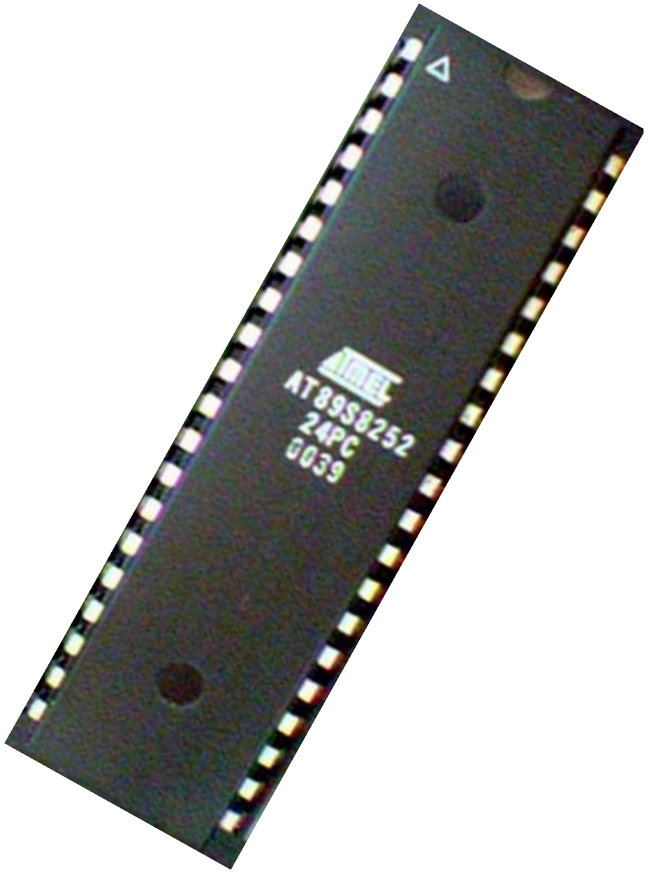
THE 8051 ARCHITECTURE

By Dr. Naveen B

Contents:



8051 features

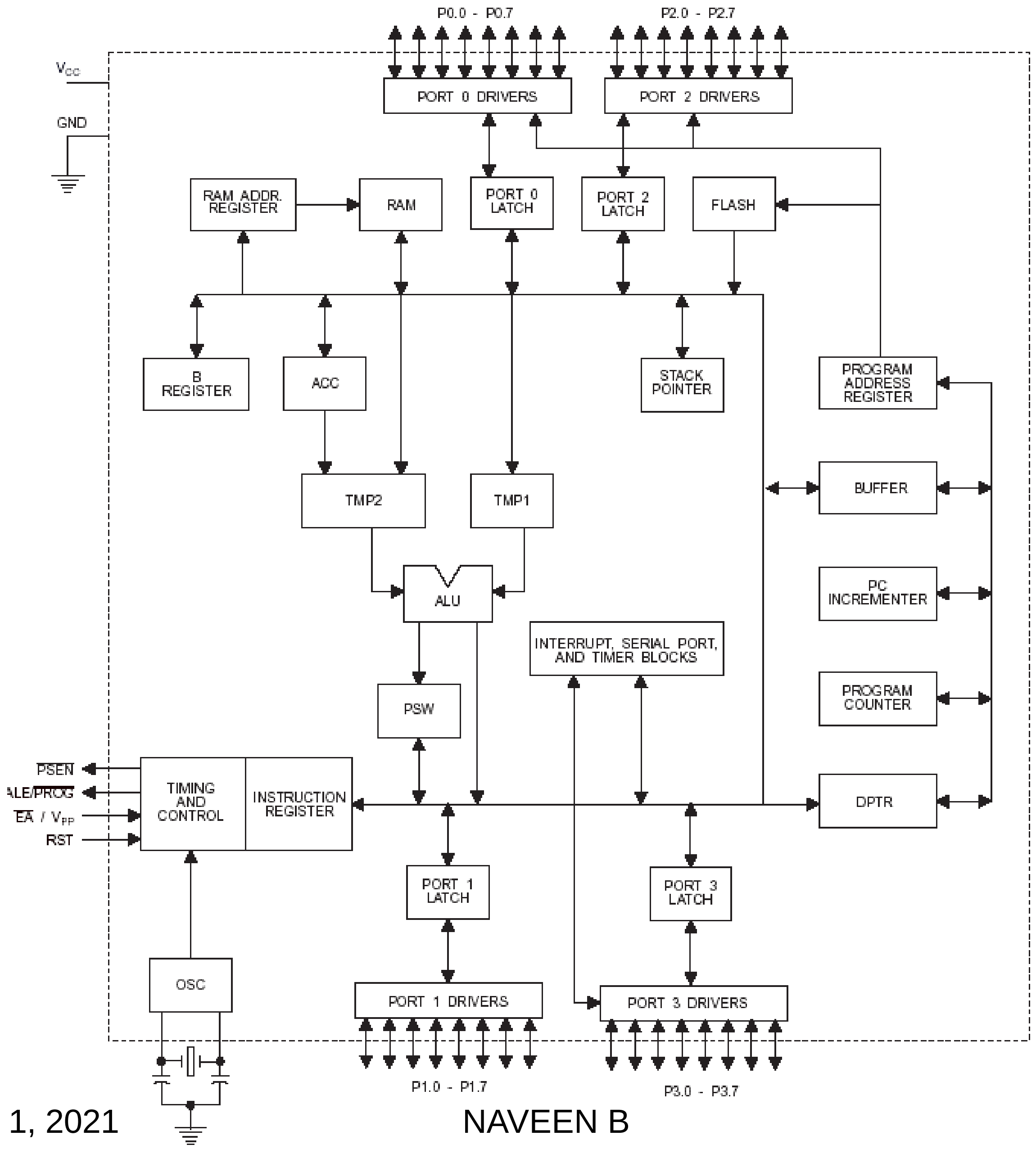
- 8 bit CPU with reg. A & B
- 16 bit PC & Data pointer (DPTR)
- 8 bit PSW, 8 bit SP, onchip oscillator & clock circuits
- 64k ROM, in which 4K is on-chip, full duplex serial SBUF
- External RAM of 64K bytes & Internal RAM of 128 bytes :

  - 4 reg. banks each containing 8 registers

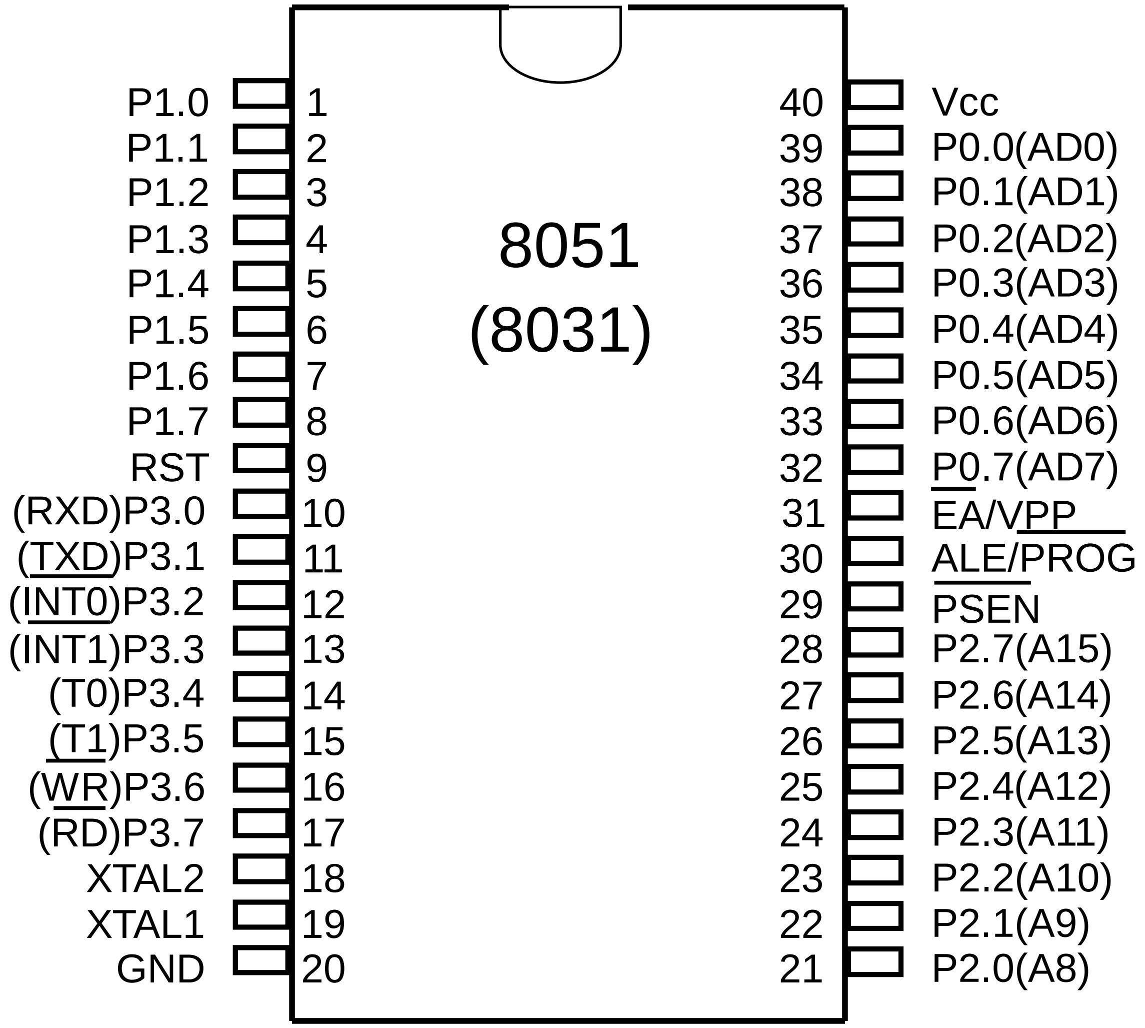
  - 80 bytes of general purpose data memory

  -16 bytes which may be addressed at the bit level
- 32 I/O pins arranged as 4 ports
- Two 16 bit timer counters, T0 & T1
- 2 external & 3 internal interrupt sources
- Control registers: TCON, TMOD, SCON, PCON, IP, IE

Block Diagram



Pin Description of the 8051



Pins of 8051（1/4）

- Vcc（pin 40）：
  - Vcc provides supply voltage to the chip.
  - The voltage source is +5V.
- GND（pin 20）：ground
- XTAL1 and XTAL2（pins 19,18）：
  - These 2 pins provide external clock.
  - Way 1：using a quartz crystal oscillator
  - Way 2：using a TTL oscillator

Pins of 8051（2/4）

- RST（pin 9）：reset
  - It is an input pin and is active high（normally low）.
    - The high pulse must be high at least 2 machine cycles.
  - It is a power-on reset.
    - Upon applying a high pulse to RST, the microcontroller will reset and all values in registers will be lost.
    - Reset values of some 8051 registers
  - Way 1：Power-on reset circuit
  - Way 2：Power-on reset with debounce

Pins of 8051（3/4）

- /EA（pin 31）：external access
  - There is no on-chip ROM in 8031 and 8032 .
  - The /EA pin is connected to GND to indicate the code is stored externally.
  - For 8051, /EA pin is connected to Vcc.
  - “/” means active low.

- /PSEN（pin 29）：program store enable
  - This is an output pin and is connected to the OE pin of the ROM.

Pins of 8051（4/4）

- ALE（pin 30）：address latch enable
  - It is an output pin and is active high.
  - 8051 port 0 provides both address and data.
  - The ALE pin is used for de-multiplexing the address and data by connecting to the G pin of the 74LS373 latch.
  - /PSEN ＆ ALE are used for external ROM.

- I/O port pins
  - The four ports P0, P1, P2, and P3.
  - Each port uses 8 pins.
  - All I/O pins are bi-directional.

Figure 4-2 (a). XTAL Connection to 8051

- Using a quartz crystal oscillator
- We can observe the frequency on the XTAL2 pin.

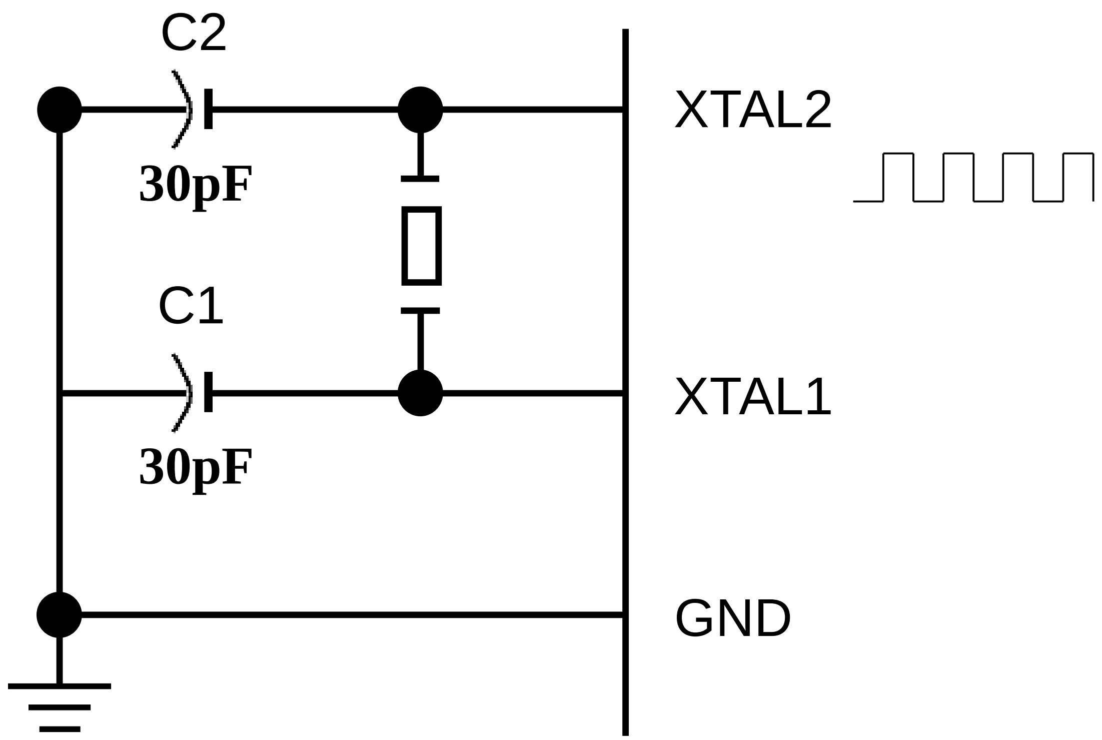
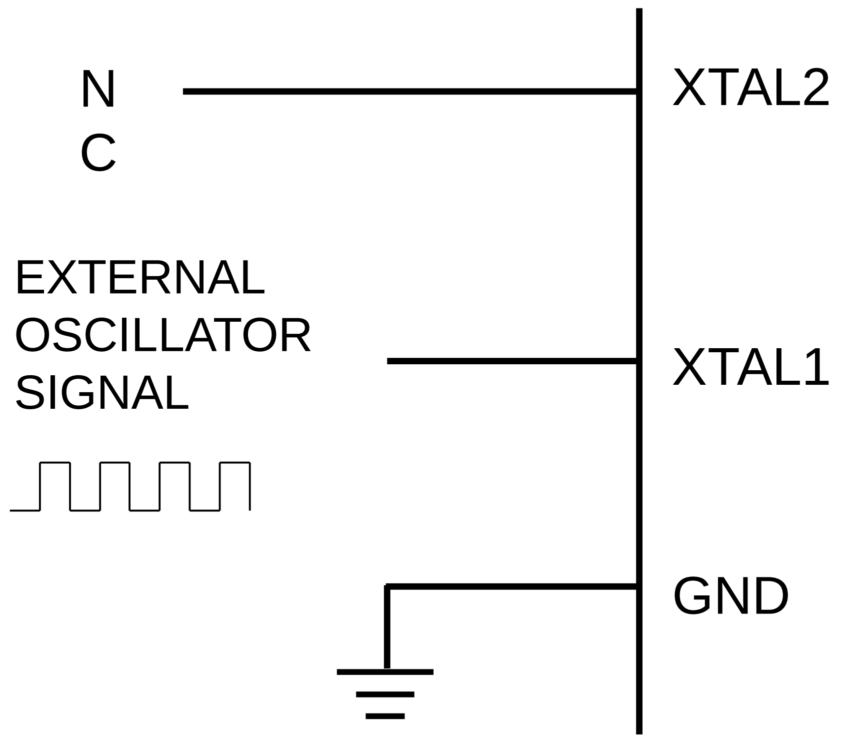


Figure 4-2 (b). XTAL Connection to an External Clock Source

- Using a TTL oscillator
- XTAL2 is unconnected.



RESET Value of Some 8051 Registers:

Register	Reset Value
PC	0000
ACC	0000
B	0000
PSW	0000
SP	0007
DPTR	0000

RAM are all zero.

