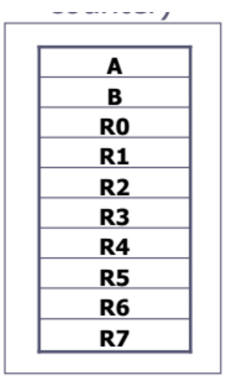
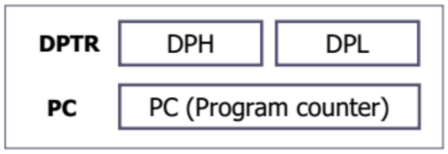
Programming the MCU 8051

1. Circuit to access internal RAM and code memory
2. Program the Ports
3. Program the Timers
4. Program the serial interface
5. Program the interrupts
6. Circuit to access external RAM and code memory

a. The most widely used registers

- A (Accumulator) - For all arithmetic and logic instructions –
- B, R0, R1, R2, R3, R4, R5, R6, R7
- DPTR (data pointer), and PC (program counter)
- Program status word register (PSW)





b. Program status word register (PSW)

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	OV	-	P

CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary carry flag.
F0	PSW.5	Available to the user for general purpose
RS1	PSW.4	Register Bank selector bit 1.
RS0	PSW.3	Register Bank selector bit 0.
OV	PSW.2	Overflow flag.
--	PSW.1	User definable bit.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of 1 bits in the accumulator.

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

c. Register banks and Switching between them

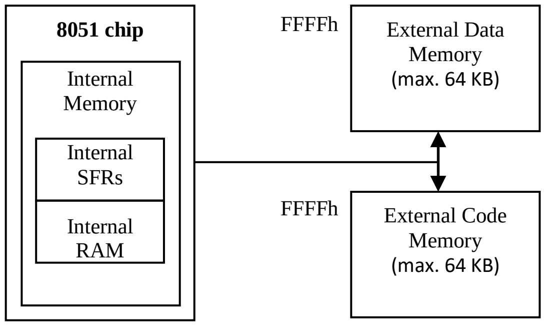
There are 4 banks of registers R0 – R7 at memory addresses 00 – 1Fh.
Switching is done through Register Bank selector bits PSW.3 & PSW.4 of *Program status word register* (PSW)

- Use the bit-addressable instructions SETB and CLR to access PSW.3 and PSW.4

Example:
At power up the register bank 0 is default and bits PSW.3 and PSW.4 are 0.
SETB PSW.4 ;sets RS1=1 and RS0=0 to select bank 2

d. Types of 8051 memory

1. On-chip memory – contained in the MCU,
2. External memory – can be connected to MCU using interface circuits.



8051 chip
Internal Memory
Internal SFRs
Internal RAM
FFFFh
External Data Memory (max. 64 KB)
FFFFh
External Code Memory (max. 64 KB)

Internal Memory

The on-chip memory of 8051 consists of 256 bytes of memory:

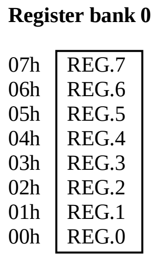
First 128 bytes:	00h to 1Fh	Register Banks
	20h to 2Fh	Bit addressable RAM
	30h to 7Fh	General purpose RAM
Next 128 bytes:	80h to FFh	Special function

Registers

The first 128 bytes is also known as internal RAM (IRAM)

Memory Space for Internal RAM

Address	RAM designation
7F-30	Scratch pad RAM
2F-20	Bit-Addressable RAM
1F-18	Registers (7..0) Bank 3
17-10	Registers (7..0) Bank 2
0F-08	Registers (7..0) Bank 1 (stack)
07-00	Registers (7..0) Bank 0



Addressing modes

The CPU can access data in various ways, which are called addressing modes

1. Immediate
2. Register
3. Register direct
4. Register indirect
5. Indexed

Immediate addressing mode

The source operand is a constant.

- The immediate data must be preceded by the pound sign, "#"

Examples:

```
MOV A, #25H          ;load 25H into A
MOV P1,#55H          ;send data to port # 1

MOV DPTR, #4521H     ;DPTR=4512H
```

- DPTR can also be accessed as two 8-bit registers, the high byte DPH and low byte DPL

```
MOV DPL, #21H        ;This is the same
MOV DPH, #45H        ;as above
```

- Use EQU directive to access immediate data:

```
Count  EQU  1EH
MOV R4, #COUNT          ;R4=1EH
MOV DPTR, #MYDATA       ;DPTR=200H
ORG 200H
MYDATA: DB "America"
```

Register addressing mode

Data transfer between registers.

Examples:

- Usually the register bank locations are accessed by the register names

```
MOV A, R4    ;copy contents of  R4 into A
MOV R2, A    ;copy contents of A into R2
MOV R7, DPL
```

- The movement of data between Rn registers is not allowed

```
MOV R4,R7        ;is invalid
```

- The source and destination registers must match in size.

```
MOV DPTR,A       ;is invalid
```

Direct addressing mode
Data transfer between register and memory by its address .
There is no “#” sign in the operand.
The entire 128 bytes of RAM can be accessed.

Examples:
```
MOV A, 04     ; copy contents of  R4 into A
```

- Usually the direct addressing mode is used to access RAM locations 30 – 7FH

```
MOV R0, 40H         ;save content of 40H in R0
MOV 56H, A          ;save content of A in 56H
```

- The Special Function Register (SFR) can be accessed by their names or by their addresses The SFR registers have addresses between 80H and FFH

```
MOV 0E0H, #55H      ;is the same as
MOV A, #55h         ;load 55H into A
MOV 0F0H, R0        ;is the same as
MOV B, R0           ;copy R0 into B
```

Special Function Register (SFR)

Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B register	0F0H
PSW*	Program status word	0D0H
SP	Stack pointer	81H
DPTR	Data pointer	2 bytes
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port0	80H
P1*	Port1	90H
P2*	Port2	0A0H
P3*	Port3	0B0H
IP*	Interrupt priority control	0B8H
IE*	Interrupt enable control	0A8H
TMOD	Timer/counter mode control	89H
TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
PCON	Power control	87H
SBUF	Serial data buffer	99H
SCON*	Serial control	98H
RCAP2L T/C 2	capture register low byte	0CAH
RCAP2H T/C 2	capture register high byte	0CBH
TL2	Timer/counter 2 low byte	0CCH
TH2	Timer/counter 2 high byte	0CDH
TL1	Timer/counter 1 low byte	8BH

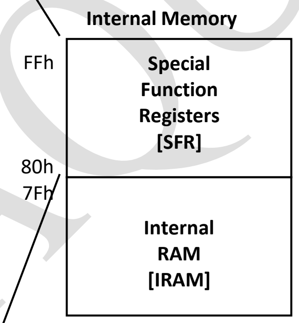
TH1	Timer/counter 1 high byte	8DH
TL0	Timer/counter 0 low byte	8AH
TH0	Timer/counter 0 high byte	8CH
T2MOD	Timer/counter mode control	0C9H

* Bit addressable

Memory addresses of SFR register

Byte address	Bit address	SFR Names
	b7 b6 b5 b4 b3 b2 b1 b0	
FFh		
F0h	F7 F6 F5 F4 F3 F2 F1 F0	B*
E0h	E7 E6 E5 E4 E3 E2 E1 E0	A* (accumulator)
D0h	D7 D6 D5 D4 D3 D2 D1 D0	PSW*
B8h	-- -- -- BC BB BB B9 B8	IP*
B0h	B7 B6 B5 B4 B3 B2 B1 B0	Port 3 (P3*)
A8h	AF -- -- AC AB AA A9 A8	IE*
A0h	A7 A6 A5 A4 A3 A2 A1 A0	Port 2 (P2*)
99h		SBUF
98h	9F 9E 9D 9C 9B 9A 99 98	SCON*
90h	97 96 95 94 93 92 91 90	Port 1 (P1*)
8Dh		TH1
8Ch		TH0
8Bh		TL1
8Ah		TL0
89h		TMOD
88h	8F 8E 8D 8C 8B 8A 89 88	TCON*
87h		PCON
83h		DPH
82h		DPL
81h		SP
80h	87 86 85 84 83 82 81 80	Port 0 (P0*)

* indicates the bit addressable SFR registers



- Only direct addressing mode is allowed for *pushing* or *popping* the stack

```
PUSH 0E0H   ;Pushing the accumulator onto the stack
PUSH A      ;invalid

PUSH 05H    ;push R5 onto stack
POP 02H     ;pop top of stack into R2
```

Register indirect addressing mode

A register is used as a pointer to the data.

a) To access the internal RAM only registers R0 and R1 are used

- When R0 and R1 hold the addresses of RAM locations, they must be preceded by the "@" sign

Examples:

```
MOV A,@R0        ;move contents of RAM whose address is held by R0 into A
MOV @R1,B        ;move contents of B into RAM ;whose address is held by R1
```

b) To accessing *externally* connected RAM or *on-chip* code ROM, the DPTR register is used as 16-bit pointer

Indexed addressing mode

It is used in accessing data elements of look-up table entries located in the on-chip code ROM
The instruction:
MOVC A, @A+DPTR

- The contents of A are added to the 16-bit register DPTR to form the 16-bit address of the needed data
- "C" means code