Figure 4-3 (a). Power-On RESET Circuit

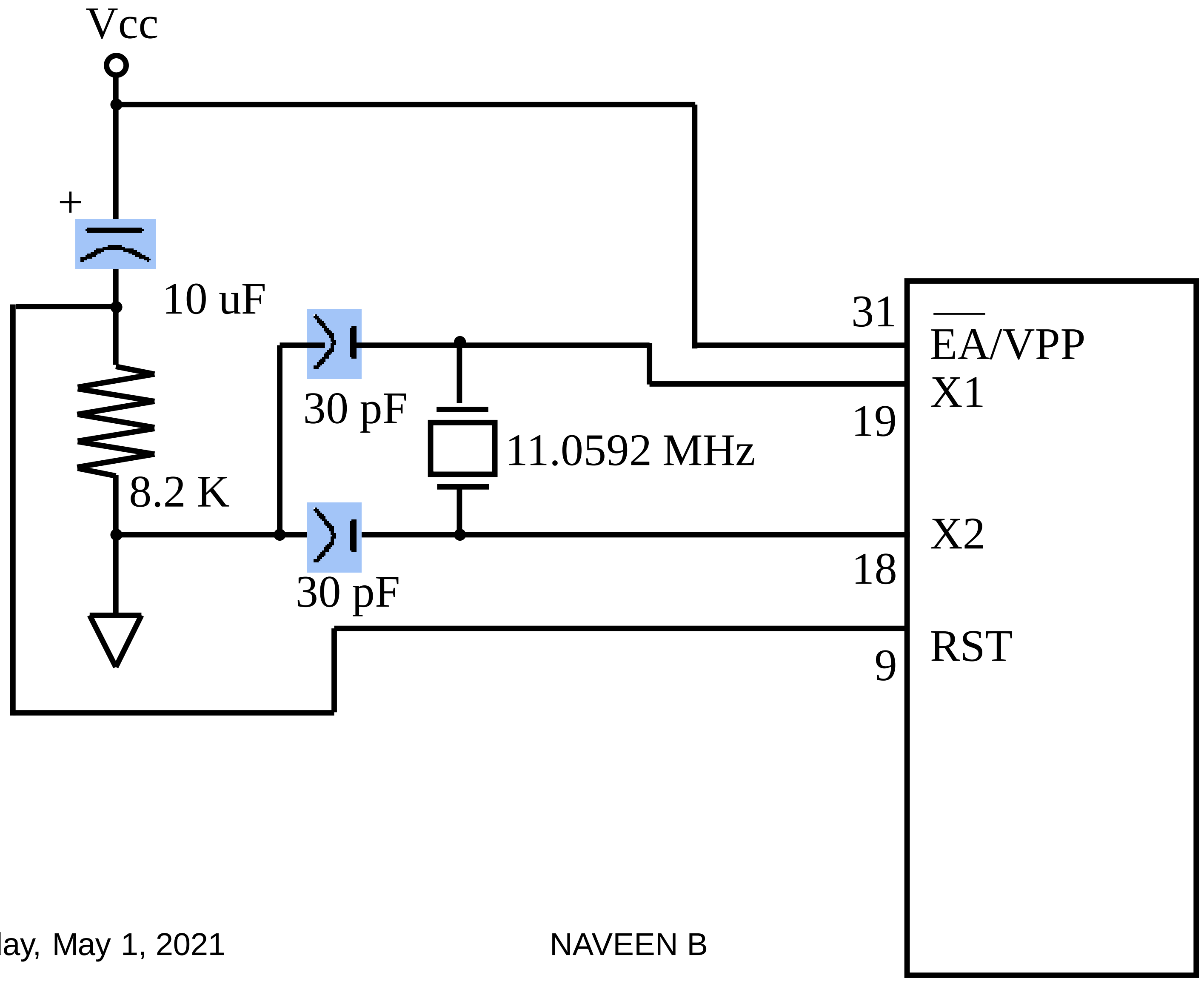
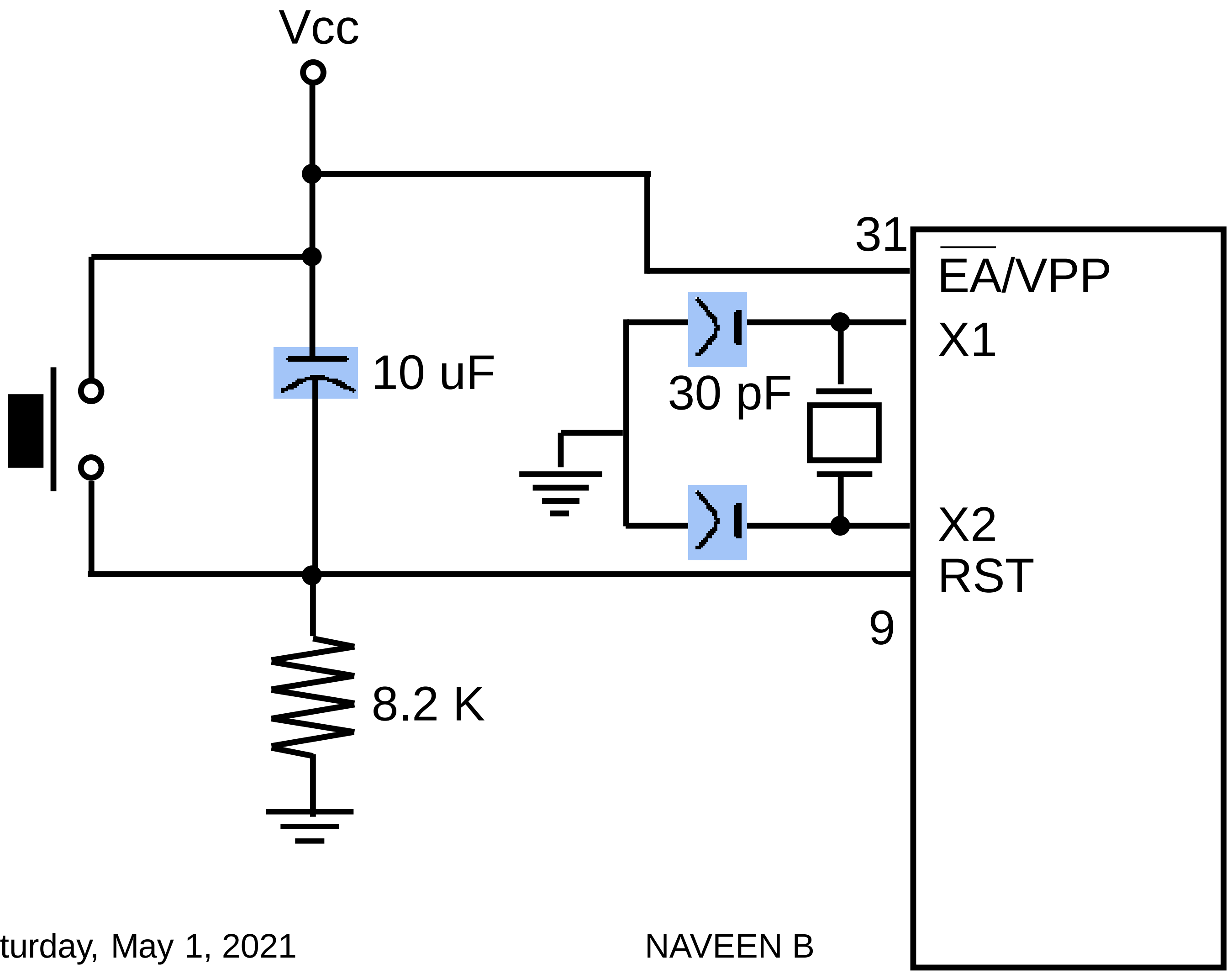


Figure 4-3 (b). Power-On RESET with Debounce

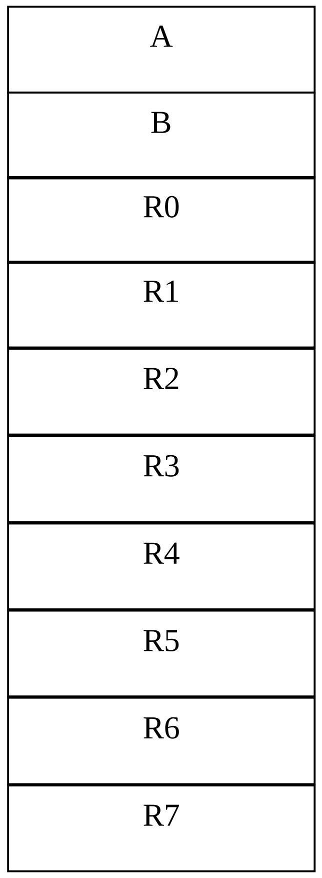


Pins of I/O Port

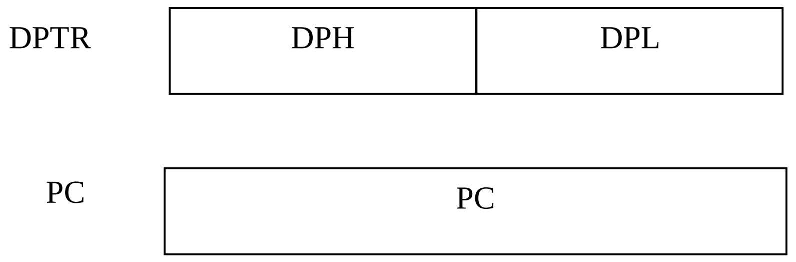
- The 8051 has four I/O ports
  - Port 0 （pins 32-39） ：P0（P0.0～P0.7）
  - Port 1（pins 1-8） ：P1（P1.0～P1.7）
  - Port 2（pins 21-28）：P2（P2.0～P2.7）
  - Port 3（pins 10-17）：P3（P3.0～P3.7）
  - Each port has 8 pins.
    - Named P0.X （X=0,1,...,7）, P1.X, P2.X, P3.X
    - Ex：P0.0 is the bit 0（LSB）of P0
    - Ex：P0.7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

Registers

Totally 34 GPRs ie. A & B registers along with 4 banks – each bank has 8 registers: R0 – R7. Other registers are PC, DPTR & SP



Some 8-bit Registers of the 8051



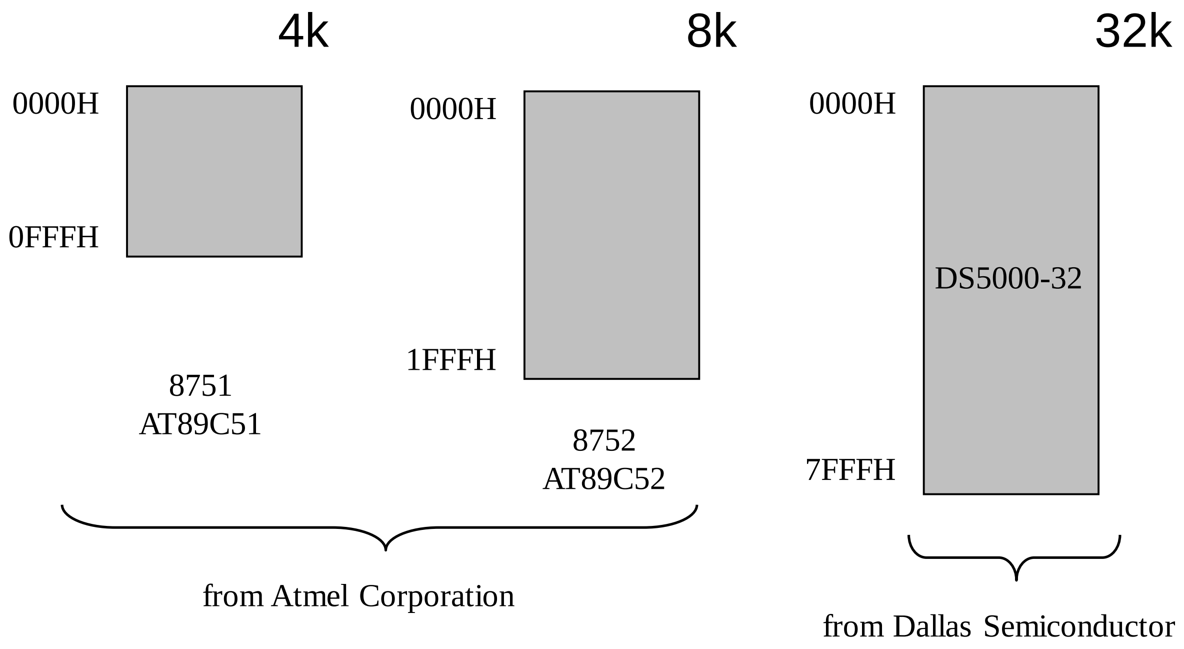
Some 8051 16-bit Register

A and B registers

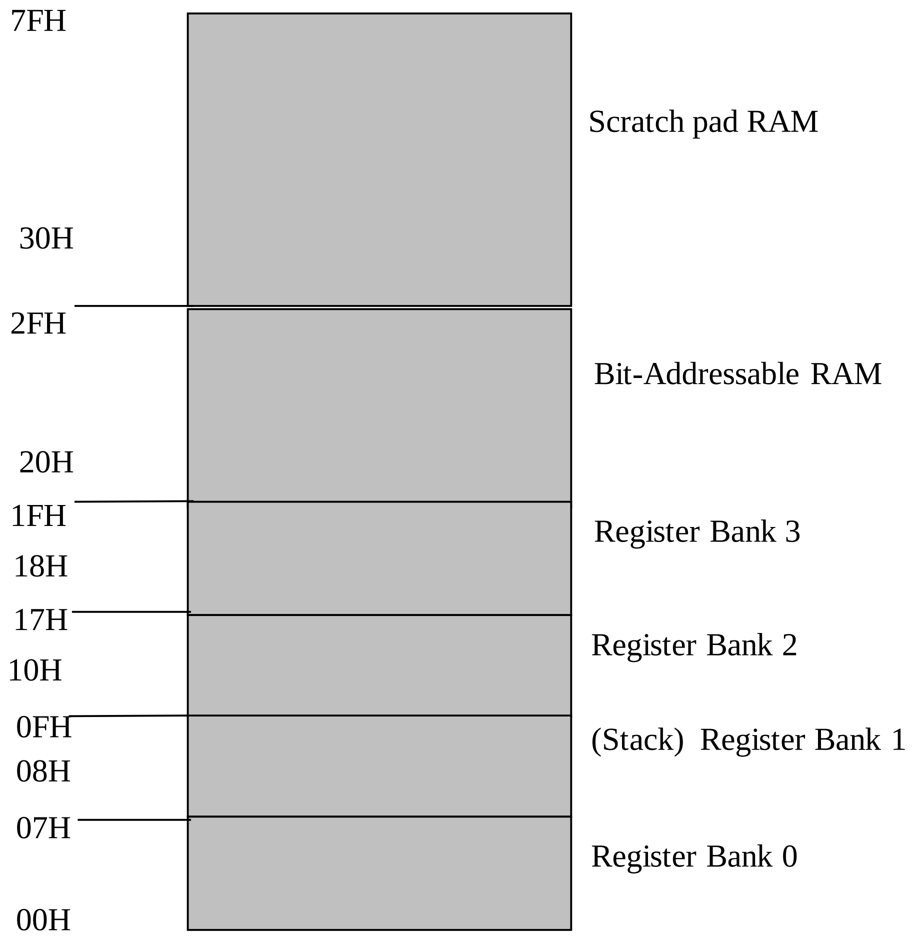
- Holds the results of math & logical operations
- ‘ A’ register is also used for data transfers between 8051 & any external memory
- ‘B’ register is used with ‘A’ register for multiplication & division operations.

Memory mapping in 8051

- **ROM memory map in 8051 family**



- **RAM memory space allocation in the 8051**

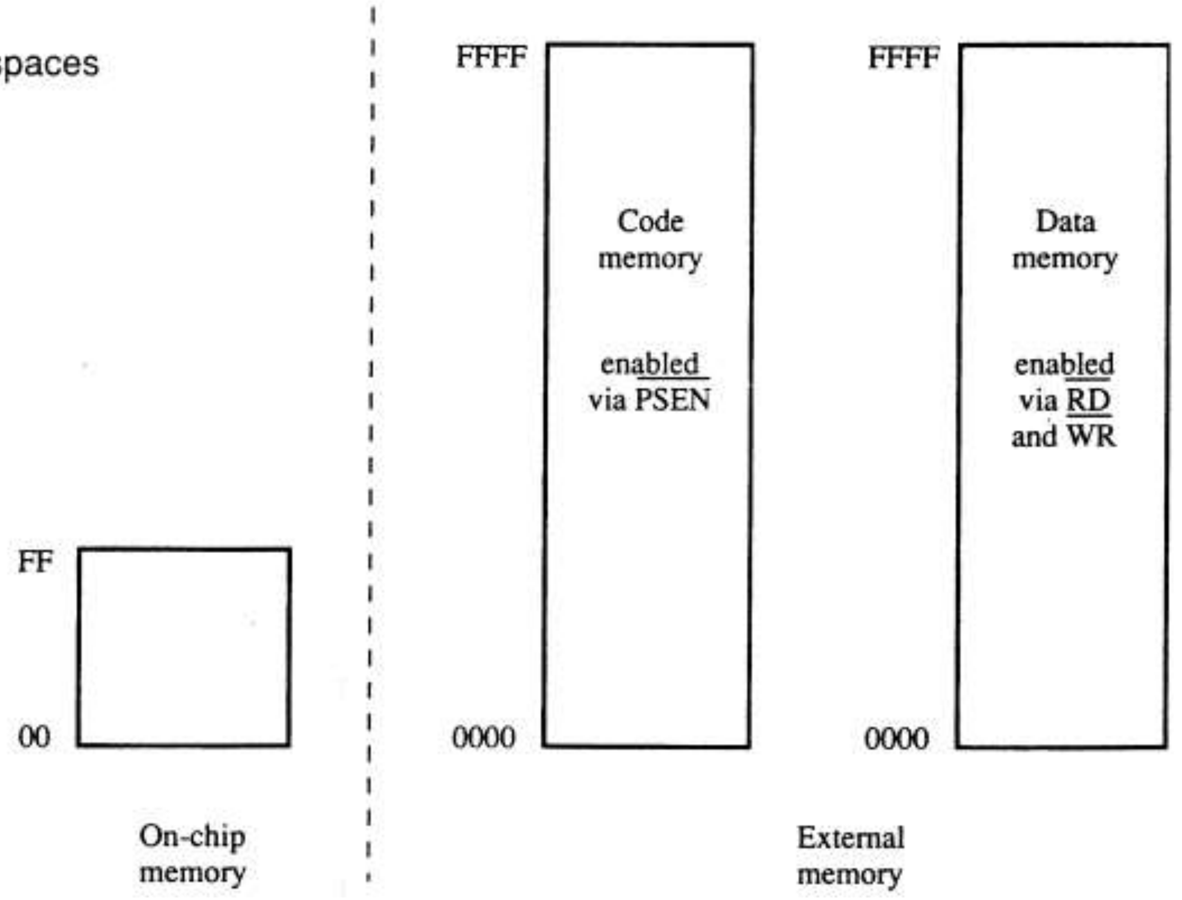


A Quick Review of USER's space…….

- 00 – 1F : 4 Banks x 8 = 32 registers
  - ONE bank at a time (RS1-RS0)
- 20 – 2F : 16 more locations = 16 Bytes
  - Also BIT addressable (00-7F address for the 128 bits)
- 30 – 7F Scratch Pad (Store-Read-Write-Modify data)
  - General Purpose RAM (80 bytes)
- 80 – FF : Special Purpose CPU Area
  - Also Contains the SFRs

Memory Space

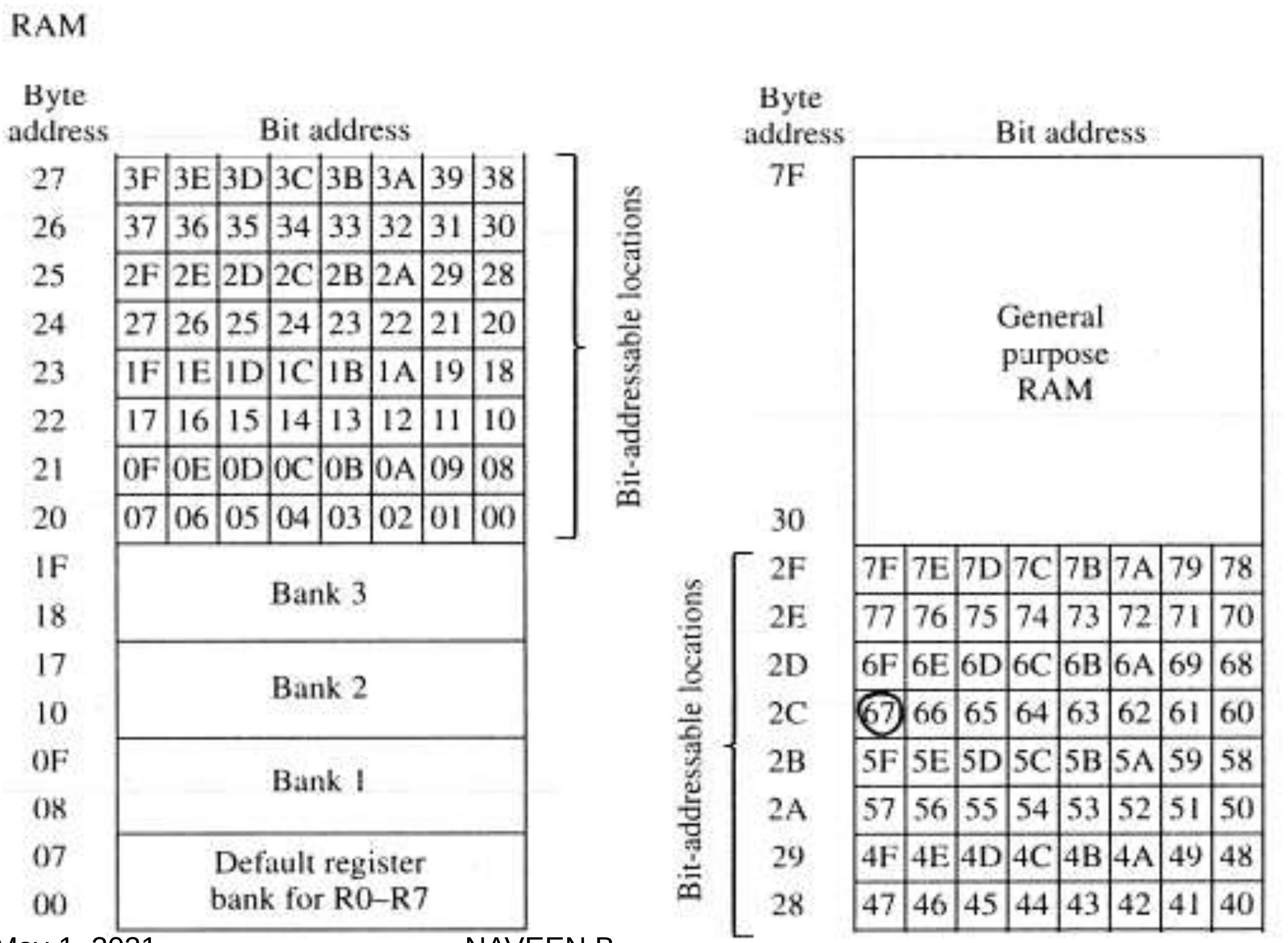
FIGURE 2–5
Summary of the 8031 memory spaces


FF
00
On-chip memory
FFFF
0000
Code memory
enabled via $\overline{PSEN}$
FFFF
0000
Data memory
enabled via $\overline{RD}$ and $\overline{WR}$
External memory

Bit Addressable RAM

Summary of the 8051 on-chip data memory

(RAM)



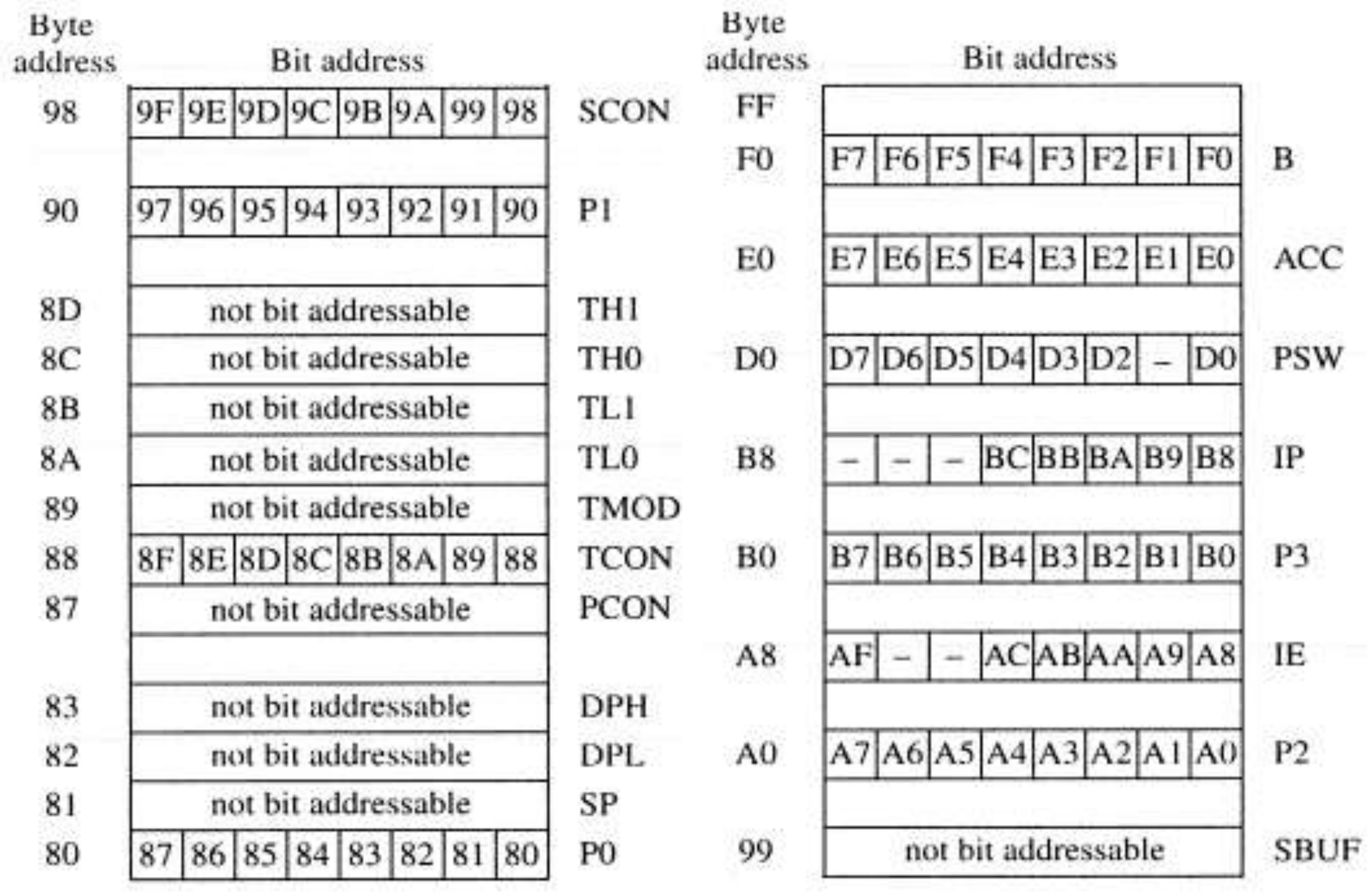
Byte address	Bit address							
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F–18	Bank 3							
17–10	Bank 2							
0F–08	Bank 1							
07–00	Default register bank for R0–R7							

Byte address	Bit address							
7F–30	General purpose 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

Bit Addressable RAM

(Special Function Registers)

Summary of the 8051 on-chip data memory



Byte address	Bit address								
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

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	–	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

Register Banks

- Four banks of 8 bit-sized registers, **R0** to **R7**
- Addresses are :

18 - 1F	for bank 3
10 - 17	for bank 2
08 - 0F	for bank 1
00 - 07	for bank 0 (default)

- Active bank selected by bits [**RS1**, **RS0**] in PSW.
- Permits fast "context switching" in interrupt service routines (ISR).

Program Status Word (PSW)

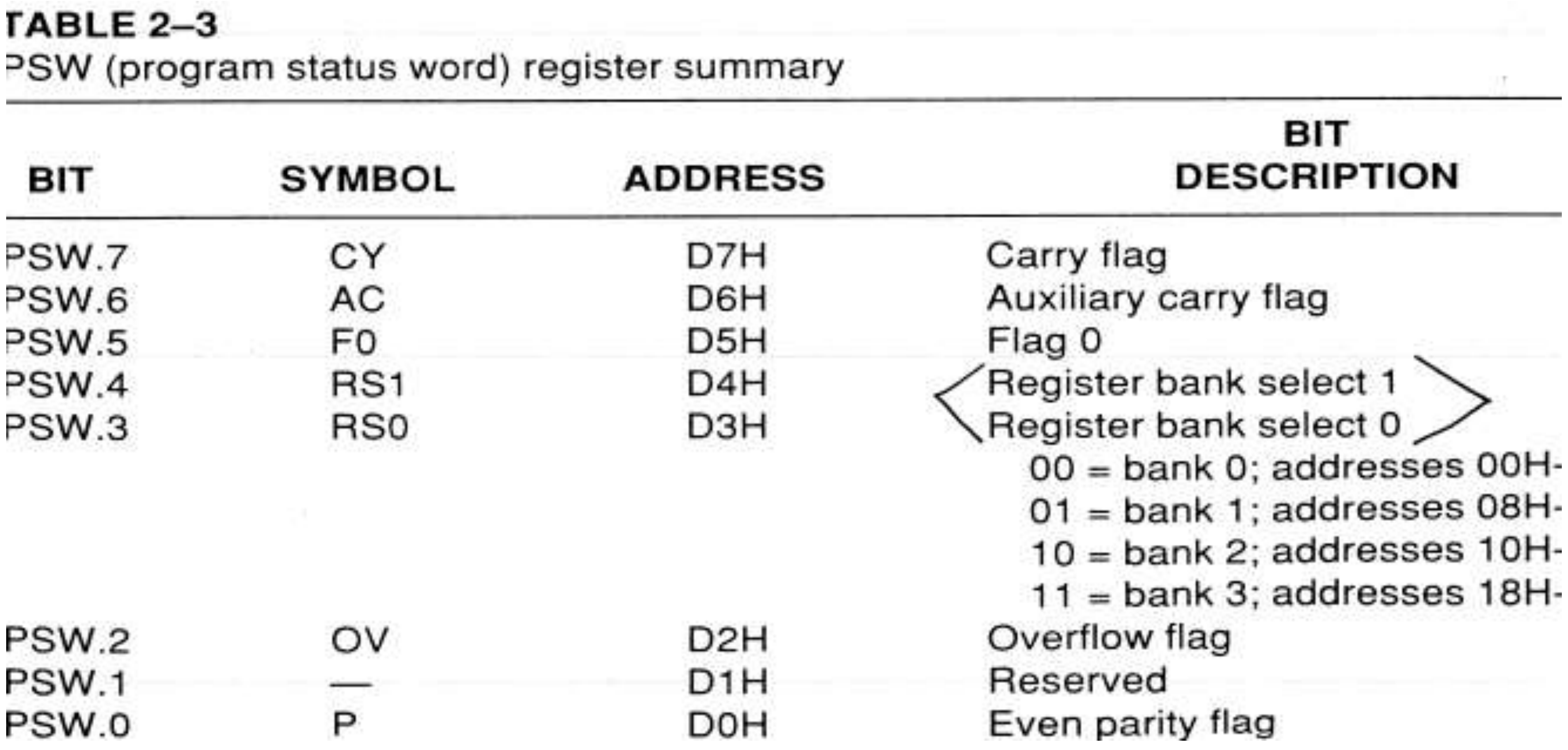


TABLE 2–3
PSW (program status word) register summary

BIT	SYMBOL	ADDRESS	BIT DESCRIPTION
PSW.7	CY	D7H	Carry flag
PSW.6	AC	D6H	Auxiliary carry flag
PSW.5	F0	D5H	Flag 0
PSW.4	RS1	D4H	Register bank select 1
PSW.3	RS0	D3H	Register bank select 0
			00 = bank 0; addresses 00H-
			01 = bank 1; addresses 08H-
			10 = bank 2; addresses 10H-
			11 = bank 3; addresses 18H-
PSW.2	OV	D2H	Overflow flag
PSW.1	—	D1H	Reserved
PSW.0	P	D0H	Even parity flag

8051 Flag bits and the PSW register

- PSW Register

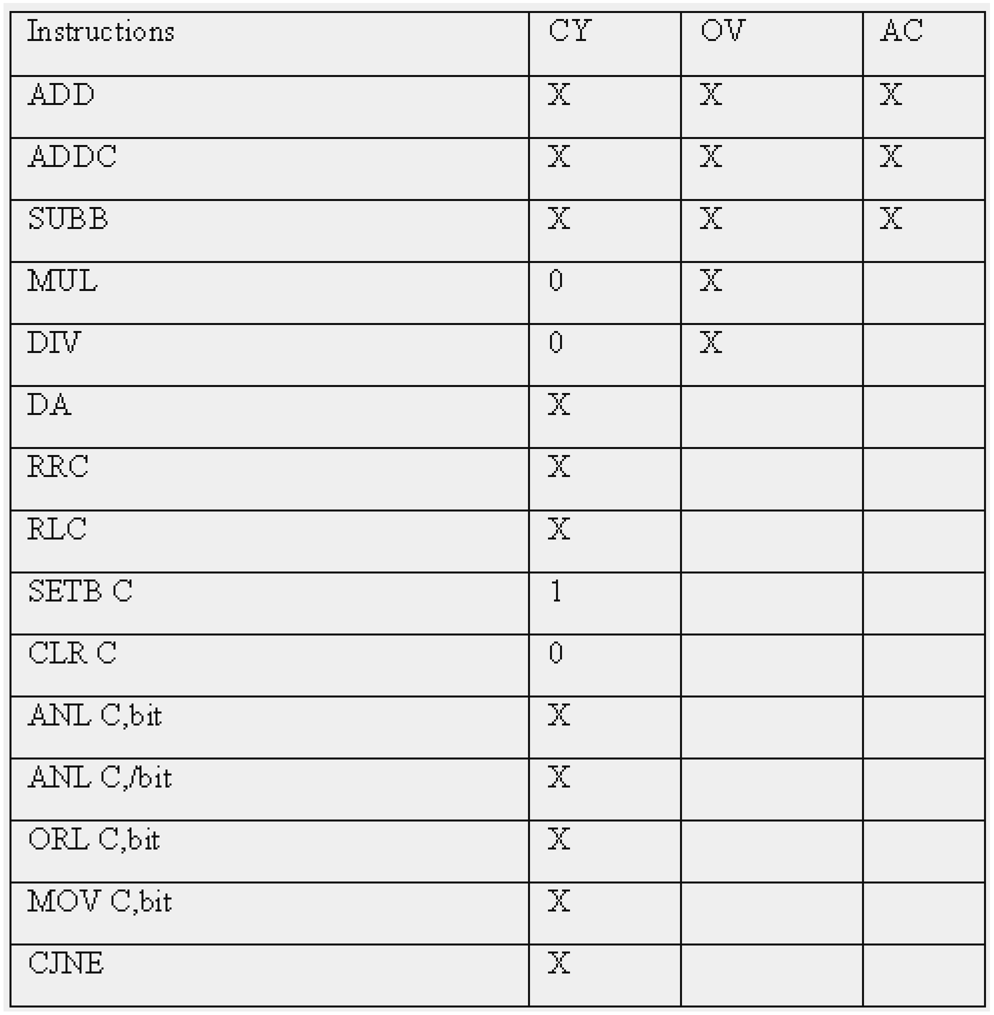
CY	AC	F0	RS1	RS0	OV	--	P

Carry flag	PSW.7	**CY**
Auxiliary carry flag	PSW.6	**AC**
Available to the user for general purpose	PSW.5	**--**
Register Bank selector bit 1	PSW.4	**RS1**
Register Bank selector bit 0	PSW.3	**RS0**
Overflow flag	PSW.2	**OV**
User define bit	PSW.1	**--**
Parity flag Set/Reset odd/even parity	PSW.0	**P**

For odd Parity, P=1, ;

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

Instructions that Affect Flag Bits:



Instructions	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RRC	X		
RLC	X		
SETB C	1		
CLR C	0		
ANL C,bit	X		
ANL C,/bit	X		
ORL C,bit	X		
MOV C,bit	X		
CJNE	X		

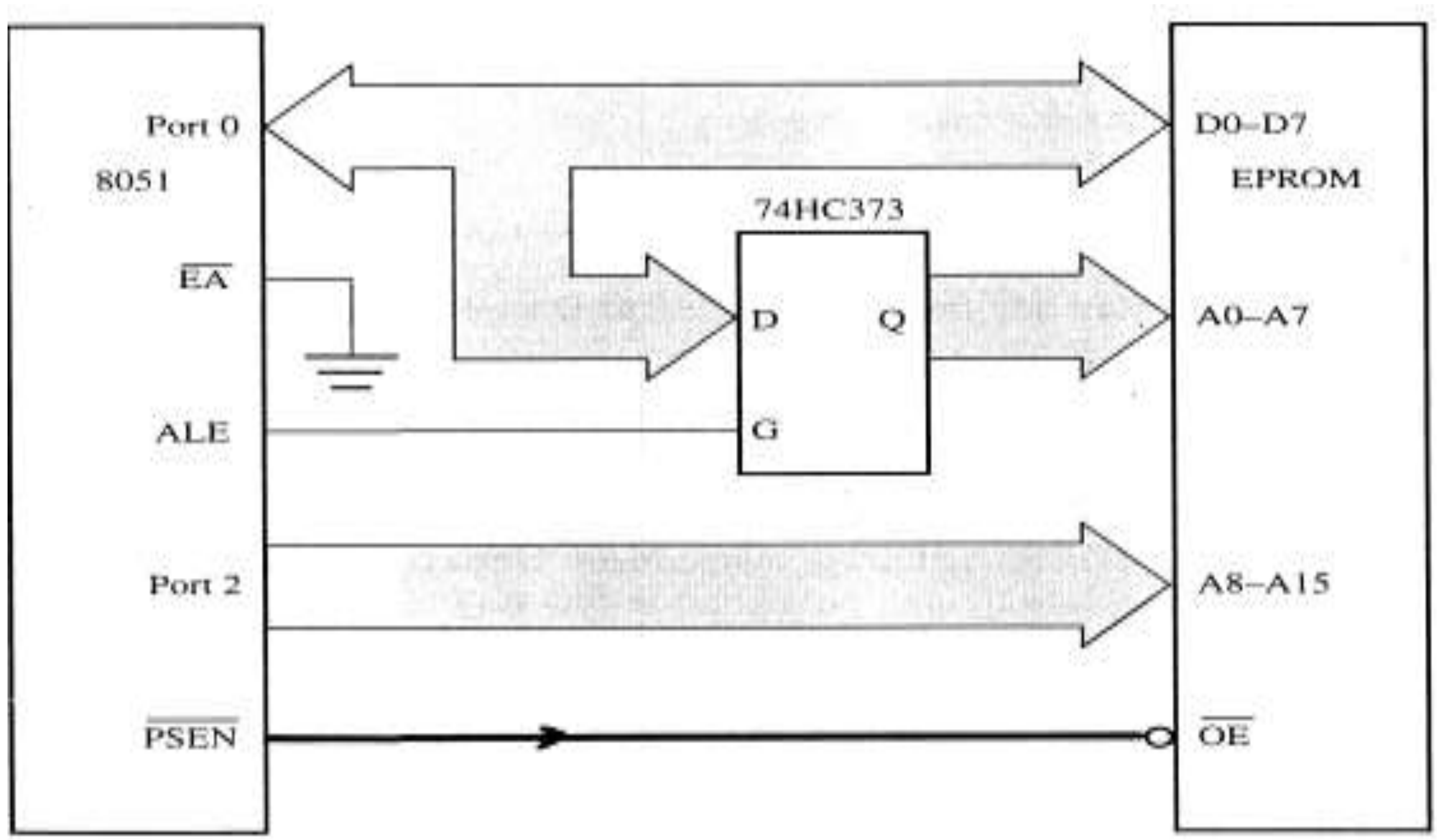
Note: X can be 0 or 1

```
Example:
MOV       A,#88H
ADD       A,#93H

     88              10001000
    +93             +10010011
     ----            --------------
     11B            10 0011011
CY=1      AC=0      P=0    OV=1
```

```
Example:
MOV       A,#38H
ADD       A,#2FH

     38              00111000
    +2F             +00101111
     ----            --------------
     67              01100111
CY=0      AC=1      P=1     OV=0
```

```
Example:
MOV       A,#9CH
ADD       A,#64H

     9C              10011100
    +64             +01100100
     ----            --------------
     100            1 00000000

CY=1      AC=1      P=0     OV=0
```