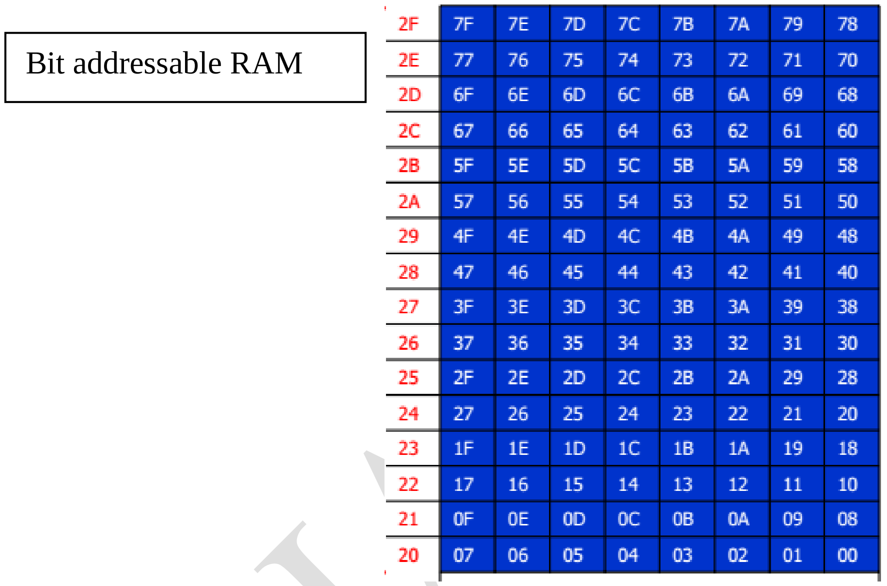
Examples:
Write a program to get the x value from P1 and send x2 to P2 continuously

```
ORG 0
MOV DPTR, #300H   ;LOAD TABLE ADDRESS
MOV A, #0FFH      ;A=FF
MOV P1, A         ;CONFIGURE P1 INPUT PORT
BACK: MOV A,P1    ;GET X
MOV A, @A+DPTR    ;GET X SQAURE FROM TABLE
MOV P2, A         ;ISSUE IT TO P2
SJMP BACK         ;KEEP DOING IT
ORG 300H
XSQR_TABLE:
DB 0,1,4,9,16,25,36,49,64,81
END
```

Single-bit operation

Single-bit instructions allow to set, clear, move, and complement individual bits of a port, memory, or register.
Bit address 00-7FH belong to RAM byte addresses 20-2FH
Bit address 80-F7H belong to SFR P0, P1, …

- Internal RAM locations 20-2FH are both byte-addressable and bit addressable



Bit addressable RAM

2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00

- The 128 bytes of RAM have the byte addresses of 00 – 7FH can be accessed in byte size using direct and register-indirect addressing modes.
- The 16 bytes of RAM locations 20 – 2FH have bit address of 00 – 7FH can be accessed by the single-bit instructions only which use direct addressing mode only.
- SFR registers A, B, PSW, IP, IE, ACC, SCON, TCON and all I/O ports are bit-addressable
- Code ROM, holding program for execution, is not bit-addressable

Instructions for single-bit operations

Instructions	Function
SETB bit	Set the bit (bit = 1)
CLR bit	Clear the bit (bit = 0)
CPL bit	Complement the bit (bit = NOT bit)
JB bit, target	Jump to target if bit = 1 (jump if bit)
JNB bit, target	Jump to target if bit = 0 (jump if no bit)
JBC bit, target	Jump to target if bit = 1, clear bit (jump if bit, then clear)

- The BIT *directive* is a widely used directive to assign the bit-addressable I/O and RAM locations
  Example: A switch is connected to pin P1.7 and an LED to pin P2.0. Write a program to get the status of the switch and send it to the LED.

Solution:

```
LED BIT P1.7          ;assign bit
SW BIT P2.0           ;assign bit
HERE: MOV C, SW       ;get the bit from the port
MOV LED, C            ;send the bit to the port
SJMP HERE             ;repeat forever
```

- Use the EQU *directive* to assign addresses
  Defined by names, like P1.7 or P2
  Defined by addresses, like 97H or 0A0H
  Example :
  A switch is connected to pin P1.7. Write a program to check the status of the switch and make the following decision.
  (a) If SW = 0, send "0" to P2
  (b) If SW = 1, send "1" to P2
  Solution:

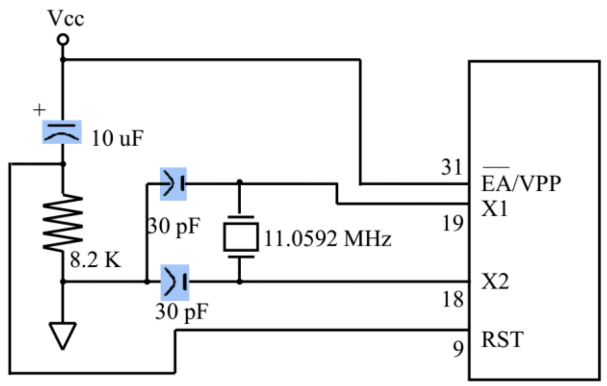
```
SW          EQU   P1.7
MYDATA      EQU   P2
HERE: MOV C,SW
        JC OVER
        MOV MYDATA,#'0'
        SJMP HERE
OVER: MOV MYDATA, #'1'
        SJMP HERE
END
```

The enhanced MCU 8052
It has another 128 bytes of on-chip RAM with addresses 80 – FFH often called upper memory.
Use *indirect addressing mode,* which uses R0 and R1 registers as pointers with values of 80H or higher to access Ram locations
– MOV @R0, A and MOV @R1, A

The SFRs are also assigned the same address space with addresses 80 – FFH .
Use *direct addressing mode* to access them
– MOV 90H, #55H or MOV P1, #55H

1) Circuit to access internal RAM and code memory

Pin 30	**/EA:** External Access - pin is pulled high by connecting it to Vcc.
Pins 18 and 19	**Crystal:** Connected to Crystal to provide system clock.
Pin 9	**RESET:** made high for small amount of time at power up to set 8051 to its initial values.



Power-on reset
RESET pin is active high .
Upon applying a high pulse to this pin, the microcontroller will reset and terminate all activitiesƒ
Activating a power-on reset will cause the following values in the registers.

Register	Reset Value
B	00
P0-P3	FF
SP	07
PSW	00
ACC	00
DPTR	0000
PC	0000

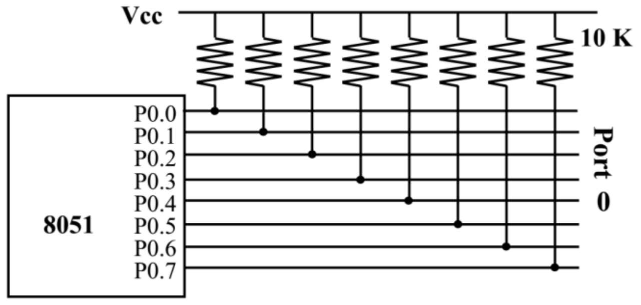
2) Program the Ports

There are four 8-bit I/O ports P_0, P_1, P_2 and P_3.

Hardware features:

Port 0 –

a) P0 is an open drain, each pin must be connected externally to pull-up resistors.



b) If 8051is connected with external memory Port 0 is designated as AD0-AD7, allowing it to be used for both address and data.

Port 1 –

It can be used for input or output port.

Port 2 –

a) It can be used for input or output port.
b) Port 2 is also designated as A8 – A15, to provide upper 8 bit address for the external memory

Port 3 –

a) It can be used for input or output port.
b) It has the additional function of providing important signals

P3 bit	Function	Pin	Function
P3.0	RxD	10	Serial communication
P3.1	TxD	11	
P3.2	/INT0	12	External interrupts
P3.3	/INT1	13	
P3.4	T0	14	Timers
P3.5	T1	15	
P3.6	/WR	16	Write / Read signals of external memories
P3.7	/RD	17	

Software features:

- All the ports upon RESET are configured as input ports
- Configuration of ports
  a) Output port - write 0 to the port,
  b) Input port - write 1 to the port,

Example:

1) Access all bits of port

- Send out continuously to port 0 values 55H and AAH alternating

```
;;The entire 8 bits of Port 0 are accessed
BACK: MOV A,#55H
      MOV P0, A
      ACALL DELAY
      MOV A, #0AAH
      MOV P0, A
      ACALL DELAY
      SJMP BACK
```

- Receive data from Port 0 and sent to P1

```
;; Make Port 0 as an input port by writing 1s to it, and then get data from P0 and send it to port 1

MOV A, #0FFH
MOV P0, A
BACK: MOV A, P0
      MOV P1,A
      SJMP BACK ; keep doing it
```

2) access only 1 or 2 bits of the port

```
BACK: CPL P1.2 ;complement P1.2
ACALL DELAY
SJMP BACK
```

Another variation of the above program

```
AGAIN: SETB P1.2        ;set only P1.2
ACALL DELAY
CLR P1.2                ;clear only P1.2
ACALL DELAY
SJMP AGAIN
```

P0	P1	P2	P3	Port Bit
P0.0	P1.0	P2.0	P3.0	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

Delay subroutine:

```
DELAY:  MOV R2, #200
AGAIN:  MOV R3, #250
HERE:   NOP
        NOP
        DJNZ R3, HERE
        DJNZ R2, AGAIN
RET
```

Larger delay subroutine:

```
DELAY:      MOV R1, #32
DEL1:       MOV R2, #32
DEL2:       MOV R3, #15
DEL3:       DJNZ R3, DEL3
            DJNZ R2, DEL2
            DJNZ R1, DEL1
RET
```

Note: If the crystal frequency is 11.0592MHz, then the delay is 0.217 ms.

3) Instructions for reading an input port

Mnemonic	Examples	Description
MOV A, PX	MOV A, P2	Bring into A the data at P2 pins
JNB PX.Y, ..	JNB P2.1, TARGET	Jump if pin P2.1 is low
JB PX.Y, ..	JB P1.3, TARGET	Jump if pin P1.3 is high
MOV C, PX.Y	MOV C, P2.4	Copy status of pin P2.4 to CY

In

Example
A switch is connected to pin P1.7. Write a program to check the status of SW and perform the following:
(a) If SW=0, send letter 'N' to P2
(b) If SW=1, send letter 'Y' to P2
Solution:

```
SETB P1.7                   ; make P1.7 an input
AGAIN:  JB P1.2,OVER        ; jump if P1.7=1
        MOV P2,#'N'         ; SW=0, issue 'N' to P2
        SJMP AGAIN          ; keep monitoring
OVER:   MOV P2,#'Y'         ; SW=1, issue 'Y' to P2
        SJMP AGAIN          ; keep monitoring
```

Example
A switch is connected to pin P1.0 and an LED to pin P2.7. Write a program to get the status of the switch and send it to the LED
Solution:

```
        SETB P1.7    ; make P1.7 an input
AGAIN:  MOV C, P1.0  ; read SW status into CF
        MOV P2.7, C  ; send SW status to LED
        SJMP AGAIN ; keep repeating
```

4) Possibilities of reading a port
a) Read the status of the input pin
b) Read the internal latch of the output port

Example: Instruction ANL P1,A
the sequence of actions is executed as follow
1. reads the internal latch of the port and brings that data into the CPU
2. This data is processed with the contents of register A
3. The result is rewritten back to the port latch
4. The port pin data is changed and now has the same value as port latch

Read-Modify-Write instructions
- read the port latch, normally read a value,
- perform an operation
- then rewrite it back to the port latch

Mnemonics	Example
ANL PX	ANL P1,A
ORL PX	ORL P2,A
XRL PX	XRL P0,A
JBC PX.Y,TARGET	JBC P1.1,TARGET
CPL PX.Y	CPL P1.2
INC PX	INC P1
DEC PX	DEC P2
DJNZ PX.Y,TARGET	DJNZ P1,TARGET
MOV PX.Y,C	MOV P1.2,C
CLR PX.Y	CLR P2.3
SETB PX.Y	SETB P2.3
Note: x is 0, 1, 2, or 3 for P0 – P3	

The ports in 8051 can be accessed by the Read-modify-write technique
1) Reading the port
2) Modifying it
3) Writing to the port

Example:

```
            MOV P1,#55H        ;P1=01010101
AGAIN:      XRL P1,#0FFH       ;XOR P1 with 1111 1111
            ACALL DELAY
            SJMP AGAIN
```

3. Program the Timers

The 8051 has two timers/counters used as

a) Timers: to generate a time delay or

b) Event counters: to count events

Steps of timer programming

1. Configure the timers using Mode Register (TMOD)

2. Select the count number using Timer registers (TH & TL)

3. Run / stop the timers using TR and TF bits of TCON register

1. Timer Mode Register (TMOD)

It configures the various timer operation modes .

TMOD is a 8-bit register

- The lower 4 bits are for Timer 0
- The upper 4 bits are for Timer 1

In both cases,

- The lower 2 bits are used to set the timer mode
- The upper 2 bits to specify the operation

TMOD register

Msb							Lsb
GATE	C/T	M1	M0	GATE	C/T	M1	M0
TIMER 1				TIMER 0			

Gating control (GATE) bit

Starting and stopping of timers of 8051 by

a) software

GATE=0. The start and stop of the timer are controlled by the TR (timer start) bits TR0 and TR1 of TCON register.

– The SETB instruction starts it, and it is stopped by the CLR instruction

b) hardware

The start and stop of the timer are controlled by an external source is achieved by making GATE=1 in the TMOD register

Timer/counter is enabled while

- the INTx pin is high (pins P3.2 and P3.3 for timers 0 and 1) and
- the Tx control pin is set (pins P3.4 and P3.5 for timers 0 and 1).

Timer or counter select (C/T) bit

- C/T = 0 - timer operation (input from internal system clock divided 12)
- C/T = 1 - for counter operation (input from Tx input pin)

Mode bits (M1, M0) bits

M1	**M0**	**Mode**	**Operating Mode**
0	0	0	**13-bit timer mode** 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	**16-bit timer mode** 16-bit timer/counter THx and TLx are cascaded; there is no prescaler
1	0	2	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded TLx each time it overflows
1	1	3	**Split timer mode**

2. Timer register

Both Timer 0 and Timer 1 are 16 bits wide, each accessed as two separate registers of low byte and high byte.

- The low byte register is called TL0/TL1 and
- The high byte register is called TH0/TH1

3. Timer/Counter Control Register (TCON)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	8Fh	bit is automatically set on the Timer 1 overflow.
TR1	8Eh	bit enables the Timer 1. 1 - Timer 1 is enabled. 0 - Timer 1 is disabled.
TF0	8Dh	bit is automatically set on the Timer 0 overflow.
TR0	8Ch	bit enables the timer 0. 1 - Timer 0 is enabled. 0 - Timer 0 is disabled.

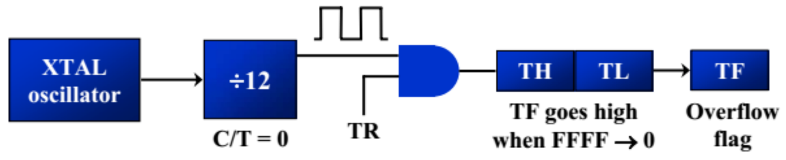
The lower 4 bits are set aside for controlling the interrupt bits

Steps to generate a time delay

1. Load the TMOD value register select timer (0 or 1) and timer mode (0 or 1),
2. Load registers TL and TH with initial count value
3. Start the timer by setting TRx bit of TCON register
4. Keep monitoring the timer flag (TF) with the JNB TFx, target instruction to see if it is high
5. Get out of the loop when TF becomes high
6. Stop the timer by clearing TRx bit of TCON register
7. Clear the TF flag for the next round
8. Go back to Step 2 to load TH and TL again

Operation of mode 1:

1. It is a 16-bit timer; therefore, it allows value of 0000 to FFFFH to be loaded into the timer's register TL and TH
2. After TH and TL are loaded with a 16-bit initial value, the timer must be started . This is done by SETB TR0 for timer 0 and SETB TR1 for timer 1
3. After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFFH When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag)
   - Each timer has its own timer flag: TF0 for timer 0, and TF1 for timer 1
   - When this timer flag is raised, stop the timer with the instructions CLR TR0 or CLR TR1, for timer 0 and timer 1, respectively
4. After the timer reaches its limit and rolls over, in order to repeat the process
   - TH and TL must be reloaded with the original value, and
   - TF must be reloaded to 0



Example
Create a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay.

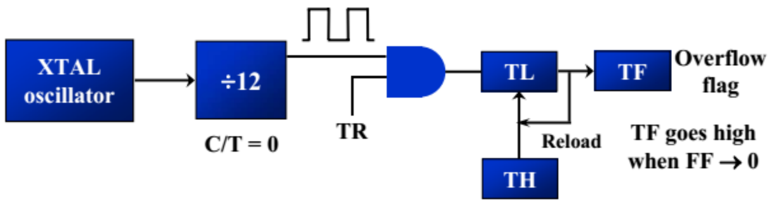
Solution
```
MOV TMOD, #01             ;Timer 0, mode 1(16-bit mode)
HERE: MOV TL0, #0F2H      ;TL0=F2H, the low byte
MOV TH0, #0FFH            ;TH0=FFH, the high byte
CPL P1.5                  ;toggle P1.5
ACALL DELAY
SJMP HERE
DELAY:
SETB TR0                  ;start the timer 0
AGAIN: JNB TF0, AGAIN     ;monitor timer flag 0 until it rolls over
CLR TR0                   ;stop timer 0
CLR TF0                   ;clear timer 0 flag
RET
```

Example
A 8051 C program to toggle only bit P1.5 continuously every 50 ms.
Use Timer 0, mode 1 (16-bit) to create the delay.

Solution
```
#include <reg51.h>
void T0M1Delay(void);
sbit mybit=P1^5;
void main(void){
while (1) {
mybit=~mybit;
T0M1Delay();
}
}
void T0M1Delay(void){
TMOD=0x01;
TL0=0xFD;
TH0=0x4B;
TR0=1;
while (TF0==0);
TR0=0;
TF0=0;
}
```

Mode 2 operation

1. It is an 8-bit timer. It allows values of 00 to FFH to be loaded into the timer's register TH
2. After TH is loaded with the 8-bit value, the 8051 gives a copy of it to TL
3. Start the timer by the instruction SETB TRx for timers
4. After the timer is started,
   - It counts up by incrementing the TL register to FFH
   - then it rolls over from FFH to 00,
   - sets high the TF (timer flag)
   - TL is reloaded automatically with the original value in the TH register



Example

Generated square wave on pin P1.0 using 8-bit/auto reload mode 2

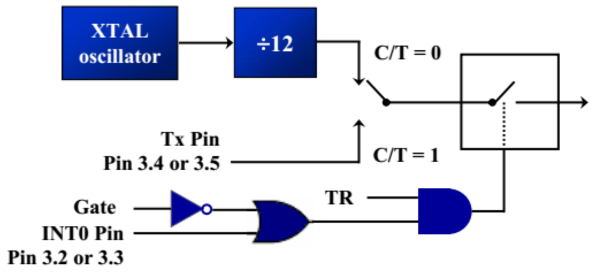
Solution

```
MOV TMOD,#20H    ; T1/8-bit/auto reload
MOV TH1,#5       ; TH1 = 5
SETB TR1         ; start the timer 1
BACK:
JNB TF1,BACK     ;till timer rolls over
CPL P1.0         ;P1.0 to hi, lo
CLR TF1          ;clear Timer 1 flag
SJMP BACK        ;mode 2 is auto-reload
```

Start or stop the timer externally

If GATE = 1, the start and stop of the timer are done externally through pins P3.2 and P3.3 for timers 0 and 1, respectively

Although the TRx is turned on by the "SETB TRx " instruction , the hardware way allows to start or stop the timer externally at any time via a simple switch



Value (in hex) loaded into TH for the following cases.