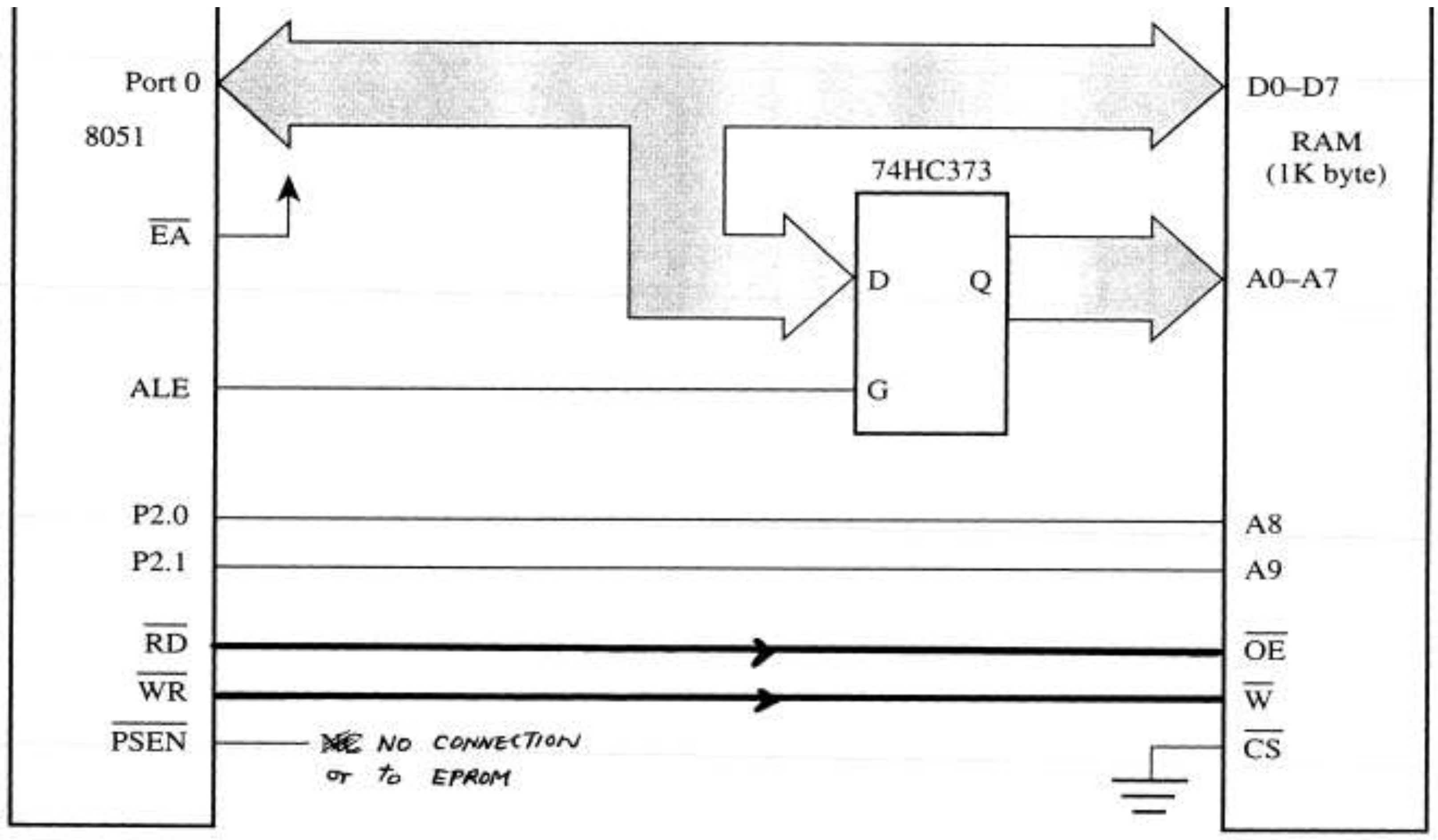
Accessing External Code Memory



Accessing External Data Memory

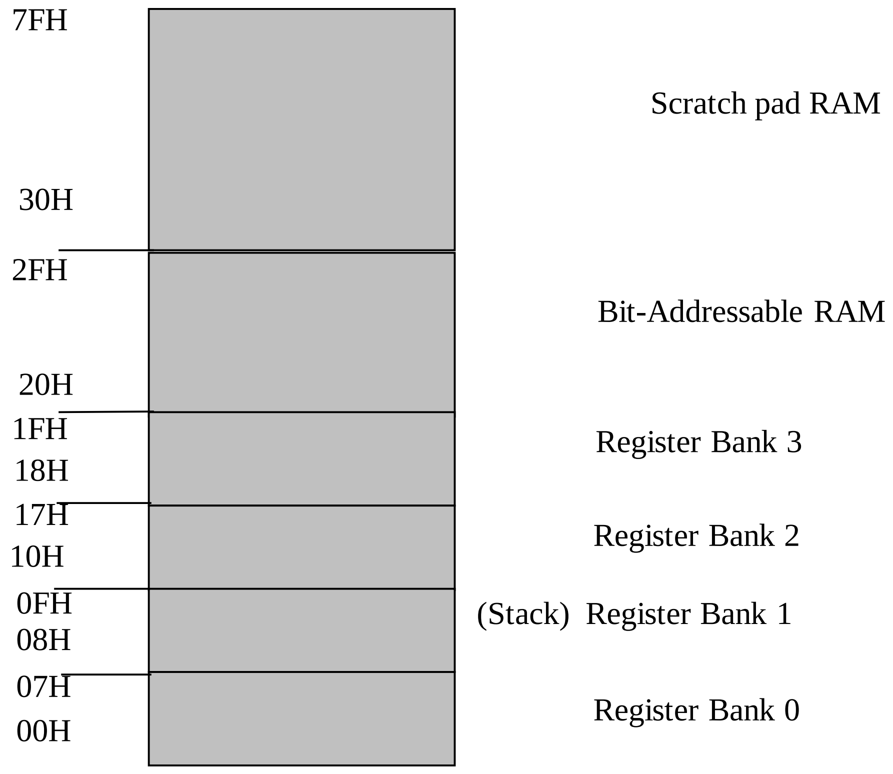
Figure 2-11

Interface to 1K RAM



Stack in the 8051

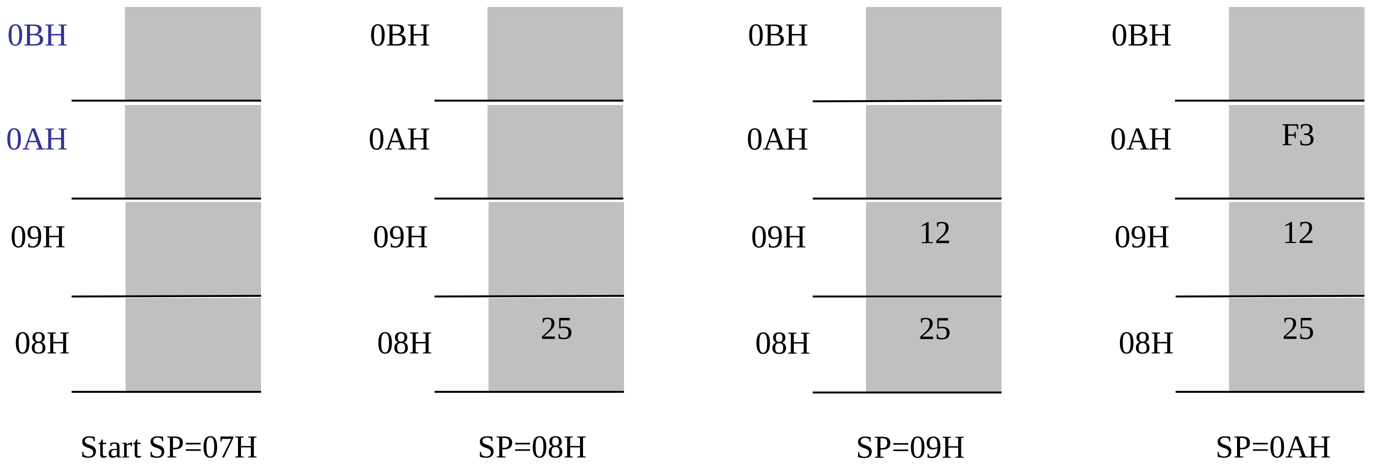
- The register used to access the stack is called **SP** (stack pointer) register.
- The stack pointer in the 8051 is only 8 bits wide, which means that it can take value 00 to 7FH.
- <u>When 8051 powered up, the SP register contains value 07.</u>



Example:

```
        MOV     R6,#25H
        MOV     R1,#12H
        MOV     R4,#0F3H

        PUSH    6
        PUSH    1
        PUSH    4
```

***Operand is 8 bit & Direct addressing only**



PC (Program Counter)

- 16 bit registers used to hold the address of a byte in memory
- Program instructions are fetched by PC
- On chip ROM addresses 0000h to 0FFFh & external addresses that exceed 0FFFh
- PC is the only reg. that does not have Internal address.

DPTR (Data pointer)

- Two 8 bit regs. DPH & DPL
- Used to furnish memory addresses for internal & external code and external Data access.
- DPTR does not have a single internal address
- DPH & DPL each assigned a address

I/O Port Circuitry

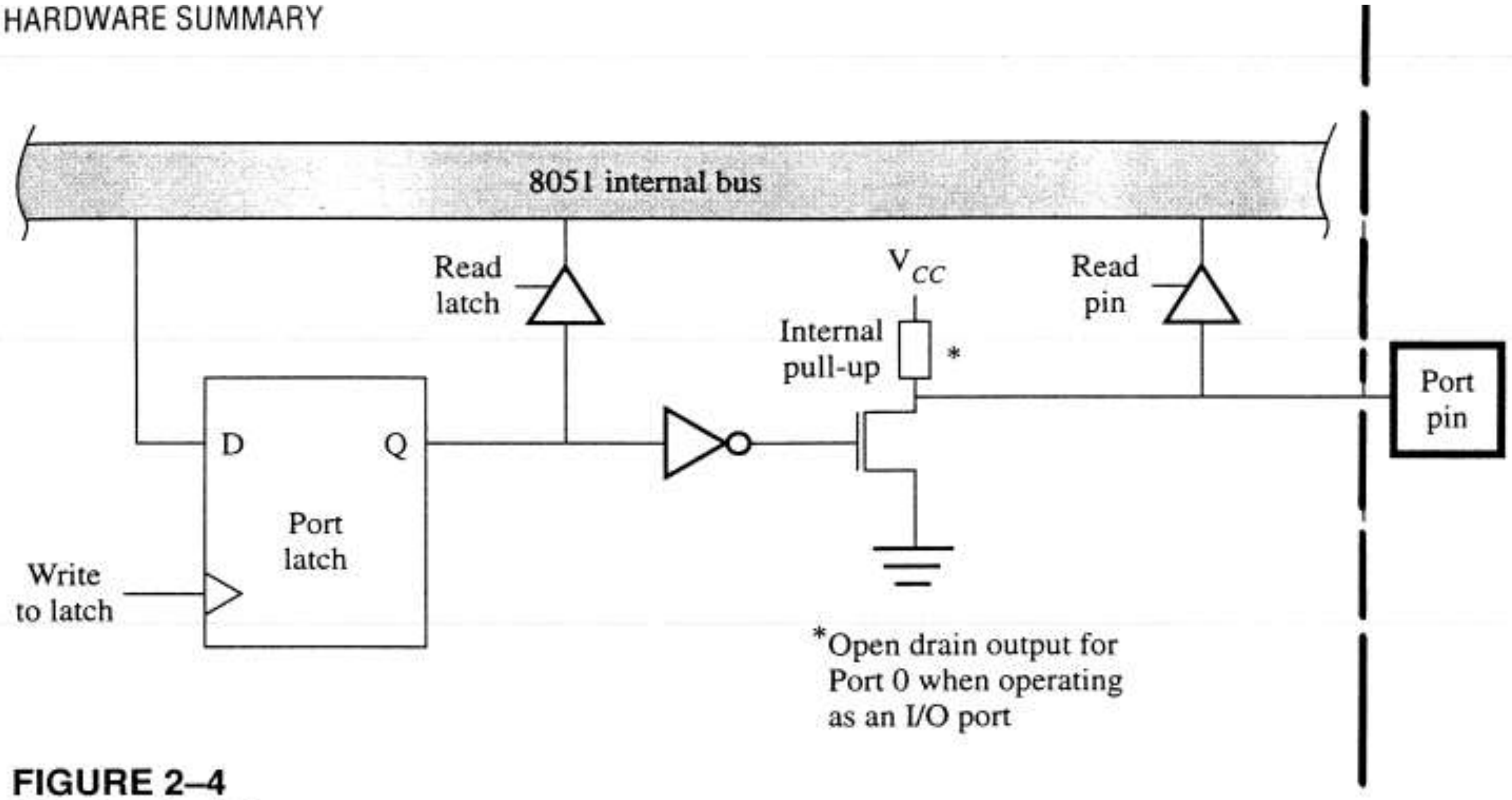
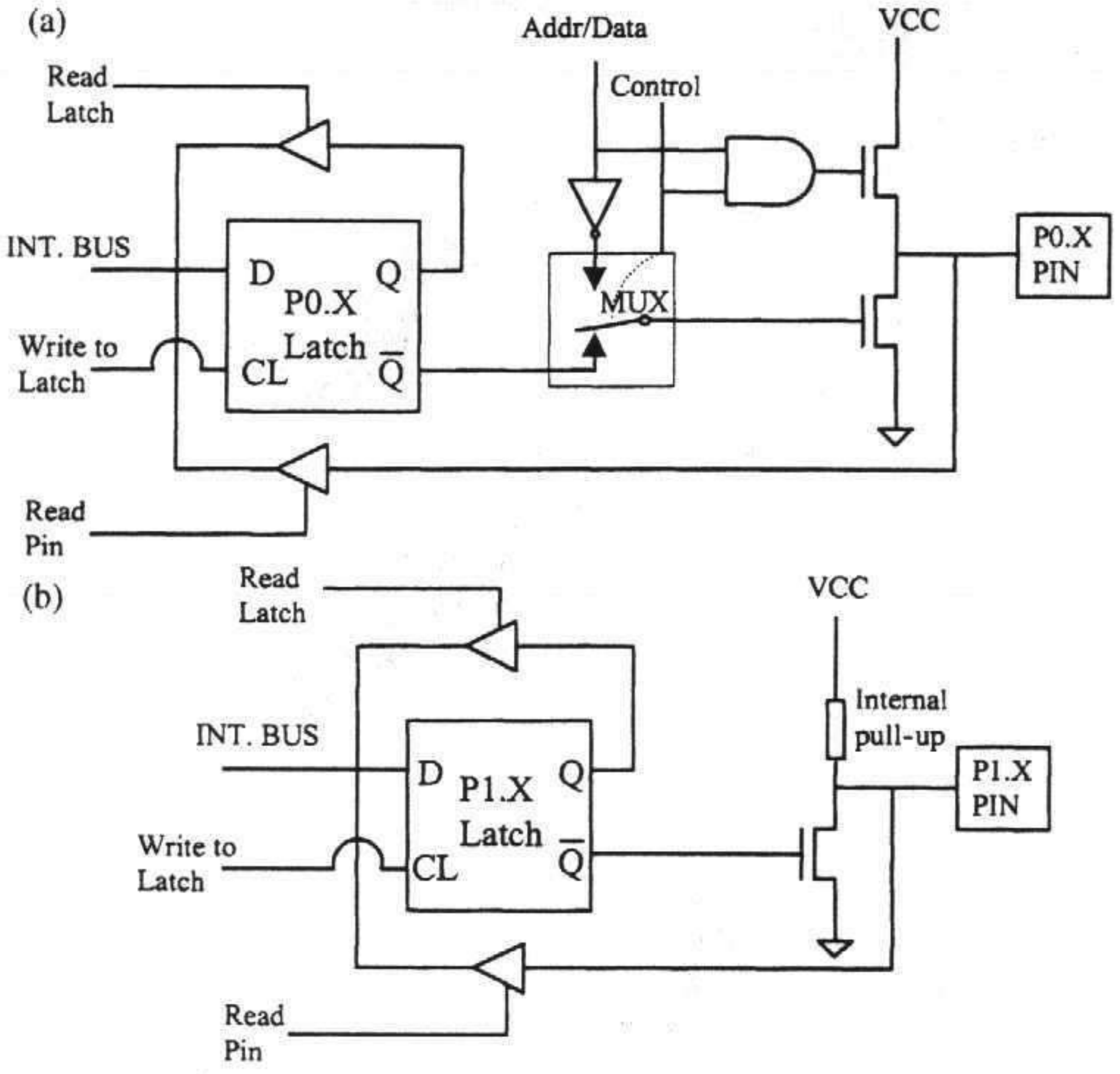
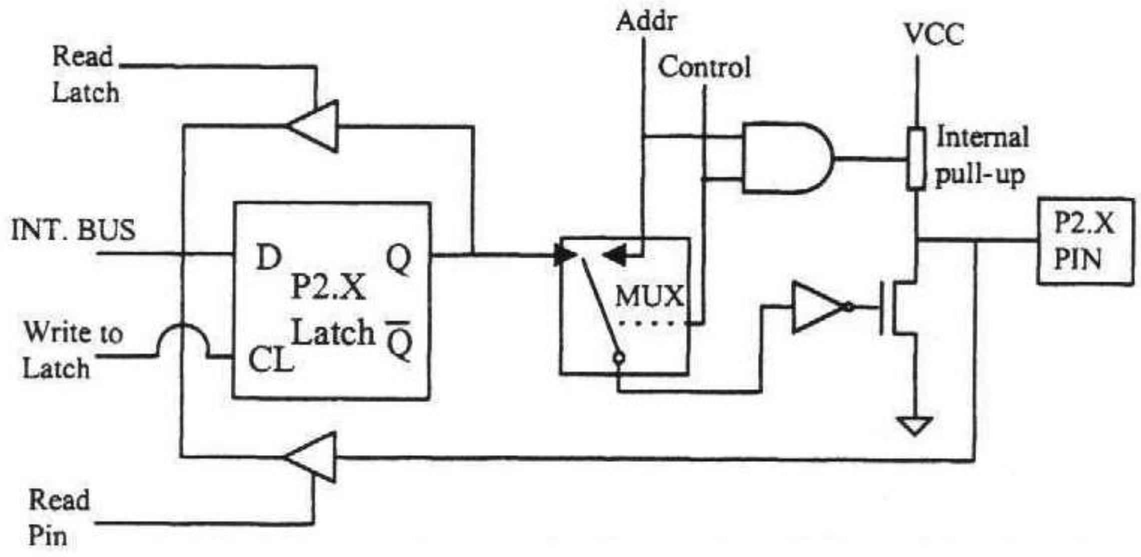


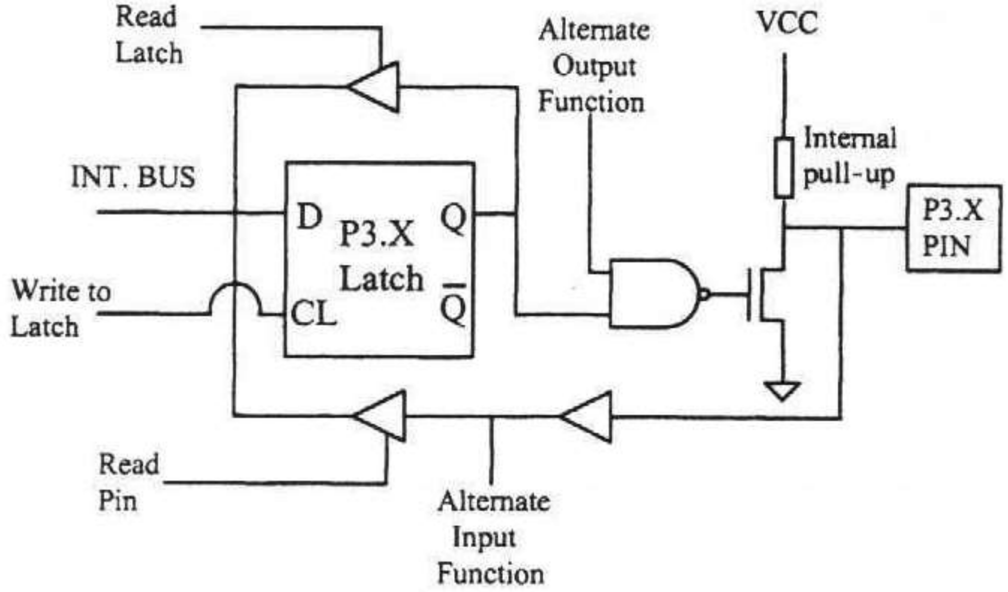
FIGURE 2–4
Circuitry for I/O ports


(a)
Read Latch
INT. BUS
Write to Latch
D
P0.X Latch
Q
CL
$\overline{Q}$
Read Pin
Addr/Data
Control
MUX
VCC
P0.X PIN
(b)
Read Latch
INT. BUS
Write to Latch
D
P1.X Latch
Q
CL
$\overline{Q}$
Read Pin
VCC
Internal pull-up
P1.X PIN

(c)


Read Latch
Addr
Control
VCC
Internal pull-up
INT. BUS
D
P2.X Latch
Q
$\overline{Q}$
MUX
P2.X PIN
Write to Latch
CL
Read Pin

(d)


Read Latch
Alternate Output Function
VCC
Internal pull-up
INT. BUS
D
P3.X Latch
Q
$\overline{Q}$
P3.X PIN
Write to Latch
CL
Read Pin
Alternate Input Function

Alternate Pin-functions

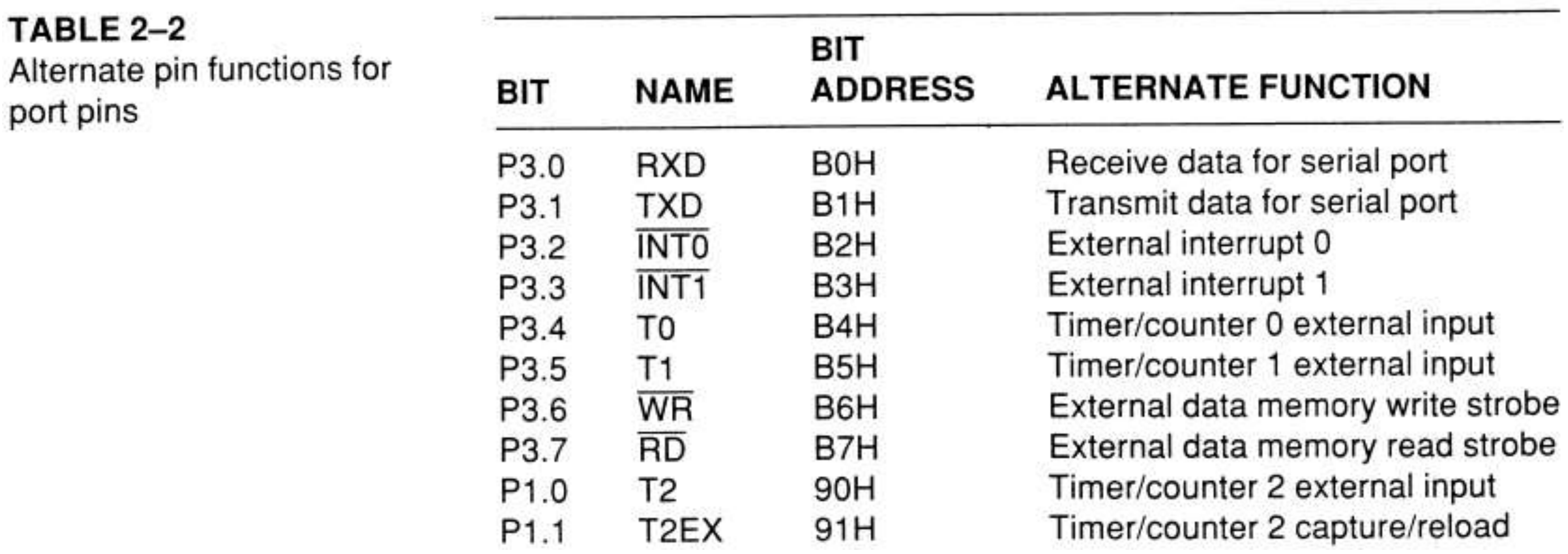


TABLE 2–2
Alternate pin functions for port pins

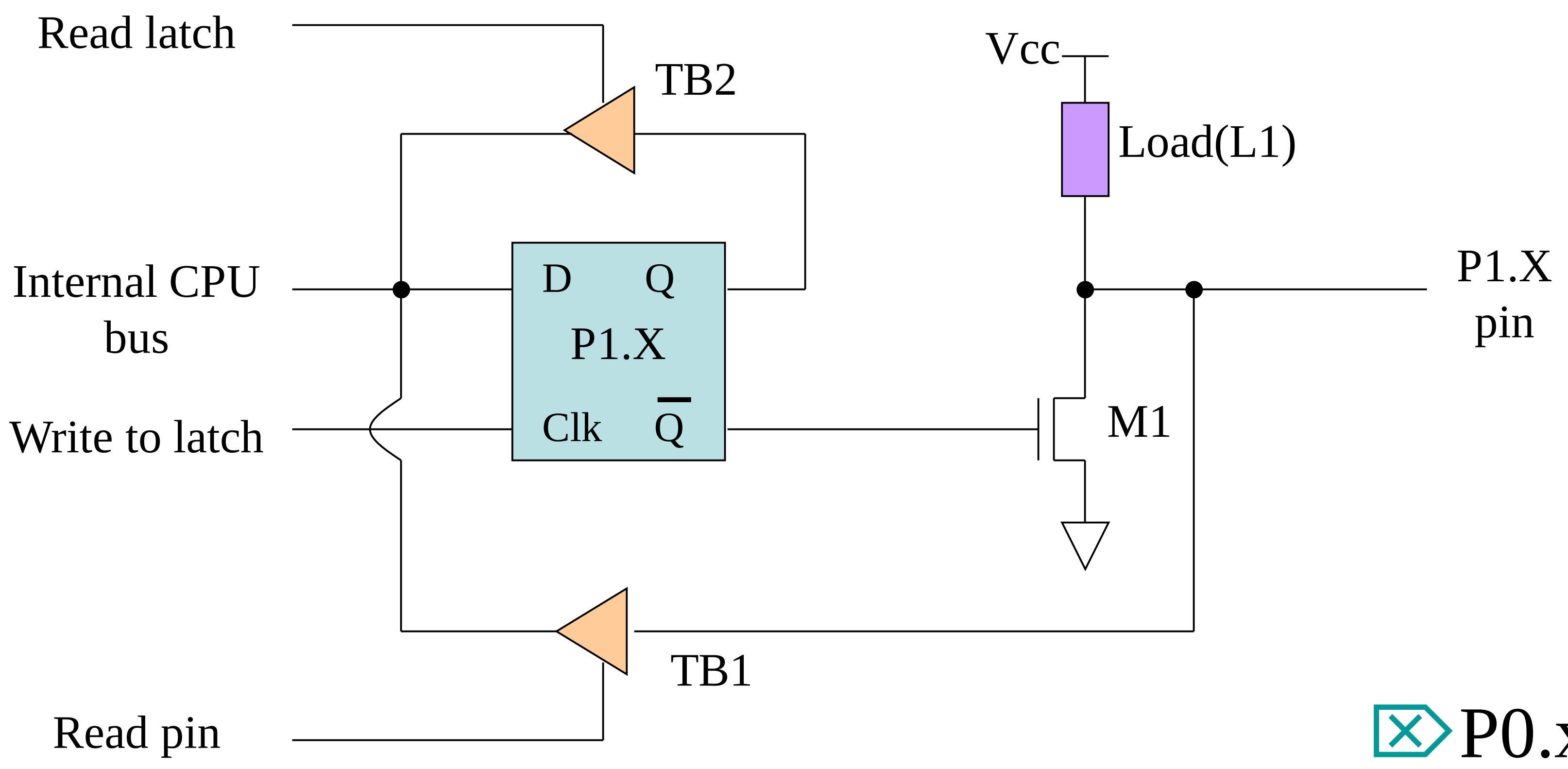
BIT	NAME	BIT ADDRESS	ALTERNATE FUNCTION
P3.0	RXD	B0H	Receive data for serial port
P3.1	TXD	B1H	Transmit data for serial port
P3.2	$\overline{\text{INT0}}$	B2H	External interrupt 0
P3.3	$\overline{\text{INT1}}$	B3H	External interrupt 1
P3.4	T0	B4H	Timer/counter 0 external input
P3.5	T1	B5H	Timer/counter 1 external input
P3.6	$\overline{\text{WR}}$	B6H	External data memory write strobe
P3.7	$\overline{\text{RD}}$	B7H	External data memory read strobe
P1.0	T2	90H	Timer/counter 2 external input
P1.1	T2EX	91H	Timer/counter 2 capture/reload

I/O Port Programming

Port 1（pins 1-8）

- Port 1 is denoted by P1.
  - P1.0 ~ P1.7
- We use P1 as examples to show the operations on ports.
  - P1 as an output port (i.e., write CPU data to the external pin)
  - P1 as an input port (i.e., read pin data into CPU bus)

A Pin of Port 1

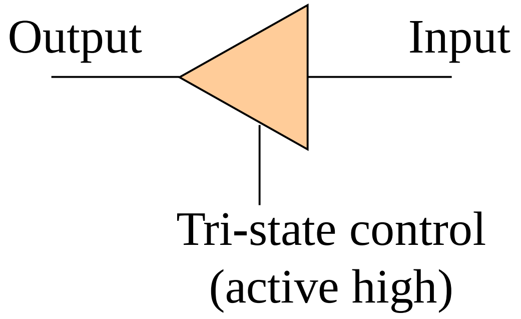


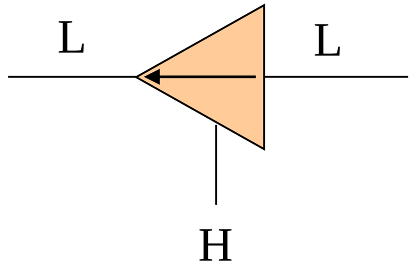
P0.x

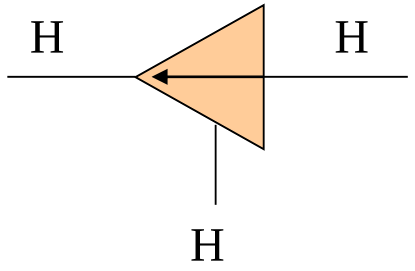
Hardware Structure of I/O Pin

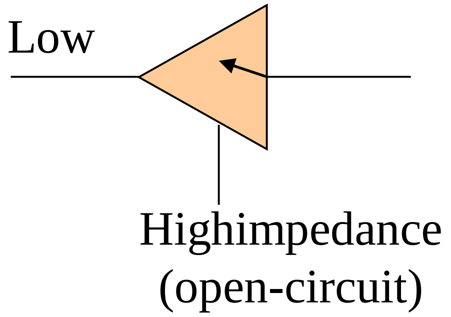
- Each pin of I/O ports
  - Internal CPU bus：communicate with CPU
  - A D latch store the value of this pin
    - D latch is controlled by “Write to latch”
      - Write to latch＝1：write data into the D latch
  - 2 Tri-state buffer：
    - TB1: controlled by “Read pin”
      - Read pin＝1：really read the data present at the pin
    - TB2: controlled by “Read latch”
      - Read latch＝1：read value from internal latch
  - A transistor M1 gate
    - Gate=0: open
    - Gate=1: close