(a) MOV TH1,#-200 (b) MOV TH0,#-60 (c) MOV TH1,#-3
(d) MOV TH1,#-12 (e) MOV TH0,#-48

Solution:

- In Windows calculator, select decimal and enter 200.
- Then select hex, then +/- to get the TH value.
- Use only the right two digits and ignore the rest since data is 8-bit.

Decimal 2's complement (TH value)

-3	FDH
-12	F4H
-48	D0H
-60	C4H
-200	38H

The number 200 is the timer count till the TF is set to 1

- The advantage of using negative values is that, there is no need to calculate the value loaded to THx

Counter programming

- In counter mode, external pulse increments the TH, TL registers
- The timer is used as a counter by putting C/T = 1 in the TMOD registers, so that the 8051 gets its pulses from outside
- The counter counts up as pulses are fed from P3.4 and P3.5 pins , these pins are called T0 (timer 0 input) and T1 (timer 1 input)

Pin	Port Pin	Function	Description
14	P3.4	T0	Timer/counter 0 external input
15	P3.5	T1	Timer/counter 1 external input

Example: Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2, which connects to 8 LEDs.

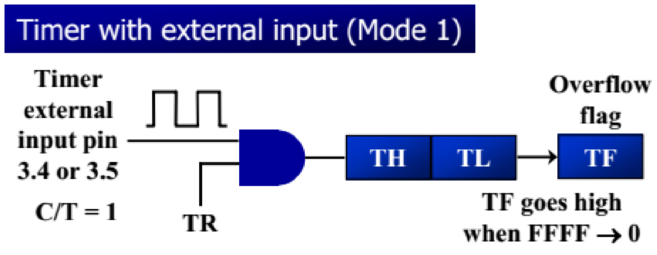
Solution:

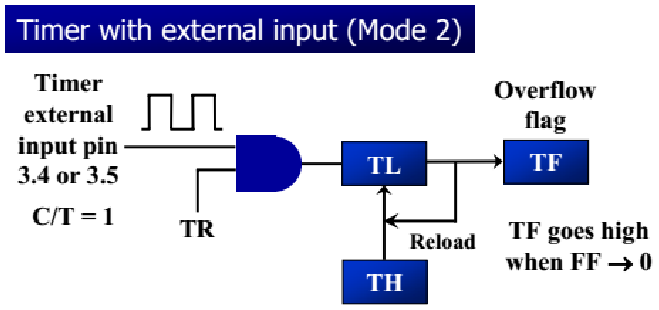
```
MOV TM0D,#01100000B        ;counter 1, mode 2, C/T=1 external pulses
MOV TH1,#0                 ;clear TH1
SETB P3.5                  ;make T1 input
AGAIN: SETB TR1            ;start the counter
BACK: MOV A,TL1            ;get copy of TL
MOV P2,A                   ;display it on port 2
JNB TF1,Back ;             keep doing, if TF = 0
CLR TR1                    ;stop the counter 1
CLR TF1                    ;make TF=0
SJMP AGAIN                 ;keep doing it
```

Note:

The timer works with a clock frequency of 1/12 of the XTAL frequency;
therefore, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency.
As a result, each clock has a period of T = 1/921.6kHz = 1.085us.

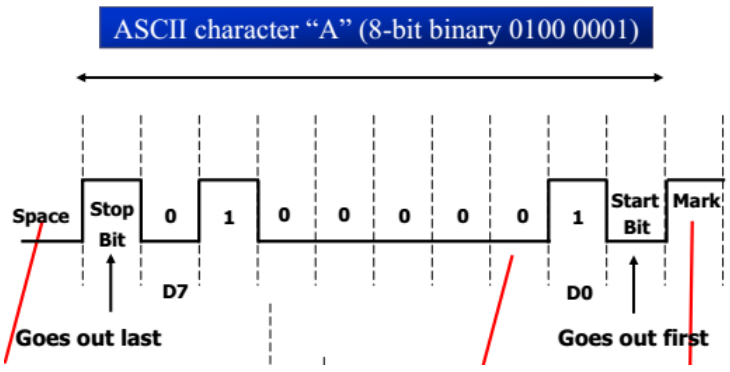
In other words, Timer 0 counts up each 1.085 us resulting in delay = number of counts × 1.085us.
The number of counts for the roll over is FFFFH – FFF2H = 0DH (13d). However, we add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raise the TF flag. This gives 14 × 1.085us = 15.19us for half the pulse.
For the entire period it is T = 2 × 15.19us = 30.38us as the time delay generated by the timer.


Timer with external input (Mode 1)
Timer external input pin 3.4 or 3.5
C/T = 1
TR
TH
TL
TF
Overflow flag
TF goes high when FFFF → 0


Timer with external input (Mode 2)
Timer external input pin 3.4 or 3.5
C/T = 1
TR
TL
TF
Overflow flag
Reload
TH
TF goes high when FF → 0

Asynchronous serial data communication

- used for character-oriented transmissions
- Each character is placed in between start and stop bits, this is called framing
- The start bit is always one bit, but the stop bit can be one or two bits
- The start bit is always a 0 (low) and the stop bit(s) is 1 (high)



- When there is no transfer, the signal is 1 (high), which is referred to as *mark*
- The transmission begins with a start bit followed by D0, the LSB, then the rest of the bits until MSB (D7), and finally, the one stop bit indicating the end of the character
- Bits per second - The rate of data transfer in serial data communication is stated in bps
- Baud rate - the number of signal changes per second

Set baud rate

- Dividing 1/12 of the crystal frequency by 32 is the default value upon activation of the 8051 RESET pin
- The machine cycle frequency of 8051 = 11.0592 / 12 = 921.6 kHz, and 921.6 kHz / 32 = 28,800 Hz is frequency by UART to timer 1 to set baud rate.

(a) 28,800 / 03 = 9600	where -03 = FDh is loaded into TH1
(b) 28,800 / 12 = 2400	where -12 = F4h is loaded into TH1
(c) 28,800 / 24 = 1200	where -24 = E8h is loaded into TH1

TF is set to 1 every 12 ticks, so it functions as a frequency divider

Pins of 8051 for serial communication

For transferring data TxD (P3.0) and receiving data RxD (P3.1) serially

Registers involved in serial data communication programming

1. Serial control (SCON) register to define the data frame, tranmision / reception control,
2. Serial data buffer (SBUF) register stores a byte data to be transferred / received
3. TMOD and TCON registers to setup the baud rate

SCON register

An 8-bit register used to program the start bit, stop bit, and data bits of data framing, etc

D7							D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
Note: Make SM2, TB8, and RB8 =0		

SM0, SM1

They determine the framing of data by specifying the number of bits per character, and the start and stop bits

SM0 = 0, SM1 = 1 Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit

REN (receive enable)

It is a bit-adressable register. When it is high, it allows 8051 to receive data on RxD pin

If low, the receiver is disable

TI (transmit interrupt)

When 8051 finishes the transfer of 8-bit character, it raises TI flag to indicate that it is ready to transfer another byte.

TI bit is raised at the beginning of the stop bit

RI (receive interrupt)

When 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in SBUF register. It raises the RI flag bit 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

SBUF register

An 8-bit register used for serial communication

SBUF register stores a byte data to be transferred via the TxD line,

SBUF holds the byte of data when it is received by 8051 RxD line.

Steps in programming the 8051 to transfer character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate
2. The TH1 is loaded with one of the values to set baud rate for serial data transfer

3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
4. TR1 is set to 1 to start timer 1
5. TI is cleared by CLR TI instruction
6. The character byte to be transferred serially is written into SBUF register
7. The TI flag bit is monitored with the use of instruction JNB TI,xx to see if the character has been transferred completely
8. To transfer the next byte, go to step 5

Example
Write a program for the 8051 to transfer "YES" serially at 9600 baud, 8-bit data, 1 stop bit, do this continuously

Solution:
```
MOV TMOD,#20H               ;timer 1,mode 2(auto reload)
MOV TH1,#-3                 ;9600 baud rate
MOV SCON,#50H               ;8-bit, 1 stop, REN enabled
SETB TR1                    ;start timer 1
AGAIN:
MOV A,#"Y"                  ;transfer "Y"
ACALL TRANS
MOV A,#"E"                  ;transfer "E"
ACALL TRANS
MOV A,#"S"                  ;transfer "S"
ACALL TRANS
SJMP AGAIN                  ;keep doing it
;serial data transfer subroutine
TRANS: MOV SBUF,A           ;load SBUF
HERE: JNB TI,HERE           ;wait for the last bit
CLR TI                      ;get ready for next byte
RET
```

Steps in programming the 8051 to receive character bytes serially
1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate,
2. TH1 is loaded to set baud rate,
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits,
4. TR1 is set to 1 to start timer 1
5. RI is cleared by CLR RI instruction
6. The RI flag bit is monitored with the use of instruction JNB RI,xx to see if an entire character has been received yet
7. When RI is raised, SBUF has the byte, its contents are moved into a safe place
8. To receive the next character, go to step 5

Example:
Write a program for the 8051 to receive bytes of data serially, and put them in P1, set the baud rate at 4800, 8-bit data, and 1 stop bit.

Solution:

```
MOV TMOD,#20H          ;timer 1,mode 2(auto reload)
MOV TH1,#-6            ;4800 baud rate
MOV SCON,#50H          ;8-bit, 1 stop, REN enabled
SETB TR1                ;start timer 1
HERE: JNB RI,HERE      ;wait for char to come in
MOV A,SBUF             ;saving incoming byte in A
MOV P1,A               ;send to port 1
CLR RI                 ;get ready to receive next  byte
SJMP HERE              ;keep getting data
```

Example 10-5
Assume that the 8051 serial port is connected to the COM port of IBM PC, and on the PC the terminal.exe program is used to send and receive data serially.
P1 and P2 of the 8051 are connected to LEDs and switches, respectively.
Write an 8051 program to (a) send to PC the message "We Are Ready", (b) receive any data send by PC and put it on LEDs connected to P1, and (c) get data on switches connected to P2 and send it to PC serially.
The program should perform part (a) once, but parts (b) and (c) continuously, use 4800 baud rate.

Solution:

```
ORG 0
MOV P2,#0FFH                ;make P2 an input port
MOV TMOD,#20H               ;timer 1, mode 2
MOV TH1,#0FAH               ;4800 baud rate
MOV SCON,#50H               ;8-bit, 1 stop, REN enabled
SETB TR1                    ;start timer 1
MOV DPTR,#MYDATA            ;load pointer for message
H_1: CLR A
MOV A,@A+DPTR               ;get the character
JZ B_1                      ;if last character get out
ACALL SEND                  ;otherwise call transfer
INC DPTR                    ;next one
SJMP H_1                    ;stay in loop
B_1: MOV A,P2               ;read data on P2
ACALL SEND                  ;transfer it serially
ACALL RECV                  ;get the serial data
MOV P1,A                    ;display it on LEDs
SJMP B_1                    ;stay in loop indefinitely
```

```
;----serial data transfer. ACC has the data------
SEND: MOV SBUF,A                ;load the data
H_2: JNB TI,H_2                 ;stay here until last bit gone
CLR TI                          ;get ready for next char
RET                             ;return to caller
;----Receive data serially in ACC----------------
RECV: JNB RI,RECV               ;wait here for char
MOV A,SBUF                      ;save it in ACC
CLR RI                          ;get ready for next char
RET                             ;return to caller
;-----The message---------------
MYDATA: DB "We Are Ready",0
END
```

Interrupt programming

Six interrupts are provided in 8051

a. Reset – power-up reset
b. Two timers interrupts for timer 0 and timer 1
c. Two hardware external interrupts INT0 and INT1 using pins P3.2 and P3.3
d. Serial communication interrupt for both receive and transfer

Note:

- Upon reset, all interrupts are disabled (masked),
- The interrupts must be enabled by software so that the microcontroller responds to them

Interrupts and interrupt vector table

Priority Upon Reset	**Interrupt**	**ROM Location**	**Pin**
	Reset	0000	
1	External Interrupt 0 (INT0)	0003	P3.2
2	Timer Interrupt 0 (TF0)	000B	
3	External Interrupt 1 (INT1)	0013	P3.3
4	Timer Interrupt 1 (TF1)	001B	
5	Serial Communication (RI and TI)	0023	

Programming

Enable the interrupts through interrupt enable register IE for enabling (unmasking) and disabling (masking) interrupts.

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Enables or disables all interrupts
--	IE.6	Not implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow or capture interrupt (8952)
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 1 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt 0

1 = Enable, 0 = disable

Example:

```
MOV IE,#10010110B          ;enable serial, ;timer 0, EX1
or
SETB IE.7                  ;EA=1, global enable
SETB IE.4                  ;enable serial interrupt
SETB IE.1                  ;enable Timer 0 interrupt
SETB IE.2                  ;enable EX1
```

```
CLR IE.1                    ;mask (disable) timer 0 interrupt only
CLR IE.7                    ;disable all interrupt
```

Timer interrupt

If the timer interrupt in the IE register is enabled,

- The microcontroller is interrupted , whenever the timer rolls over, TF is raised,
- The microcontroller jumps to the interrupt vector table to service the ISR,
- In the ISR there is no need of for a “CLR TFx” instruction before RETI. Since the 8051 clears the TF flag internally upon jumping to the interrupt vector table.
- The instruction RETI at the end of ISR does
  a. Popping off return address from stack into program counter to resume the main program,
  b. Clear the TF0, TF1 and interrupt-in-service flags IE0 (TCON.1) , IE1 (TCON.3) in the TCON register indicating that servicing of interrupt is over and a new interrupt can be accepted.

Example

Write a program to generate a square wave if 50Hz frequency on pin P1.2. Use an interrupt for timer 0. Assume that XTAL=11.0592 MHz

Solution:

```
ORG 0
LJMP MAIN
ORG 000BH                           ;ISR for Timer 0
CPL P1.2
MOV TL0,#00
MOV TH0,#0DCH
RETI

ORG 30H
;--------main program for initialization
MAIN:   MOV TMOD,#00000001B ;Timer 0, Mode 1
MOV TL0,#00
MOV TH0,#0DCH
MOV IE,#82H                         ;enable Timer 0 interrupt
SETB TR0
HERE:   SJMP HERE
END
```

Example

Write a program to create a square wave of 200 µs period (5000 Hz) on pin P2.1. Use timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.
Use timer 0 in mode 2 (auto reload). TH0 = 100/1.085 us = 92

Solution:

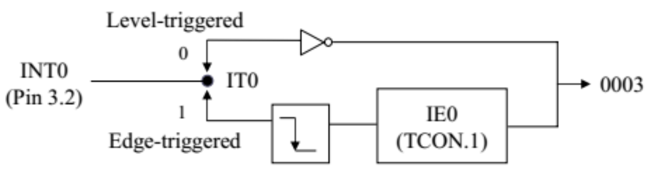
```
ORG 0000H
LJMP MAIN
```

```
;ISR for timer 0 to generate square wave
ORG 000BH                ;Timer 0 interrupt vector table
CPL P2.1                 ;toggle P2.1 pin
RETI                     ;return from ISR
;The main program for initialization
ORG 0030H                ;bypass vector table space
MAIN: MOV TMOD,#02H      ;Timer 0, mode 2
MOV TH0,#-92             ;TH0=A4H for -92
MOV IE,#82H              ;IE=10000010b enable Timer 0
SETB TR0                 ;Start Timer 0
BACK: SJMP BACK          ;keep looping  unless interrupted by TF0
END
```

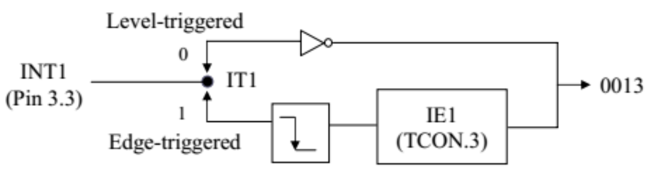
External hardware interrupts

- Two external hardware designated as INT0 and INT1 interrupts upon pins P3.2 and P3.3
- The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
- Two activation levels for the external hardware interrupts - Level triggered and Edge triggered
- Level triggered interrupts is the default mode upon reset

Activation of INT0



Activation of INT1



level-triggered mode

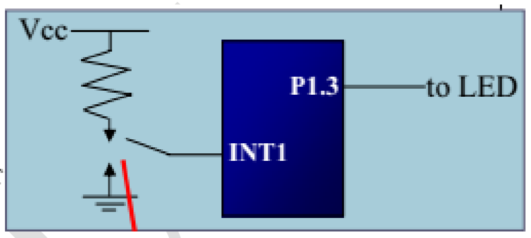
- INT0 and INT1 pins are normally high
- A low-level signal triggers the interrupt
- The low-level signal at the INT pin must be removed before the execution of the last instruction of the ISR, RETI; otherwise, another interrupt will be generated

Programming:

Configure IE register - enable bits EA (IE.7), EX1 (IE.2), / EX0 (IE.0)

Example:
Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on.

Solution:
```
ORG 0000H
LJMP
;--ISR for INT1 to turn on LED
ORG 0013H                ;INT1 ISR
SETB P1.3                 ;turn on LED
MOV R3,#255
BACK: DJNZ R3,BACK       ;keep LED on for a while
CLR P1.3                 ;turn off the LED
RETI                     ;return from ISR
;--MAIN program for initialization
ORG 30H
MAIN: MOV IE,#10000100B ;enable external INT 1
HERE: SJMP HERE          ;stay here until get interrupted
END
```



Pressing the switch will cause the LED to be turned on. If it is kept activated, the LED stays on.

Edge-triggered mode
A falling edge (high to low) signal at pins INT0 and INT1 interrupts the micro-controller and force to jump to location in the vector table to service the ISR.

Programming:
1. Configure IE register - enable bits EA (IE.7), EX1 (IE.2), / EX0 (IE.0)
2. Configure TCON register - enable bits IT1 (TCON.2), IT0 (TCON.0) to make an edge triggered interrupt.

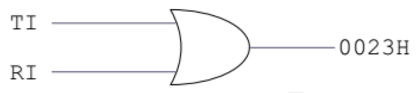
Example:
Assume that pin 3.3 (INT1) is connected to a pulse generator, write a program in which the falling edge of the pulse will send a high to P1.3, which is connected to an LED. The LED is turned on and off at the same rate as the pulses are applied to the INT1 pin.