Tri-state Buffer

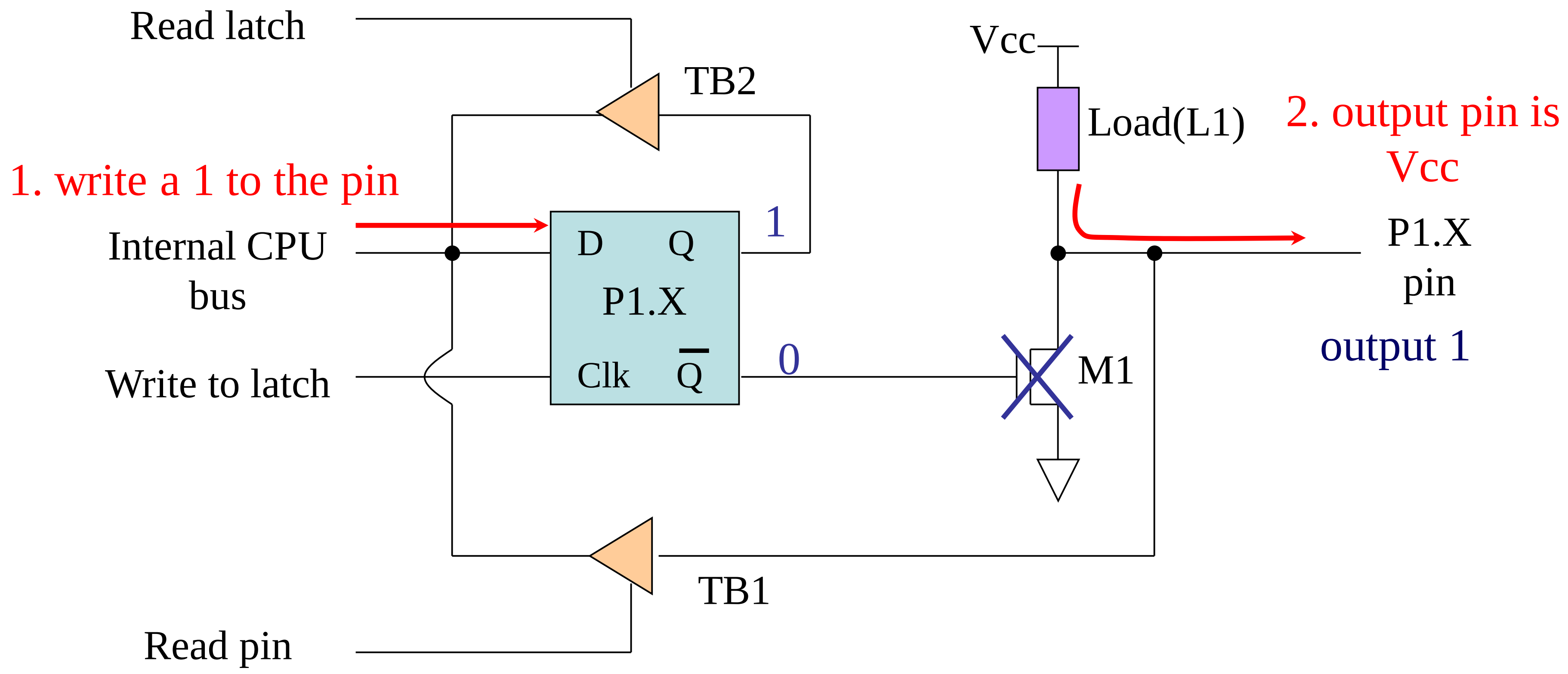




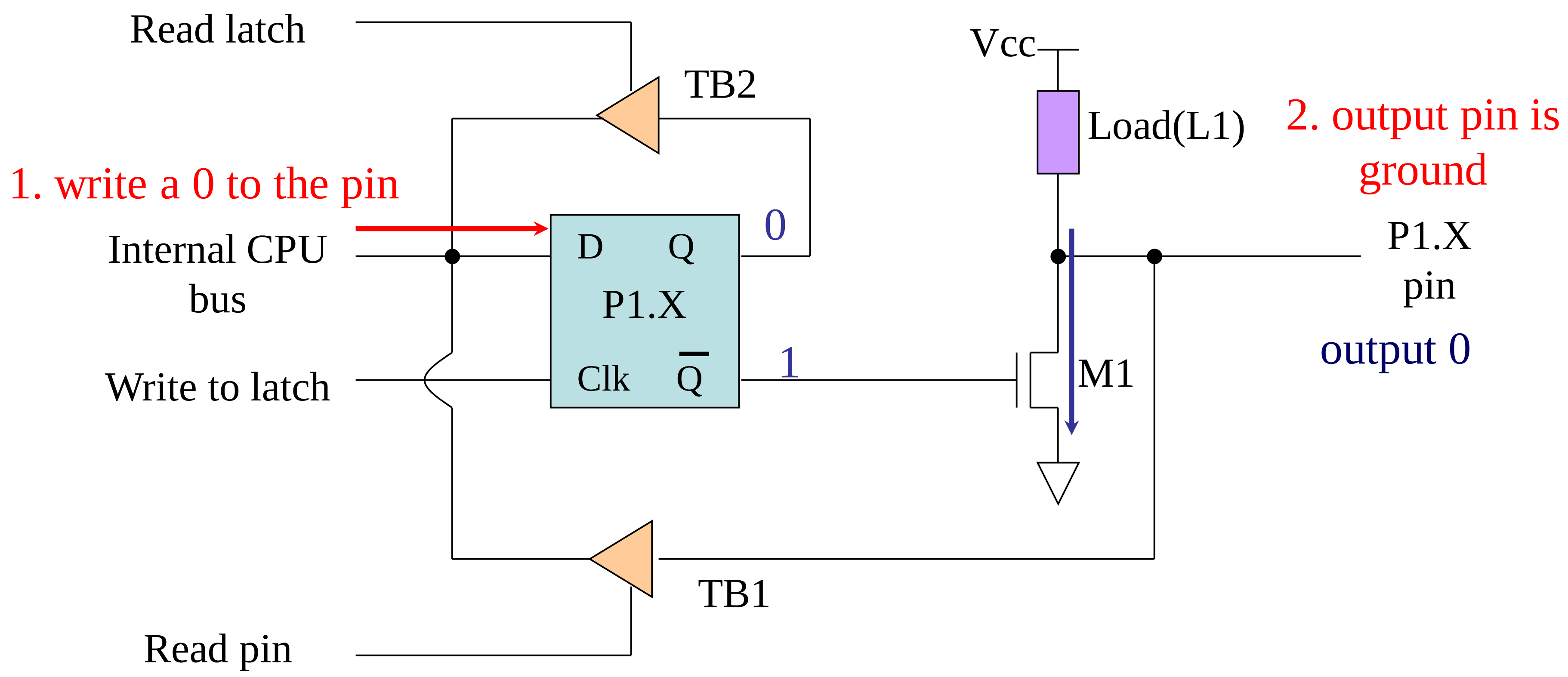




Writing “1” to Output Pin P1.X



Writing "0" to Output Pin P1.X



8051 IC

Port 1 as Output（Write to a Port）

- Send data to Port 1：

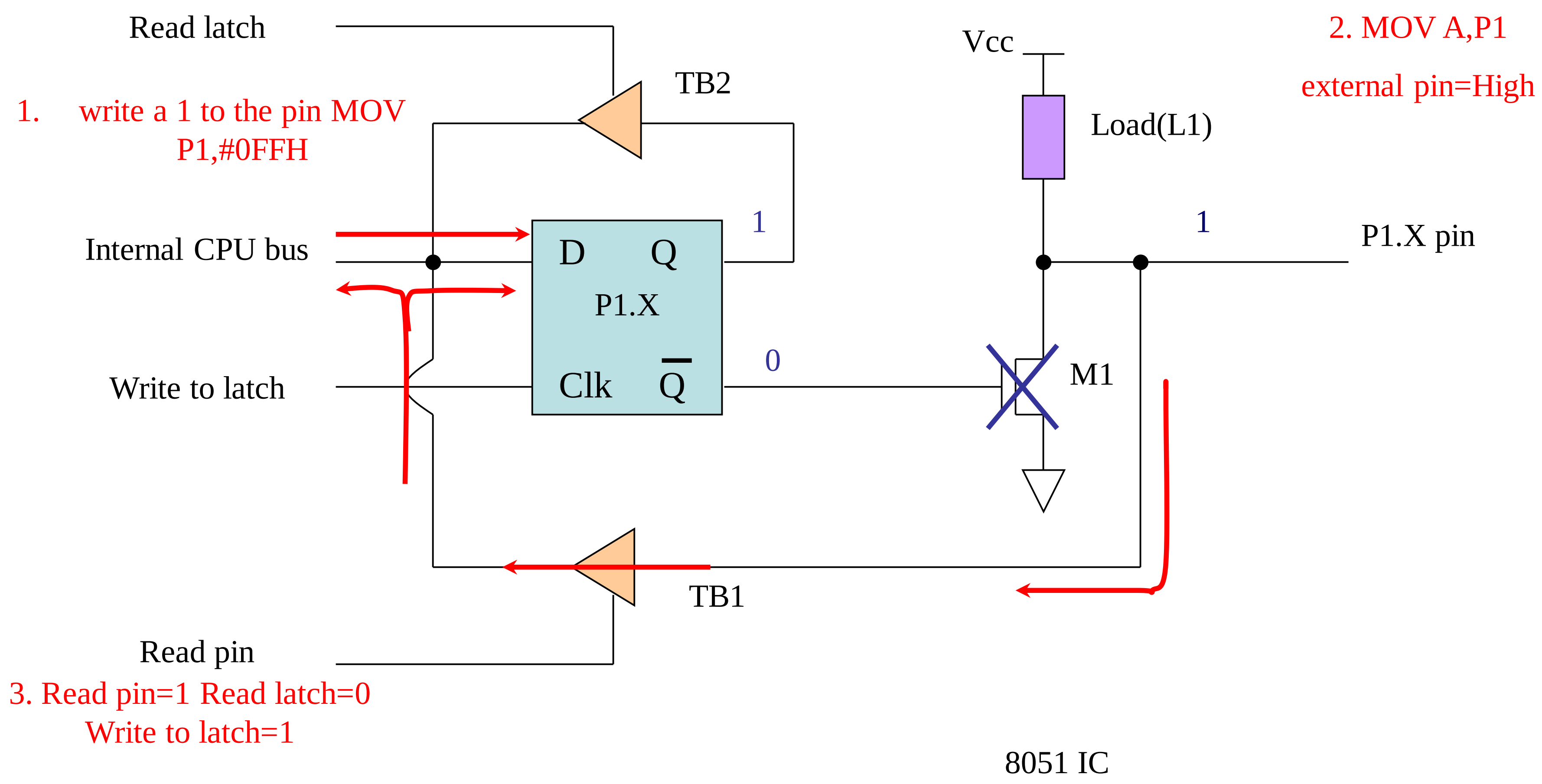
```
        MOV     A,#55H
BACK:   MOV     P1,A
        ACALL   DELAY
        CPL  A
        SJMP  BACK
```

  - Let P1 toggle.
  - You can write to P1 directly.

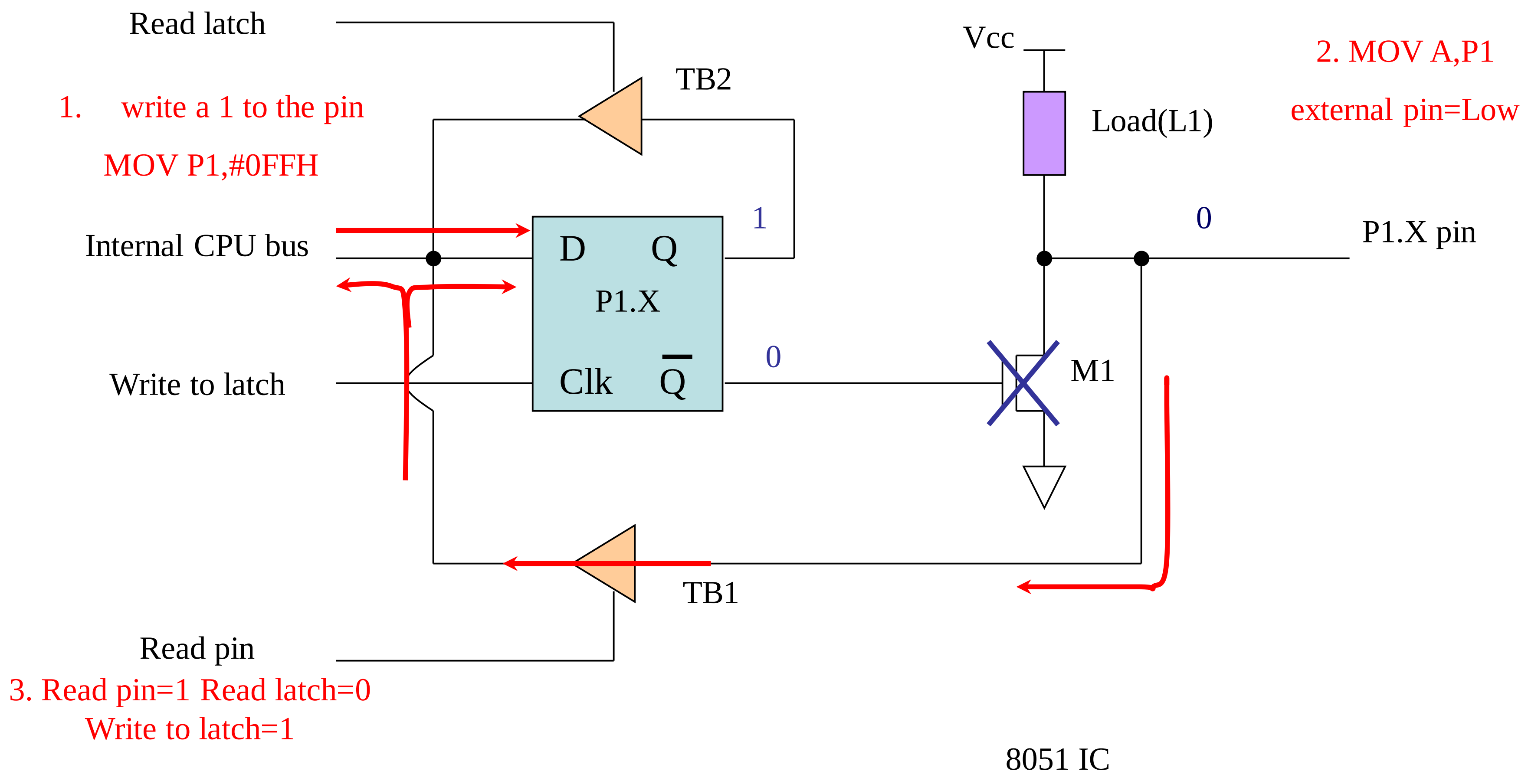
Reading Input v.s. Port Latch

- When reading ports, there are two possibilities：
  - Read the status of the input pin. （from *external pin value*）
    - MOV A, PX
    - JNB P2.1, TARGET ; jump if P2.1 is not set
    - JB P2.1, TARGET ; jump if P2.1 is set
  - Read the *internal latch* of the output port.
    - ANL P1, A ; P1 ← P1 AND A
    - ORL P1, A ; P1 ← P1 OR A
    - INC P1 ; increase P1

Reading "High" at Input Pin



Reading “Low” at Input Pin



Port 1 as Input（Read from Port）

- In order to make P1 an input, the port must be programmed by writing 1 to all the bit.

```
        MOV   A,#0FFH      ;A=11111111B
        MOV   P1,A         ;make P1 an input port
BACK:   MOV   A,P0         ;get data from P0
        MOV   P2,A         ;send data to P2
        SJMP  BACK
```

  - To be an input port, P0, P1, P2 and P3 have similar methods.

Instructions For Reading an Input Port

- Following are instructions for reading external pins of ports:

Mnemonics	**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

Reading Latch

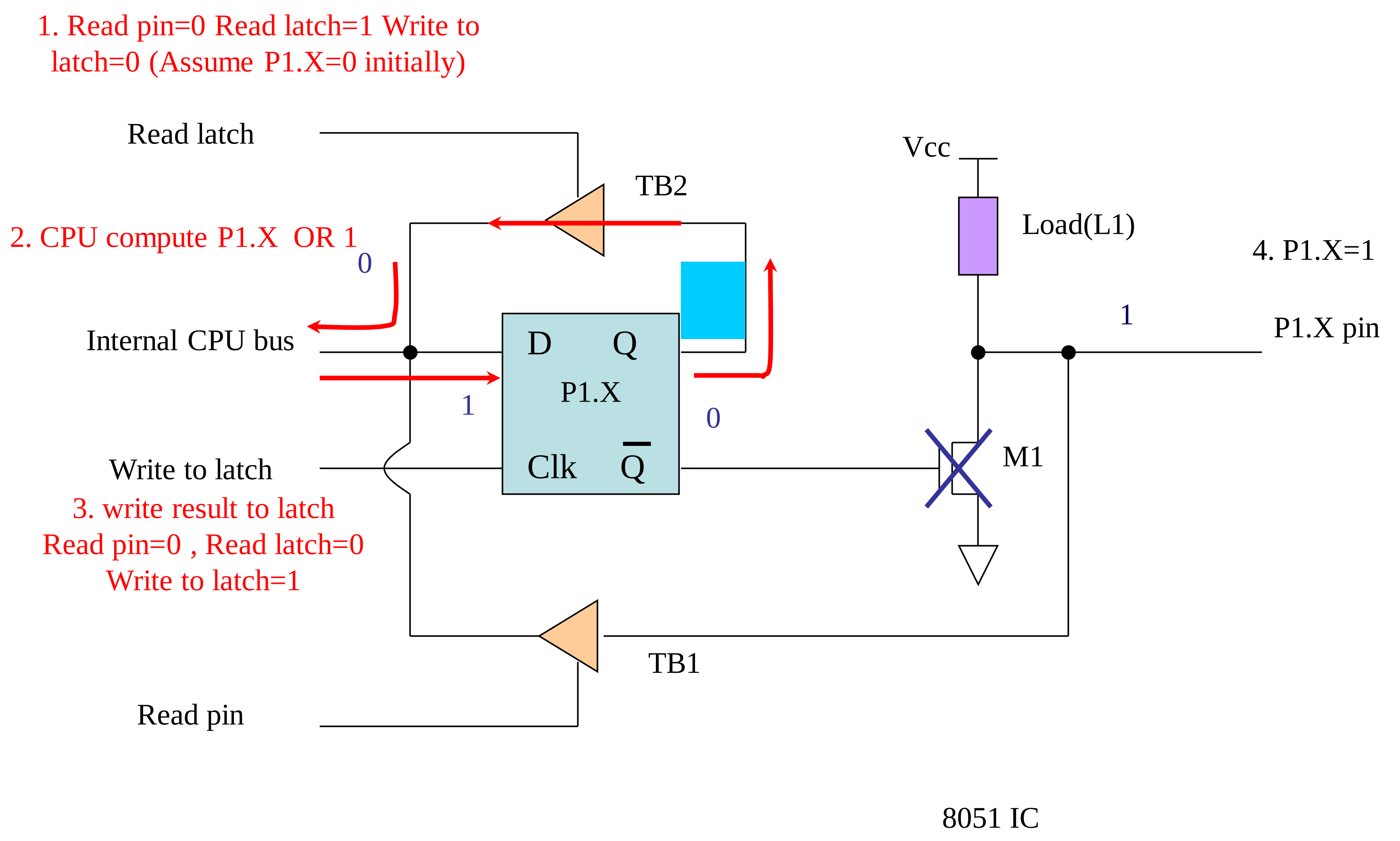
- Exclusive-or the Port 1：

  MOV P1,#55H ;P1=01010101

  ORL P1,#0F0H ;P1=11110101

  1. The read latch activates TB2 and bring the data from the Q latch into CPU.
     - Read P1.0=0
  2. CPU performs an operation.
     - This data is ORed with bit 1 of register A. Get 1.
  3. The latch is modified.
     - D latch of P1.0 has value 1.
  4. The result is written to the external pin.
     - External pin (pin 1: P1.0) has value 1.

Reading the Latch



Read-modify-write Feature

- Read-modify-write Instructions

- This features combines 3 actions in a single instruction：
  1. CPU reads the latch of the port
  2. CPU perform the operation
  3. Modifying the latch
  4. Writing to the pin
  – Note that 8 pins of P1 work independently.

Port 1 as Input（Read from latch）

- Exclusive-or the Port 1：

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

  - Note that the XOR of 55H and FFH gives AAH.
  - XOR of AAH and FFH gives 55H.
  - The instruction read the data in the latch (not from the pin).
  - The instruction result will put into the latch and the pin.

Read-Modify-Write Instructions

Mnemonics	**Example**
ANL	ANL P1,A
ORL	ORL P1,A
XRL	XRL P1,A
JBC PX.Y, TARGET	JBC P1.1, TARGET
CPL	CPL P1.2
INC	INC P1
DEC	DEC P1
DJNZ PX, TARGET	DJNZ P1,TARGET
MOV PX.Y,C	MOV P1.2,C
CLR PX.Y	CLR P1.3
SETB PX.Y	SETB P1.4

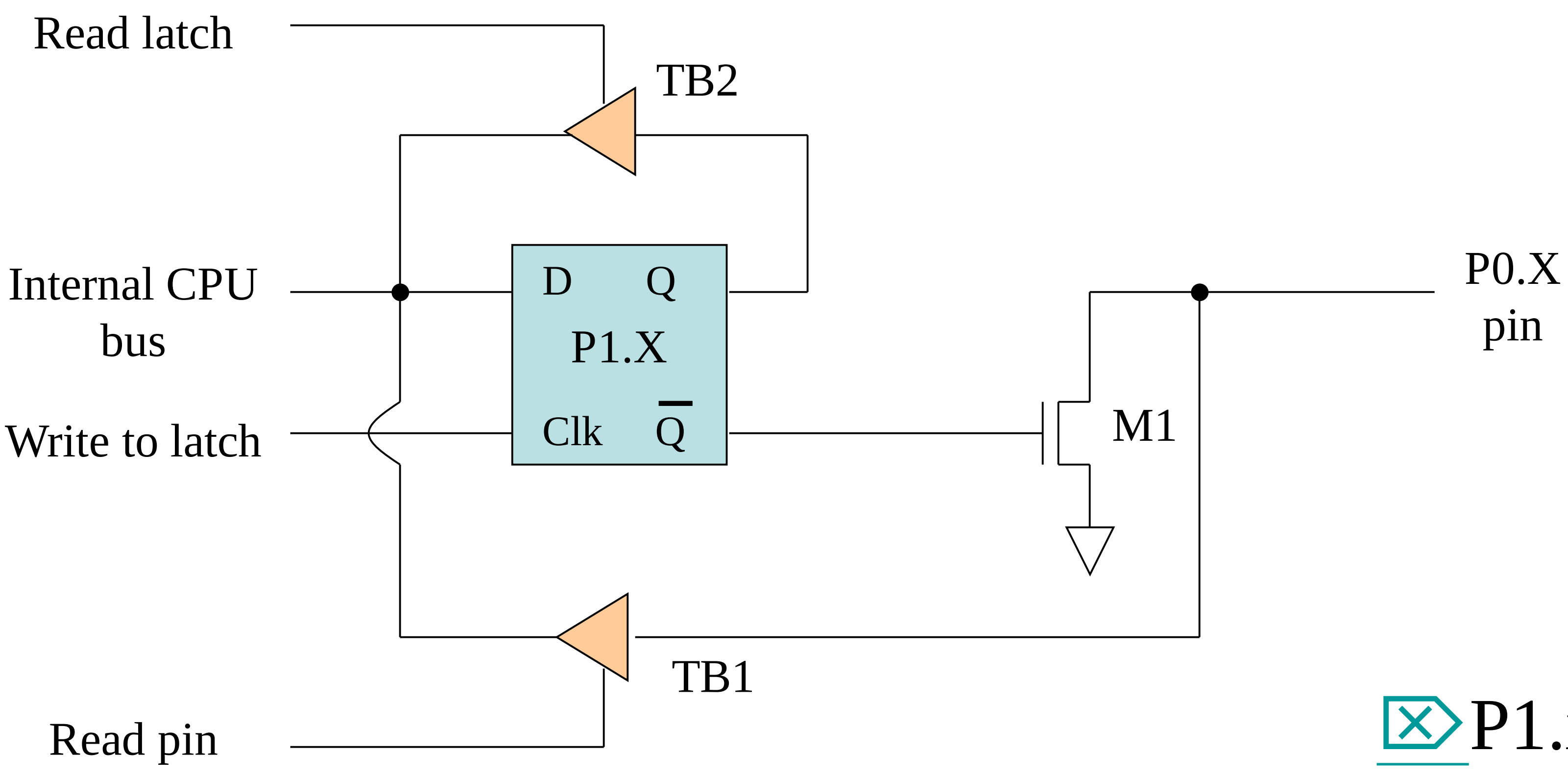
You are able to answer this Questions:

- How to write the data to a pin？
- How to read the data from the pin？
  - Read the value present at the external pin.
    - Why we need to set the pin first？
  - Read the value come from the latch（not from the external pin）.
    - Why the instruction is called read-modify write?