Solution:
```
ORG 0000H
LJMP MAIN
;--ISR for hardware interrupt INT1 to turn on LED
ORG 0013H                ;INT1 ISR
SETB P1.3                ;turn on LED
MOV R3,#255
BACK: DJNZ R3,BACK       ;keep the LED on for a while
CLR P1.3                 ;turn off the LED
RETI                     ;return from ISR
```

```
;------MAIN program for initialization
ORG 30H
MAIN: SETB TCON.2         ;make INT1 edge-triggered int.
MOV IE,#10000100B         ;enable External INT 1
HERE: SJMP HERE           ;stay here until get interrupted
END
```

Serial communication interrupt

- There is only one interrupt for serial communication used to both send and receive data
- Serial interrupt is invoked by TI or RI flags If the serial interrupt bit ES (IE.4) in the IE register is enabled,
- When RI or TI is raised the 8051 gets interrupted and jumps to memory location 0023H to execute the ISR
- In the ISR examine the TI and RI flags to see which one caused the interrupt and respond accordingly
- TI (transfer interrupt) is raised when the last bit of the framed data, the stop bit, is transferred, indicating that the SBUF register is ready to transfer the next byte
- RI (received interrupt) is raised when the entire frame of data, including the stop bit, is received indicating that the the SBUF register has a byte.
- The last instruction before RETI is the clearing of RI or TI flags.
- The serial interrupt is used mainly for receiving data and is never used for sending data serially.



Programming
Example
Write a program in which the 8051 reads data from P1 and writes it to P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that XTAL=11.0592. Set the baud rate at 9600.

Solution:

```
ORG 0000H
LJMP MAIN
ORG 23H
LJMP SERIAL               ;jump to serial int ISR

ORG 30H
MAIN: MOV P1,#0FFH        ;make P1 an input port
MOV TMOD,#20H             ;timer 1, auto reload
MOV TH1,#0FDH             ;9600 baud rate
MOV SCON,#50H             ;8-bit,1 stop, ren enabled
MOV IE,10010000B          ;enable serial int.
SETB TR1                  ;start timer 1
BACK: MOV A,P1            ;read data from port 1
```

```
MOV SBUF,A              ;give a copy to SBUF
MOV P2,A                ;send it to P2
SJMP BACK               ;stay in loop indefinitely

;-----------------serial port isr
ORG 100H
SERIAL: JB TI,TRANS     ;jump if TI is high
MOV A,SBUF              ;otherwise due to receive
CLR RI                  ;clear RI since CPU doesn't
RETI                    ;return from ISR
TRANS: CLR TI           ;clear TI since CPU doesn't
RETI                    ;return from ISR
END
```

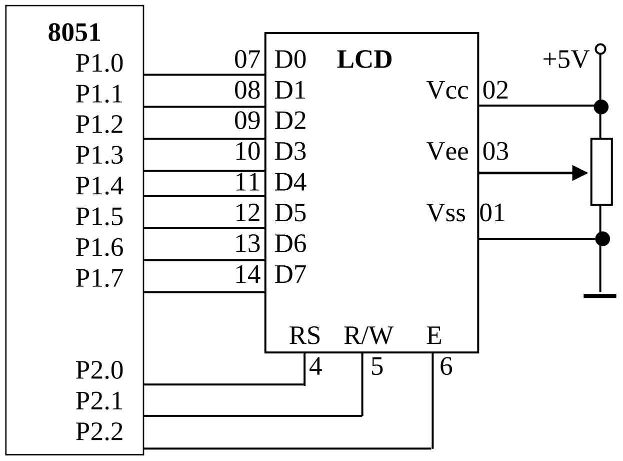
Note:

- The moment a byte is written into SBUF it is framed and transferred serially.
- When the last bit (stop bit) is transferred the TI is raised,
- The serial interrupt is invoked if corresponding bit in the IE register is high.
- In the serial ISR, check for both TI and RI since both could have invoked interrupt

LCD Programming

LCD displays modules are the replacement for the 7 segment display due to their versatility.

LCD Connections



Programming

Consists of

1. Configure LCD through control register,
2. Send data through data register.

Configuration of LCD

a. Typical command codes to the control register

Ser.	Description	Code
1.	Initialize LCD mode – lines, matrix	38H - (2 lines, 5x7 matrix)
2.	Display ON, cursor ON	0EH
3.	Clear LCD	01H
4.	Shift cursor RIGHT	06H
5.	Cursor line and position	86H – (line 1, position 6) C6H – (line 2, position 6)

Consult data sheet of LCD for complete command code

b. Procedure

- Make pin RS (register select) = 0 – select control register,
- Make pin R/W (read or write select) = 0 for write,
- Send a high-to-low pulse to the E (enable) pin (through delay) to enable the internal latch of the LCD for write,
- Give the LCD some delay.

Send data

c. Procedure

- Send data over D7 – D0 lines
- Make pin RS (register select) = 1 – select data register,
- Make pin R/W (read or write select) = 0 for write,

- Send a high-to-low pulse to the E pin (through delay) to enable the internal latch of the LCD for write.

Sending code / data to the LCD with checking busy flag
Monitor the busy flag before issuing a command / data to the LCD.

d. Procedure

- Make pin RS (register select) = 0 – access control register,
- Make pin R/W (read or write select) = 1 for read,
- Send a low-to-high pulse to the E pin (through delay) to enable the internal latch of the LCD for read,
- Loop until busy flag = 0.
  After reading the command register if bit D7 (the busy flag) is high, the LCD is busy and no information (commands or data) should be send to it. Only when D7=0, commands or data can be send to LCD.

Program # 1
Calls a time delay before sending next data/command

```
; P1.0-P1.7=D0-D7, P2.0=RS, P2.1=R/W, P2.2=E
ORG 0H
; configure LCD
MOV A,#38H     ;init. lcd 2 lines, 5x7 matrix
ACALL COMNWRT       ;call command subroutine
ACALL DELAY         ;give LCD some time
MOV A,#0EH           ;display on, cursor on
ACALL COMNWRT       ;call command subroutine
ACALL DELAY         ;give LCD some time
MOV A,#01           ;clear LCD
ACALL COMNWRT       ;call command subroutine
ACALL DELAY         ;give LCD some time
MOV A,#06H          ;shift cursor right
ACALL COMNWRT       ;call command subroutine
ACALL DELAY         ;give LCD some time
MOV A,#84H          ;cursor at line 1, pos. 4
ACALL COMNWRT       ;call command subroutine
ACALL DELAY         ;give LCD some time
; send data
MOV A,#'N'          ;display letter N
ACALL DATAWRT       ;call display subroutine
ACALL DELAY         ;give LCD some time
MOV A,#'O'          ;display letter O
ACALL DATAWRT       ;call display subroutine
AGAIN: SJMP AGAIN   ;stay here
```

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

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

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

Program # 2

Check busy flag before sending data, command to LCD

```
; P1.0-P1.7=D0-D7, P2.0=RS, P2.1=R/W, P2.2=E
ORG 0H
MOV A,#38H            ;init. LCD 2 lines ,5x7
matrix
ACALL COMMAND         ;issue command
MOV A,#0EH             ;LCD on, cursor on
ACALL COMMAND         ;issue command
MOV A,#01H            ;clear LCD command
ACALL COMMAND         ;issue command
MOV A,#06H            ;shift cursor right
ACALL COMMAND         ;issue command
MOV A,#86H            ;cursor: line 1, pos. 6
ACALL COMMAND         ;command subroutine
MOV A,#'N'            ;display letter N
ACALL DATA_DISPLAY
MOV A,#'O'            ;display letter O
ACALL DATA_DISPLAY
HERE:SJMP HERE        ;STAY HERE

COMMAND:
ACALL READY           ;is LCD ready?
MOV P1,A              ;issue command code
CLR P2.0              ;RS=0 for command
CLR P2.1              ;R/W=0 to write to LCD
SETB P2.2             ;E=1 for H-to-L pulse
CLR P2.2              ;E=0,latch in
RET

DATA_DISPLAY:
ACALL READY           ;is LCD ready?
MOV P1,A              ;issue data
SETB P2.0             ;RS=1 for data
CLR P2.1              ;R/W =0 to write to LCD
SETB P2.2             ;E=1 for H-to-L pulse
CLR P2.2              ;E=0,latch in
RET

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

Program # 3

Call a time delay before sending next data/command

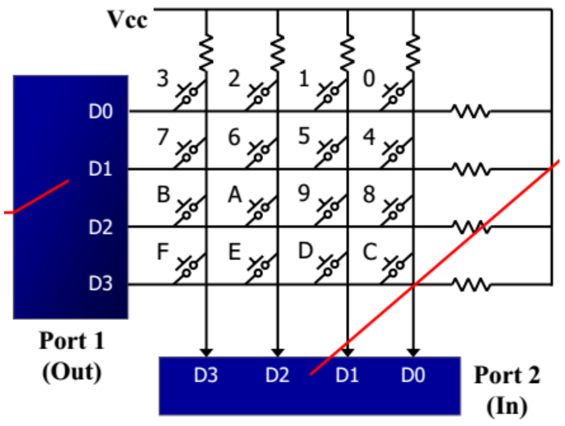
```
; P1.0-P1.7=D0-D7, P2.0=RS, P2.1=R/W, P2.2=E
ORG 0
MOV DPTR,#MYCOM
C1: CLR A
MOVC A,@A+DPTR
ACALL COMNWRT       ;call command subroutine
ACALL DELAY         ;give LCD some time
INC DPTR
JZ SEND_DAT
SJMP C1
SEND_DAT:
MOV DPTR,#MYDATA
D1: CLR A
MOVC A,@A+DPTR
ACALL DATAWRT       ;call command subroutine
ACALL DELAY         ;give LCD some time
INC DPTR
JZ AGAIN
SJMP D1
AGAIN: SJMP AGAIN   ;stay here

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

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

DELAY: MOV R3,#250  ;50 or higher
HERE2: MOV R4,#255  ;R4 = 255
HERE: DJNZ R4,HERE  ;stay until R4 becomes 0
DJNZ R3,HERE2
RET

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

Example
Write an 8051 C program to send letters ‘M’, ‘D’, and ‘E’ to the LCD using the busy flag method.
Solution:

```
#include <reg51.h>
sfr ldata = 0x90;   //P1=LCD data pins
sbit rs = P2^0;
sbit rw = P2^1;
sbit en = P2^2;
sbit busy = P1^7;
void main(){
lcdcmd(0x38);
lcdcmd(0x0E);
lcdcmd(0x01);
lcdcmd(0x06);
lcdcmd(0x86);      //line 1, position 6
lcdcmd(‘M’);
lcdcmd(‘D’);
lcdcmd(‘E’);
}
void lcdcmd(unsigned char value){
lcdready();         //check the LCD busy flag
ldata = value;      //put the value on the pins
rs = 0;
rw = 0;
en = 1;             //strobe the enable pin
MSDelay(1);
en = 0;
return;
}
void lcddata(unsigned char value){
lcdready();         //check the LCD busy flag
ldata = value;      //put the value on the pins
rs = 1;
rw = 0;
en = 1;             //strobe the enable pin
MSDelay(1);
en = 0;
return;
}
void lcdready(){
busy = 1;           //make the busy pin at input
rs = 0;
rw = 1;
while(busy==1){  //wait here for busy flag
en = 0;             //strobe the enable pin
MSDelay(1);
en = 1;
}
void lcddata(unsigned int itime){
unsigned int i, j;
for(i=0;i<itime;i++)
for(j=0;j<1275;j++);
}
```

Keyboard interfacing

- Keyboards are organized in a matrix of rows and columns
- The CPU accesses both rows and columns through ports
- Thus, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor
- When a key is pressed, a row and a column make a contact. Otherwise, there is no connection between rows and columns **.**



Matrix Keyboard Connection to ports

- The rows are connected to an output port and
- The columns are connected to an input port.
- If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground.
- If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (Vcc).

Detect a key press

- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch,
- Then it reads the columns
  - If the data read from columns is D3 – D0 =1111, no key has been pressed and the process continues till key press is detected
  - If one of the column bits has a zero, this means that a key press has occurred
    For example, if D3 – D0 = 1101, this means that a key in the D1 column has been pressed.

Detect row of key press

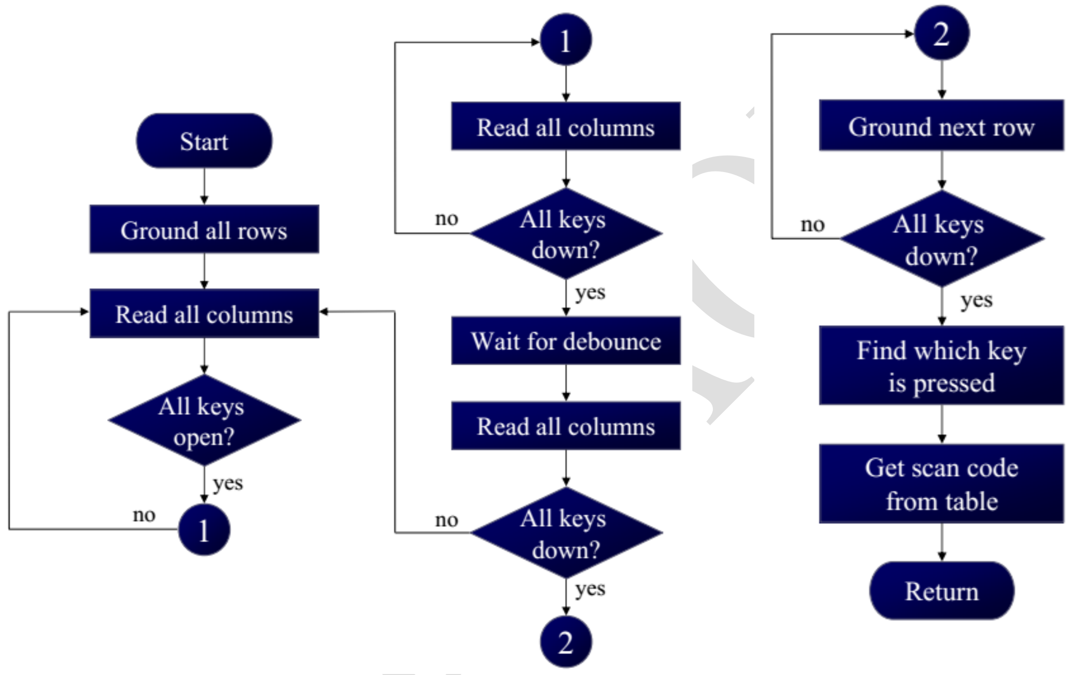
- To detect which row key press belongs to, it grounds one row at a time, reading the columns each time.
  - If it finds that all columns are high, this means that the key press cannot belong to that row
  - Therefore, it grounds the next row and continues until it finds the row the key press belongs to

- Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or ASCII) for that row

Identify the key press

- To identify the key press, it rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low
  - Upon finding the zero, it pulls out the ASCII code for that key from the look-up table otherwise, it increments the pointer to point to the next element of the look-up table **.**

Flowchart for the program



Program

Keyboard Program

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

Tentative Projects

a. Blinking single LED
b. Blinking LEDs animation connected to a port
c. Display a character on a 7 segment display
d. Display characters of a string one by one on a 7 segment display
e. Display characters of a string on array of five 7 segment display
f. Display characters of a string scrolling on array of five 7 segment display
g. Door entry audio alarm with door number display on a 7 segment display
h. Simple octal key board
i. 3x3 matrix key board
j. Automatic water pump controller
k. Interfacing LCD display controller
l. Display characters of a string scrolling on LCD display controller
m. Serial data transmission and reception with display on LCD display controller – close loop to itself
n. Serial communication with PC using serial to usb converter / adruino controller (MPU removed), putty.exe as hyperterminal

- Use 89C2051 (20 pin) Microprocessor

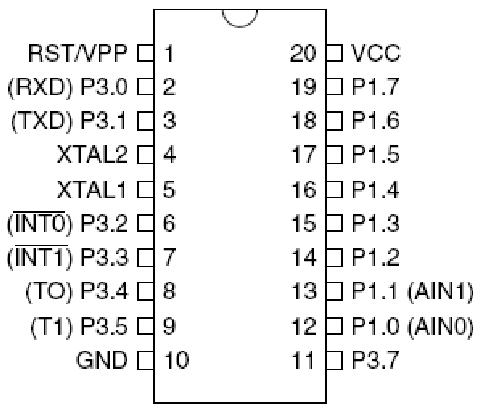
Hardware for MCU based projects:

Ser.	Item	Qty.
1	MCU – 89C2051	1
2	ZIF socket 20 pin	1
3	Resistance 8.2 kOm ¼ W 1 kOm ¼ W	1 10
4	Capacitor (electrolytic) 10 µF 50V + 100 µF 50V	1+1
5	Capacitor (non electrolytic) 30 pF	2
6	Crystal 11.0592 MHz	1
7	Voltage regulator 7805	1
8	Micro switch push type	10
9	LED red	10
10	Breadboard	1
11	Transistors NPN low power Mid power	8 2
12	Battery (storage) 9V	1
13	Speaker (for computer)	1
14	Buzzer 5V	1
15	Musical IC UM66	1
16	7 segment display (common cathode)	5
17	20 char x 2 lines display controller	1
18	Connecting wires (multicolor)	
19	Long nose mini pliers with cutter	1

Presentation:

Page 1	Name of University
	Subject Name & Code
	Project name
	Participant name & ID
	Section & group
	Name of instructor
	Date of submission
Page 2	Objective of project
	Circuit diagram
	List of components
Page 3	Flowchart of program
	Printout of program
Page 4	Brief description of program
	Achievements (Result)

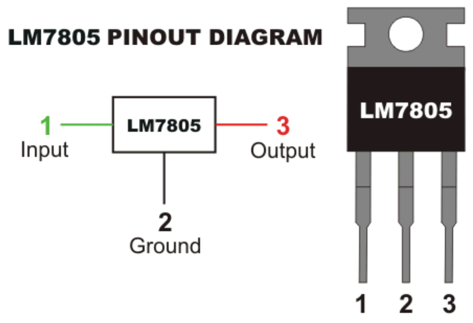
The AT89C2051

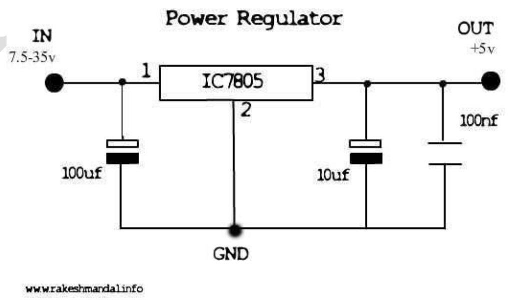


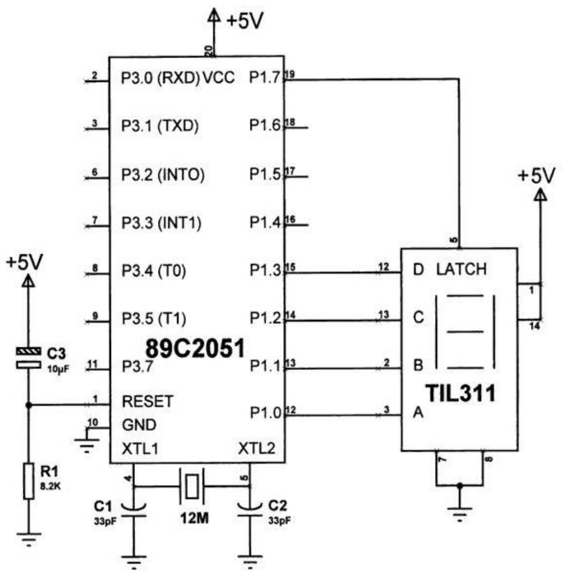
Features of the AT89C2051 processor:
Compatible with Intel 8051
2kB of internal FLASH program memory
Voltage range from 2.7 to 6 V
Clock 0 Hz to 24 MHz
128 bytes of internal RAM
15 programmable I/O pins
Two 16-bit counters/timers
Analog comparator
5 interrupt sources
Programmable UART - (RS232)