Other Pins

- P1, P2, and P3 have internal pull-up resisters.
  - P1, P2, and P3 are not open drain.
- P0 has no internal pull-up resistors and does not connects to Vcc inside the 8051.
  - P0 is open drain.
  - Compare the figures of P1.X and P0.X.
- However, for a programmer, it is the same to program P0, P1, P2 and P3.
- All the ports upon RESET are configured as output.

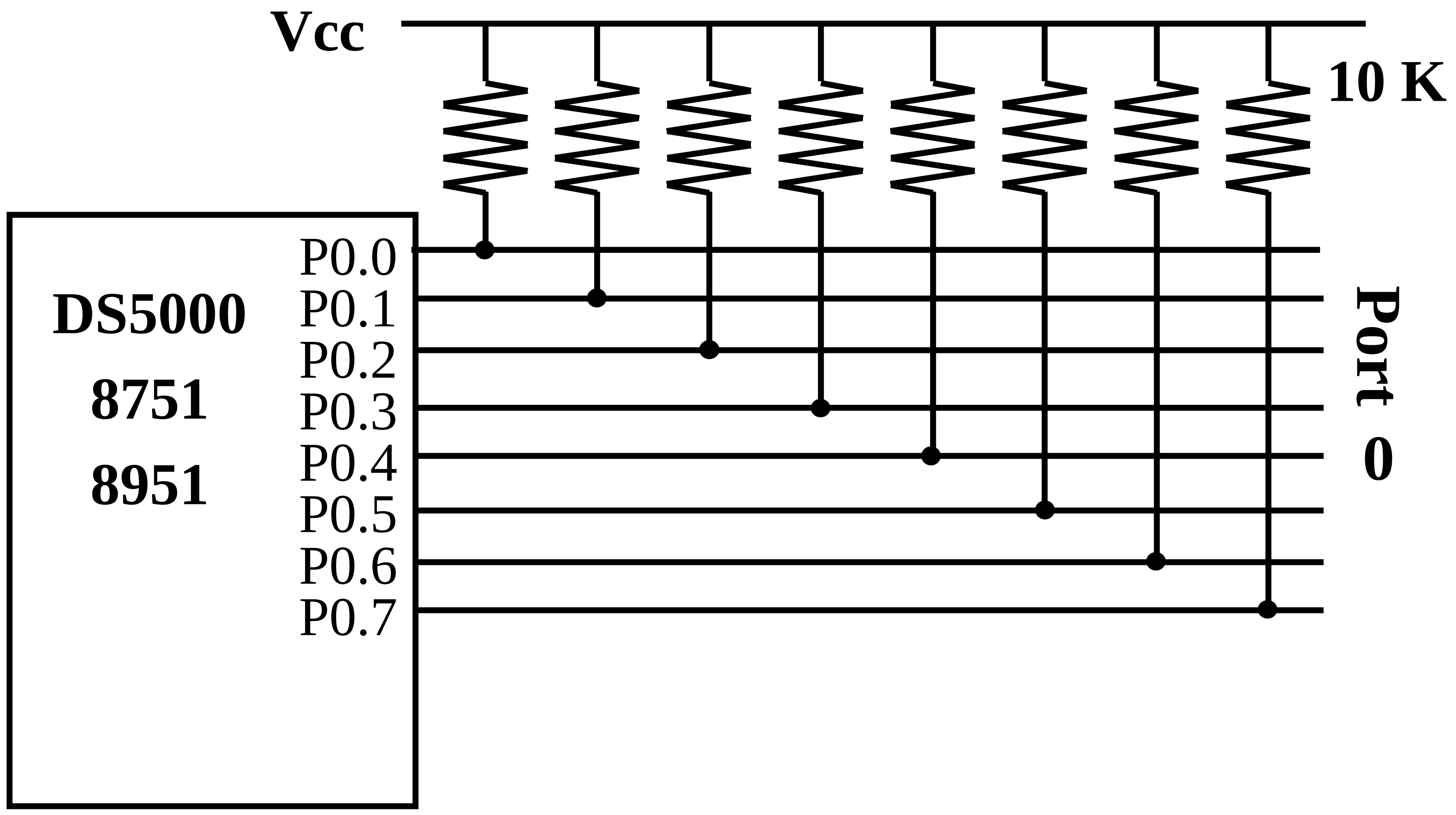
A Pin of Port 0



Port 0（pins 32-39）

- P0 is an open drain.
  - Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips.
- When P0 is used for simple data I/O we must connect it to external pull-up resistors.
  - Each pin of P0 must be connected externally to a 10K ohm pull-up resistor.
  - With external pull-up resistors connected upon reset, port 0 is configured as an output port.

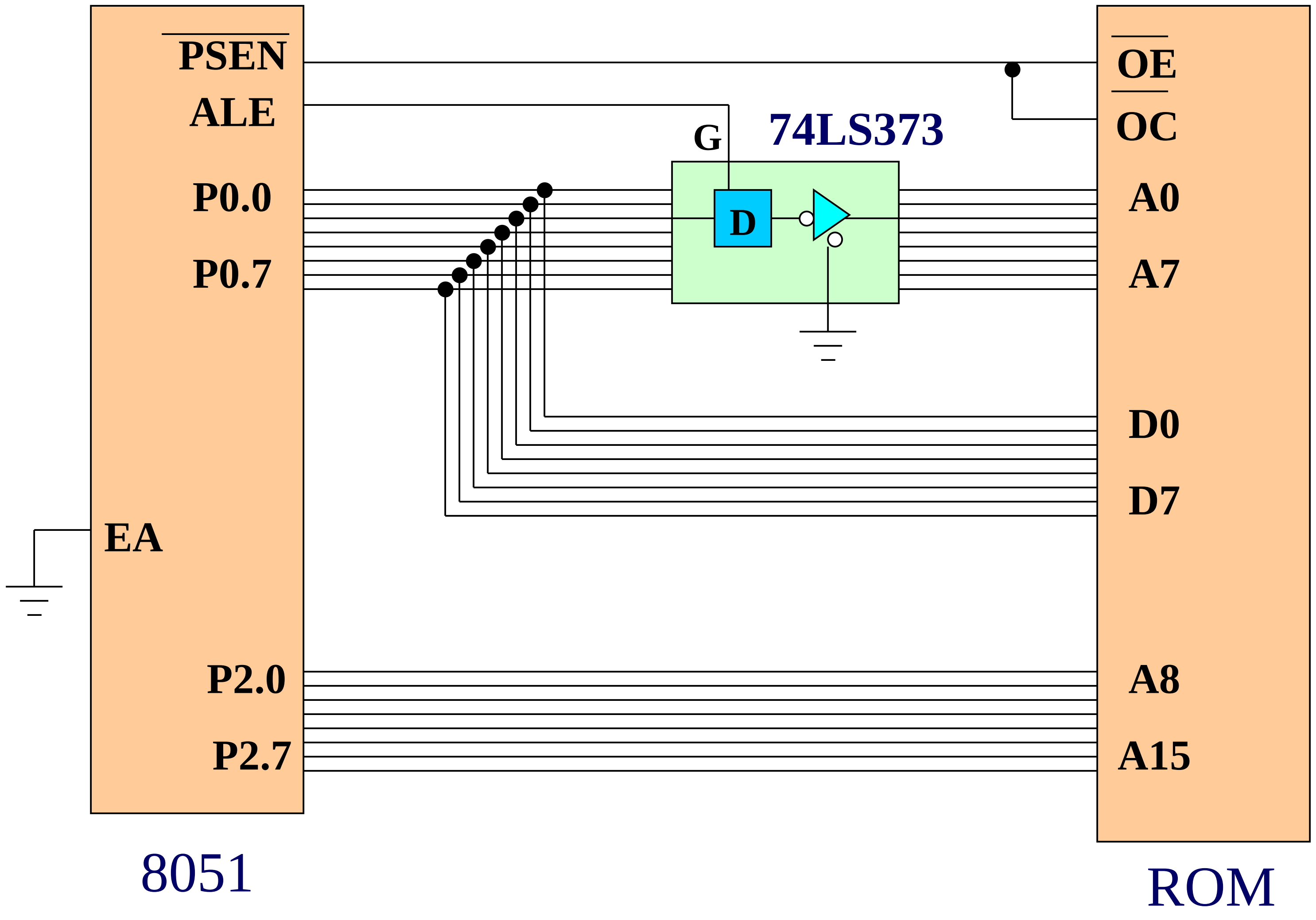
Port 0 with Pull-Up Resistors



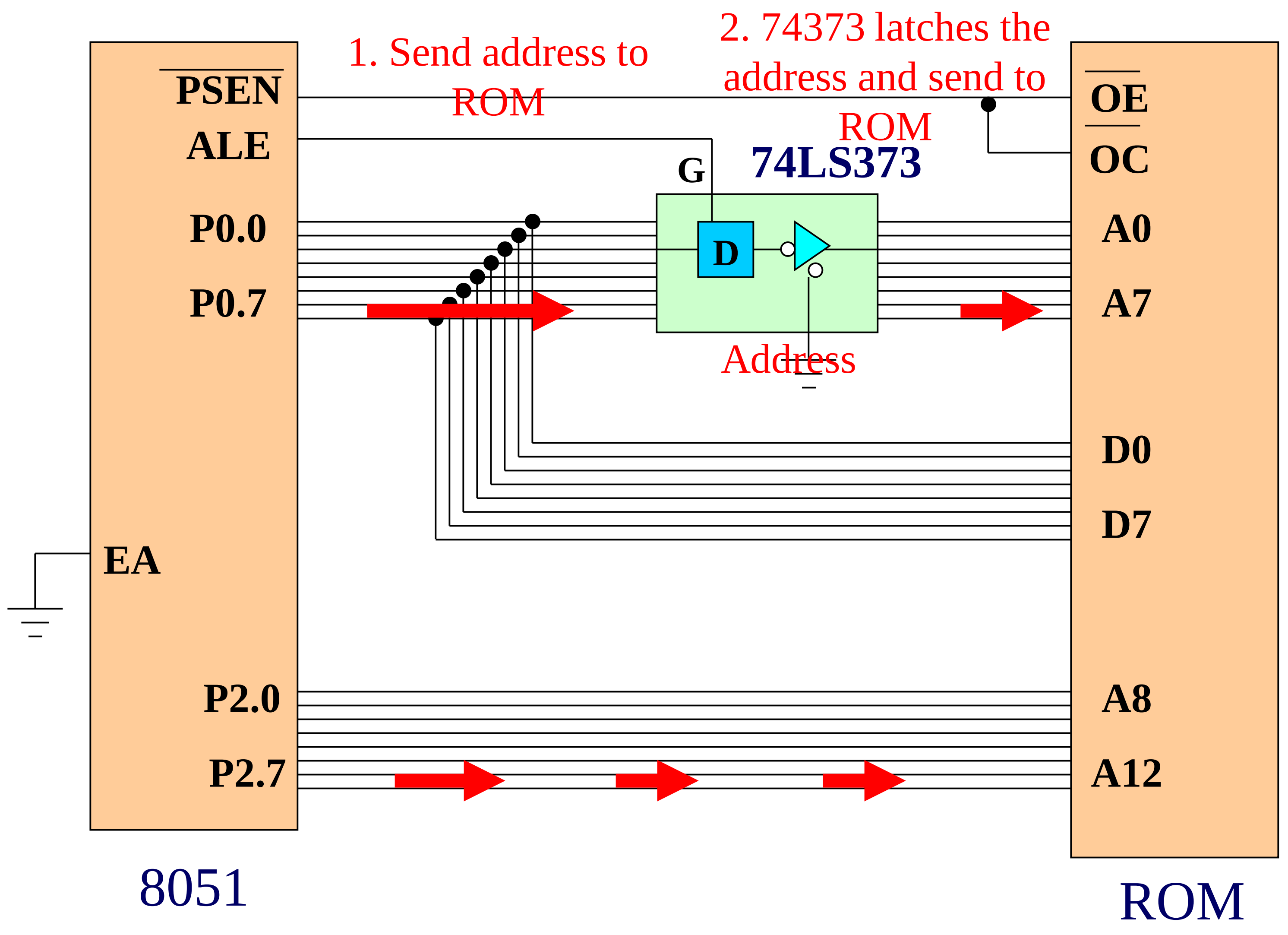
Dual Role of Port 0

- When connecting an 8051 to an external memory, the 8051 uses ports to send addresses and read instructions.
  - 8051 is capable of accessing 64K bytes of external memory.
  - 16-bit address：P0 provides both address A0-A7, P2 provides address A8-A15.
  - Also, P0 provides data lines D0-D7.
- When P0 is used for address/data multiplexing, it is connected to the 74LS373 to latch the address.
  - There is no need for external pull-up resistors

74LS373

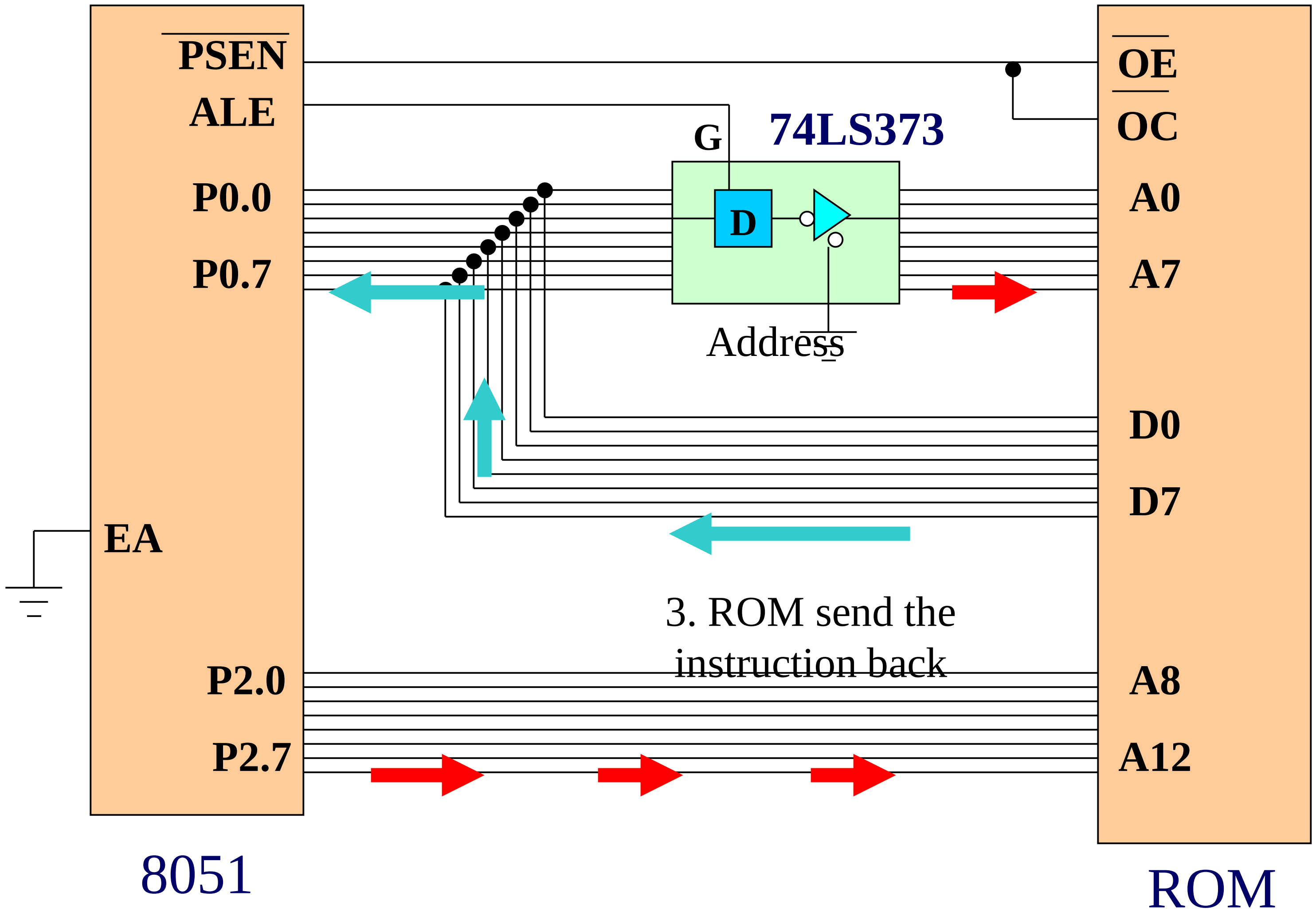

$\overline{\text{PSEN}}$
ALE
P0.0
P0.7
EA
P2.0
P2.7
8051
G
D
74LS373
$\overline{\text{OE}}$
$\overline{\text{OC}}$
A0
A7
D0
D7
A8
A15
ROM

Reading ROM (1/2)



Reading ROM (2/2)

2. 74373 latches the address and send to ROM



3. ROM send the instruction back

ALE Pin

- The ALE pin is used for de-multiplexing the address and data by connecting to the G pin of the 74LS373 latch.
  - When ALE=0, P0 provides data D0-D7.
  - When ALE=1, P0 provides address A0-A7.
  - The reason is to allow P0 to multiplex address and data.

Port 2（pins 21-28）

- Port 2 does not need any pull-up resistors since it already has pull-up resistors internally.
- In an 8051-based system, P2 are used to provide address A8-A15.