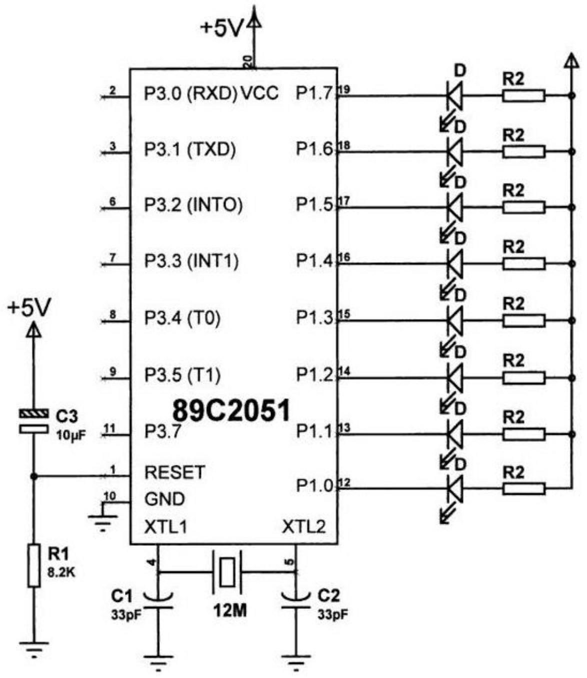
- Port pins P1.2 to P1.7 provide internal pull-ups.
- Port pins P1.0 and P1.1 require external pull-ups.
- The analog comparator on pins P1.0 and P1.1. Its output is accessible to the software via the P3.6 bit.

Voltage Regulator 7805

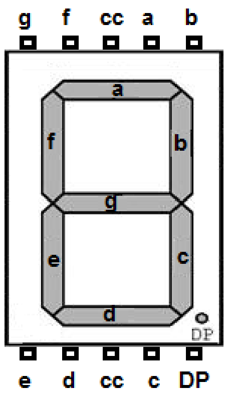




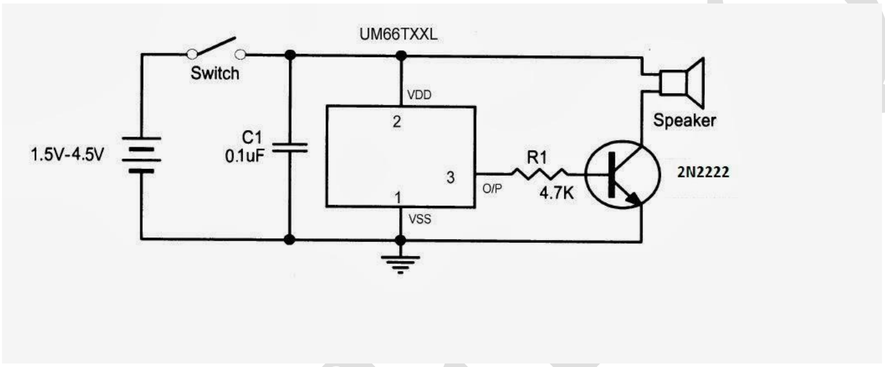




Common cathode & segment display pinout diagram



Musical IC UM66



TYPES OF INSTRUCTIONS

1. Arithmetic Instructions
2. Branch Instructions
3. Data Transfer Instructions
4. Logic Instructions
5. Bit-oriented Instructions

1. ARITHMETIC INSTRUCTIONS

Mnemonic	**Description**	**Byte**	**Cycle**
ADD A,Rn	Adds the register to the accumulator	1	1
ADD A,direct	Adds the direct byte to the accumulator	2	2
ADD A,@Ri	Adds the indirect RAM to the accumulator	1	2
ADD A,#data	Adds the immediate data to the accumulator	2	2
ADDC A,Rn	Adds the register to the accumulator with a carry flag	1	1
ADDC A,direct	Adds the direct byte to the accumulator with a carry flag	2	2
ADDC A,@Ri	Adds the indirect RAM to the accumulator with a carry flag	1	2
ADDC A,#data	Adds the immediate data to the accumulator with a carry flag	2	2
SUBB A,Rn	Subtracts the register from the accumulator with a borrow	1	1
SUBB A,direct	Subtracts the direct byte from the accumulator with a borrow	2	2
SUBB A,@Ri	Subtracts the indirect RAM from the accumulator with a borrow	1	2
SUBB A,#data	Subtracts the immediate data from the accumulator with a borrow	2	2
INC A	Increments the accumulator by 1	1	1
INC Rn	Increments the register by 1	1	2
INC Rx	Increments the direct byte by 1	2	3
INC @Ri	Increments the indirect RAM by 1	1	3
DEC A	Decrements the accumulator by 1	1	1
DEC Rn	Decrements the register by 1	1	1
DEC Rx	Decrements the direct byte by 1	1	2
DEC @Ri	Decrements the indirect RAM by 1	2	3
INC DPTR	Increments the Data Pointer by 1	1	3
MUL AB	Multiplies A and B	1	5
DIV AB	Divides A by B	1	5
DA A	Decimal adjustment of the accumulator according to BCD code	1	1

2. BRANCH INSTRUCTIONS

Mnemonic	**Description**	**Byte**	**Cycle**
ACALL addr11	Absolute subroutine call	2	6
LCALL addr16	Long subroutine call	3	6
RET	Returns from subroutine	1	4
RETI	Returns from interrupt subroutine	1	4
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump (from –128 to +127 locations relative to the following instruction)	2	3
JC rel	Jump if carry flag is set. Short jump.	2	3
JNC rel	Jump if carry flag is not set. Short jump.	2	3
JB bit,rel	Jump if direct bit is set. Short jump.	3	4
JBC bit,rel	Jump if direct bit is set and clears bit. Short jump.	3	4
JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if the accumulator is zero. Short jump.	2	3
JNZ rel	Jump if the accumulator is not zero. Short jump.	2	3
CJNE A,direct,rel	Compares direct byte to the accumulator and jumps if not equal. Short jump.	3	4
CJNE A,#data,rel	Compares immediate data to the accumulator and jumps if not equal. Short jump.	3	4
CJNE Rn,#data,rel	Compares immediate data to the register and jumps if not equal. Short jump.	3	4
CJNE @Ri,#data,rel	Compares immediate data to indirect register and jumps if not equal. Short jump.	3	4
DJNZ Rn,rel	Decrements register and jumps if not 0. Short jump.	2	3
DJNZ Rx,rel	Decrements direct byte and jump if not 0. Short jump.	3	4
NOP	No operation	1	1

3. DATA TRANSFER INSTRUCTIONS

Mnemonic	**Description**	**Byte**	**Cycle**
MOV A,Rn	Moves the register to the accumulator	1	1
MOV A,direct	Moves the direct byte to the accumulator	2	2
MOV A,@Ri	Moves the indirect RAM to the accumulator	1	2
MOV A,#data	Moves the immediate data to the accumulator	2	2
MOV Rn,A	Moves the accumulator to the register	1	2
MOV Rn,direct	Moves the direct byte to the register	2	4
MOV Rn,#data	Moves the immediate data to the register	2	2
MOV direct,A	Moves the accumulator to the direct byte	2	3
MOV direct,Rn	Moves the register to the direct byte	2	3
MOV direct,direct	Moves the direct byte to the direct byte	3	4
MOV direct,@Ri	Moves the indirect RAM to the direct byte	2	4
MOV direct,#data	Moves the immediate data to the direct byte	3	3
MOV @Ri,A	Moves the accumulator to the indirect RAM	1	3
MOV @Ri,direct	Moves the direct byte to the indirect RAM	2	5
MOV @Ri,#data	Moves the immediate data to the indirect RAM	2	3
MOV DPTR,#data	Moves a 16-bit data to the data pointer	3	3
MOVC A,@A+DPTR	Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)	1	3
MOVC A,@A+PC	Moves the code byte relative to the PC to the accumulator (address=A+PC)	1	3
MOVX A,@Ri	Moves the external RAM (8-bit address) to the accumulator	1	3-10
MOVX A,@DPTR	Moves the external RAM (16-bit address) to the accumulator	1	3-10
MOVX @Ri,A	Moves the accumulator to the external RAM (8-bit address)	1	4-11
MOVX @DPTR,A	Moves the accumulator to the external RAM (16-bit address)	1	4-11
PUSH direct	Pushes the direct byte onto the stack	2	4
POP direct	Pops the direct byte from the stack/td>	2	3
XCH A,Rn	Exchanges the register with the accumulator	1	2
XCH A,direct	Exchanges the direct byte with the accumulator	2	3
XCH A,@Ri	Exchanges the indirect RAM with the accumulator	1	3
XCHD A,@Ri	Exchanges the low-order nibble indirect RAM with the accumulator	1	3

4. LOGIC INSTRUCTIONS

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

5. BIT-ORIENTED INSTRUCTIONS

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

Postscript

The materials included in this handout is of practical nature. Anyone can use any part of it without restriction.

For any query feel free to drop mail at *sadananduk13@gmail.com*.
Good luck!