Port 3（pins 10-17）

- Port 3 does not need any pull-up resistors since it already has pull-up resistors internally.
- Although port 3 is configured as an output port upon reset, this is not the way it is most commonly used.
- Port 3 has the additional function of providing signals.
  - Serial communications signal：RxD, TxD（Chapter 10）
  - External interrupt：/INT0, /INT1（Chapter 11）
  - Timer/counter：T0, T1（Chapter 9）
  - External memory accesses in 8031-based system：/WR, /RD（Chapter 14）

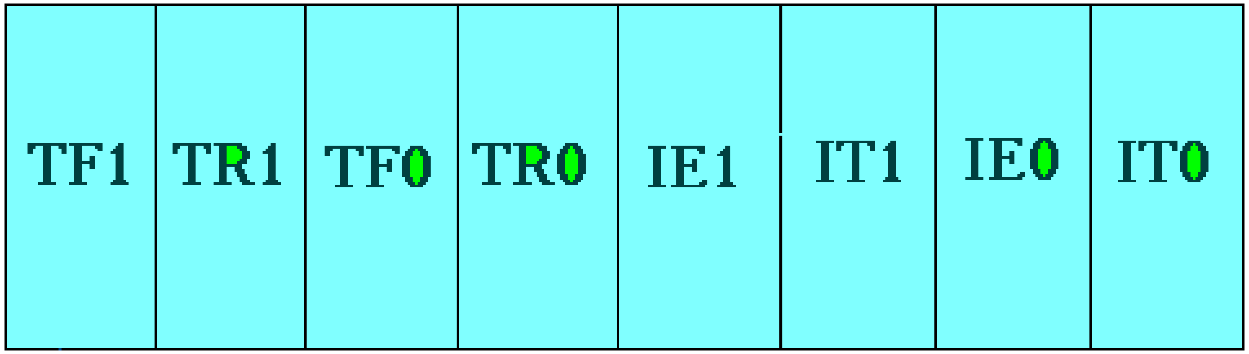
Port 3 Alternate Functions

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

Counters & timers

To relinquish the Burden of the processor from software loops for timing & counting , two 16 bit counters T0 & T1 are provided. These 2 are divided into 8 bit registers as timer low(TL0, TL1) and high(TH0, TH1) bytes. Counter action is controlled by timer mode control (TMOD) & timer/counter control reg. (TCON)

TCON Register:

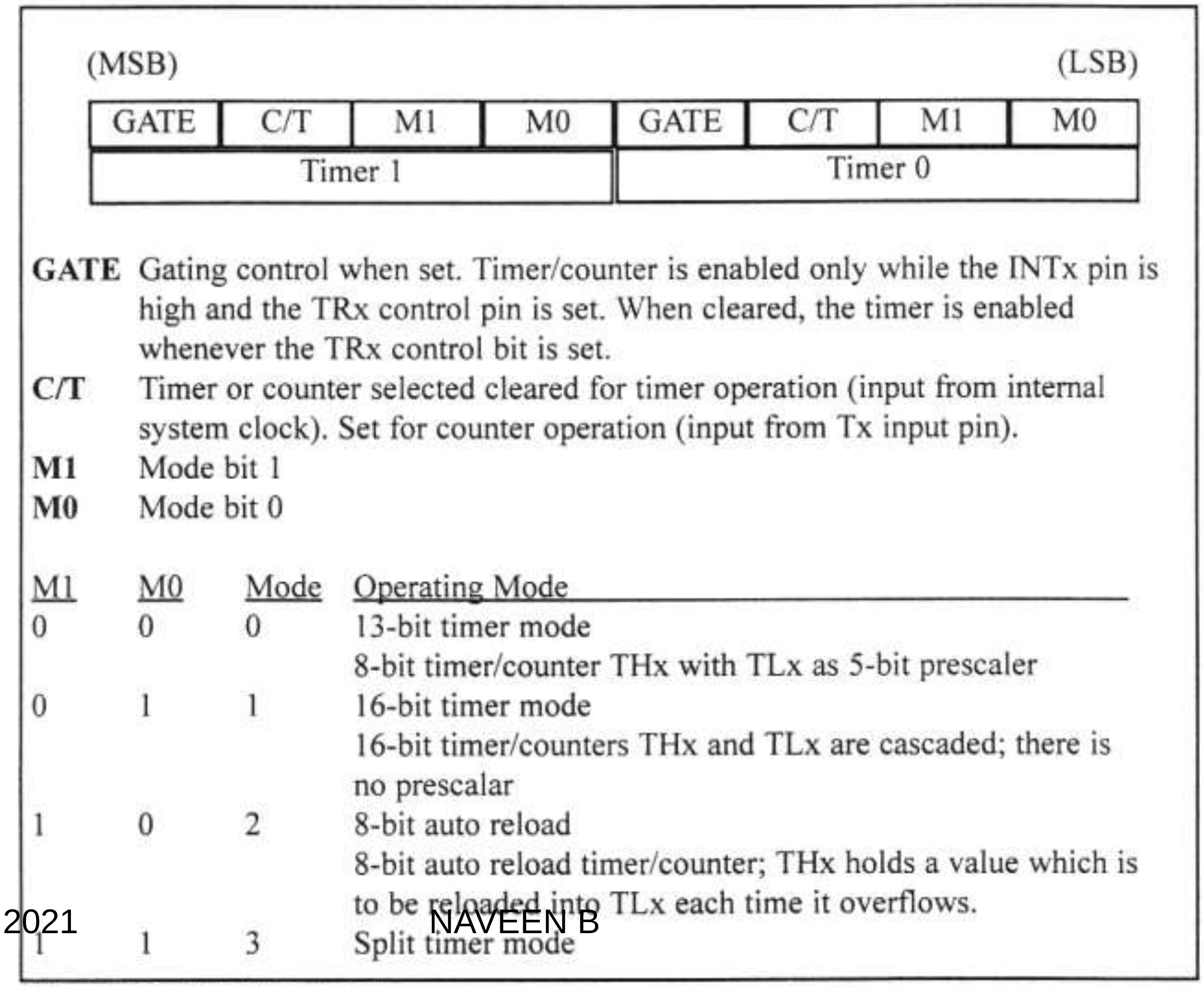


TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- TF1: Timer 1 overflow flag.
- TR1: Timer 1 run control bit.(set to 1 to enable timer to count)
- TF0: Timer 0 overflag.
- TR0: Timer 0 run control bit.
- IE1: External interrupt 1 edge flag.
- IT1: External interrupt 1 type flag.
- IE0: External interrupt 0 edge flag.
- IT0: External interrupt 0 type flag.

Timer mode control reg. (TMOD)

(Not bit addressable)



(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

GATE Gating control when set. Timer/counter is enabled only while the INTx pin is high and the TRx control pin is set. When cleared, the timer is enabled whenever the TRx control bit is set.

C/T Timer or counter selected cleared for timer operation (input from internal system clock). Set for counter operation (input from Tx input pin).

M1 Mode bit 1

M0 Mode bit 0

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/counters THx and TLx are cascaded; there is no prescalar
1	0	2	8-bit auto reload 8-bit auto reload timer/counter; THx holds a value which is to be reloaded into TLx each time it overflows.
1	1	3	Split timer mode

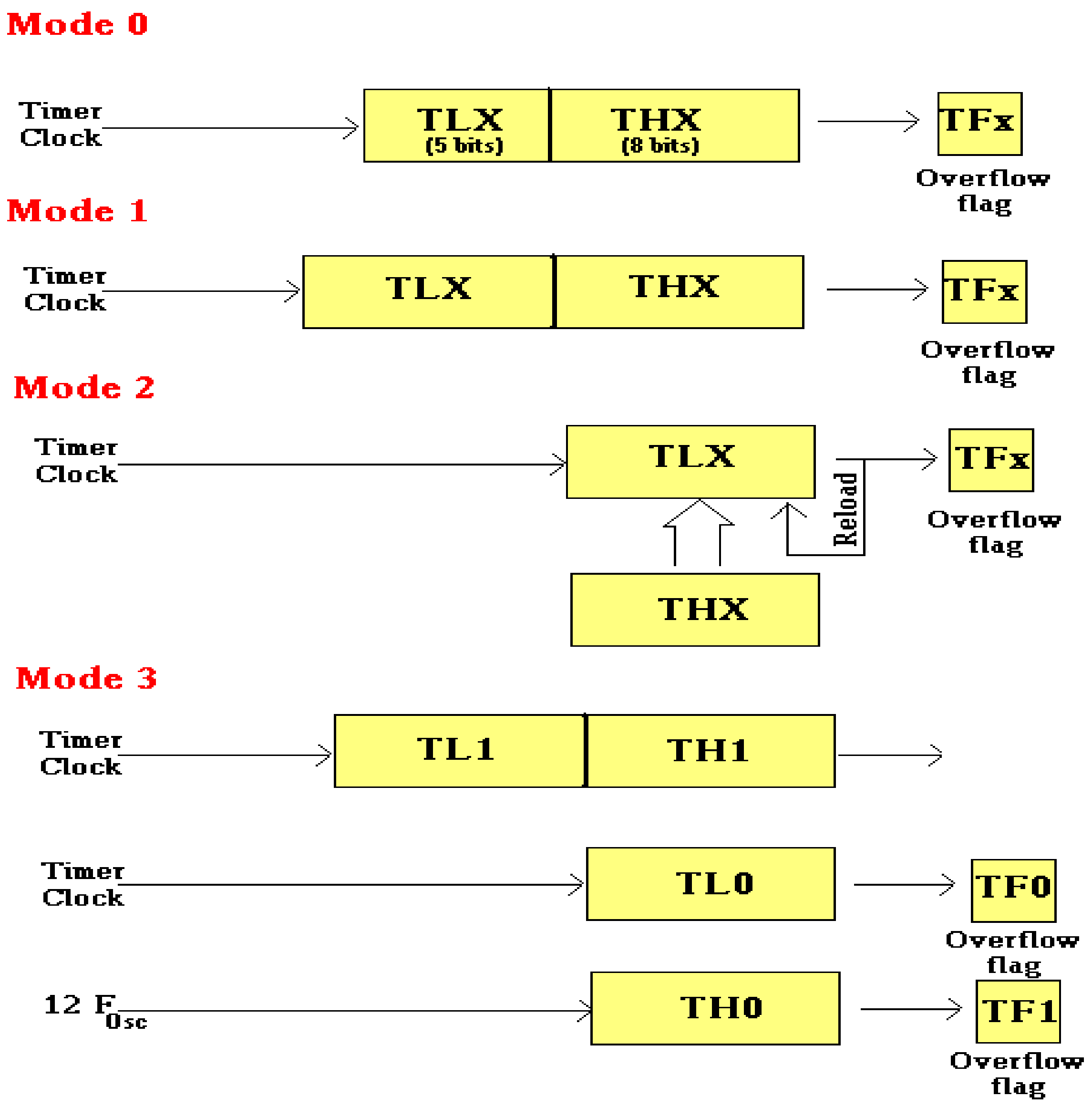
Figure 9-3. TMOD Register

Gate = 0 : start & stop are software control

(Start by SETB TRx & stop by CLR TRx)

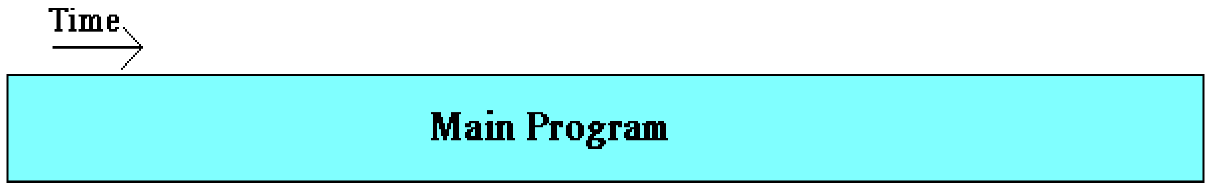
Gate = 1 : start & stop are controlled by hardware

by an external source (Pins P3.2 & P3.3)

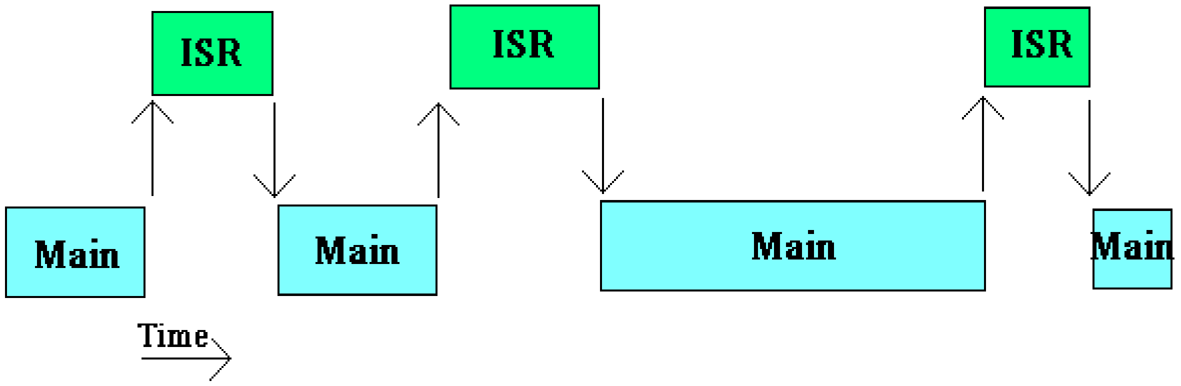

Mode 0
Timer Clock
TLX (5 bits)
THX (8 bits)
TFx
Overflow flag
Mode 1
Timer Clock
TLX
THX
TFx
Overflow flag
Mode 2
Timer Clock
TLX
Reload
TFx
Overflow flag
THX
Mode 3
Timer Clock
TL1
TH1
Timer Clock
TL0
TF0
Overflow flag
12 F Osc
TH0
TF1
Overflow flag

Interrupt :

Program execution without intrrupts :



Program execution with intrrupts :



ISR : Intrrupt Service Routin

Interrupt Vectors

Interrupt	Vector Address
System Reset	0000H
External 0	0003H
Timer 0	000BH
External 1	0013H
Timer 1	001BH
Serial Port	0023H
Timer 2	002BH

Interrupt Enable Register :

EA	—	ET2	ES	ET1	EX1	ET0	EX0

- EA : **Global enable/disable**.
- **---** : **Undefined**.
- ET2 :Enable Timer 2 interrupt.
- ES :Enable Serial port interrupt.
- ET1 :Enable Timer 1 interrupt.
- EX1 :Enable External 1 interrupt.
- ET0 : Enable Timer 0 interrupt.
- EX0 : Enable External 0 interrupt.

Interrupt priority register

---	---	PT2	PS	PT1	PX1	PT0	PX0

Bit	symbol	function
7	--	not implemented
6	--	not implemented
5	PT2	reserved for future use
4	PS	priority for serial port interrupt
3	PT1	priority for timer 1 overflow interrupt
2	PX1	priority for external interrupt 1
1	PT0	priority for timer 0 overflow interrupt
0	PX0	priority for external interrupt 0

IP=0 (low priority)

IP=1 (High priority)

Interrupt priority

- If the 2 interrupts with the same priority occur at the same time, then they have the following ranking-

- IE0, TF0, IE1,   TF1,   Serial interrupt (RI or TI)

Serial data I/O

- 8051 has a full duplex serial port
- SBUF to hold data, SCON controls data communication and PCON controls data rates
- SBUF is physically 2 regs., one is to hold write only data and another is read only data

Serial port control (SCON)

SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

Bit	**symbol**	**function**
7	SM0	serial port mode bit 0
6	SM1	serial port mode bit 1
5	SM2	multiprocessor commn. Bit
4	REN	receive enable bit
3	TB8	transmitted bit 8
2	RB8	received bit 8
1	TI	transmit interrupt flag
0	RI	receive interrupt flag

There are 4 programmable modes for serial data commn.

SM0	SM1	Mode	Description
0	0	0	shift register; baud=f/12
0	1	1	8-bit UART ; baud=variable
1	0	2	9-bit UART ; baud=f/32 or f/64
1	1	3	9-bit UART; baud=variable

Mode 0 (shift register mode) : SBUF to transmit/receive 8 data bits using pin RXD for both functions. TXD is used to supply shift pulses to external circuits. Baud rate is fixed at f/12.

Mode 1 (standard UART): SBUF becomes a 10-bit full duplex receiver/transmitter at the same time using RXD & TXD. 1 start bit, 8 data bits (LSB first) & 1 stop bit. 8 data bits go to SBUF, start bit is discarded & stop bit is saved in RB8.

Mode 2 (multiprocessor mode): similar to mode1 except 11 bits are transmitted (9 data bits). 9th bit is copied from bit TB8 during transmit & stored in bit RB8 when data is received. Both start & stop bits are discarded.

For multiprocessor commn., set the 9th data bit.

Mode 3 (serial data mode): this is identical to mode 2 except that the baud rate is as in mode 1, using timer 1 to generate frequencies.

PC variable Baud Rates

110 bps

150

300

600

1200

2400

4800

9600 (default)

19200

Power mode control (PCON)

SMOD	--	-	-	GF1	GF0	PD	IDL

Bit	symbol	function
7	SMOD	serial baud rate modify bit
6-4	--	not implemented
3	GF1	general purpose user flag bit 1
2	GF0	general purpose user flag bit 0
1	PD	power down bit
0	IDL	ideal mode bit

- This is not bit addressable

8051Instruction Set

By Dr. Naveen B

Instruction Set

Data Transfer Instructions

MOVE Destination, Source

MOV Rn, A
MOV Rn, direct

MOV DPTR, # data 16
MOVC A, @ A + DPTR
MOVC A, @ A+PC

MOVX A, @DPTR
MOVX @DPTR, A
MOVX A, @Ri

**** No flags are affected**

Logical Operations

Byte level Logical operations

The operations are done in each individual bit of the source & destination bytes.

ANL **A , Rn**

ORL **A , @Rp**

XRL **A , #27h**

- **Destination is Accumulator or direct addressing & source may be any addressing mode.**
- **Use port as a source but not as a destination**

CLR A **Clear Acc**

CPL A **Complement Acc**

** No flags are affected

Bit level Logical Operations

- It is very convenient to alter a single bit of a byte.

CPL C (Complement)

CPL bit

ANL C, bit AND direct bit to CY

ANL C, /bit

ORL C, bit

ORL C, /bit

EX:SETB 00h ...Bit 0 of RAM byte 20h = 1

ANL C, /00h ...C=0; bit 0 of RAM byte 20h =1

BOOLEAN OPERATIONS

CLR C	**Clear CY**
CLR bit	**Clear direct bit**
SETB C	**Set CY**
SETB bit	**Set direct bit (SETB P2.4)**
MOV C, bit	**Move direct bit to CY**
MOV bit, C	**Move CY to direct bit**
MOV 7Fh, C	**…. Bit 7 of RAM byte 2Fh =1..Assume C=1**

Note:

- CLR A --- it is for byte & only ‘A’ reg. no other registers or addressing modes

CLR Acc.0 for individual bit in ‘A’ reg.

CLR b bits in SFRs & bit addressable area.

Rotate and Swap Operations

RL A	**Rotate Acc Left**	**:** $b_{n+1} \leftarrow b_n$
RR A	**Rotate Acc Right**	**:** $b_n \leftarrow b_{n+1}$
RLC A	**Rotate Acc Left through Carry**	
RRC A	**Rotate Acc Right through Carry**	

SWAP A **Exchange between the nibbles**

* Only CY flag is affected in RRC A & RLC A

Example programs:

1) Double the number in Reg. R2 & put the result in R3 & R4.

```
CLR C
MOV A, R2
RLC A
MOV R4, A
CLR A
RLC A
MOV R3, A
```

2) OR the contents of ports 1 & 2, put the result in external RAM location 0100h

```
MOV A, 90h
ORL A, 0A0h
MOV DPTR, #0100h
MOVX @DPTR, A
```

3) Configure P1 to read switches at P1.0 & P1.1. If P1.0 is high, turn ON a relay connected to P2.5 by sending a logic high o/p. If P1.0 is low, clear P2.5. If the status of the switch at P1.1 is high, turn OFF the relay connected to P2.6 by sending a logic low o/p. If P1.1 is low, set P2.6 to high state.

Logic:

i/p's at P1	o/p's at P2
XXXXXX00	X10XXXXX
XXXXXX01	X11XXXXX
XXXXXX10	X00XXXXX
XXXXXX11	X01XXXXX

Program:

```
        mov p1, #0FFh
         Mov p2, #00h
         Mov A, p1
         Anl A, #03h
         Cpl Acc.1
         Rl a
         Rl a
         Rl a
         Rl a
         Rl a
         Mov p2, A
```

4) Swap every even numbered bit of register R3 in bank 0 woth the odd numbered bit to its left. Swap bit 0 with bit 1, bit 2 with bit 3, and so on until bit 6 is swapped with bit 7

```
MOV A, R3
RL A
ANL A, #0AAh
PUSH 0E0h
MOV A, R3
RR A
ANL A, #55h
MOV R3, A
POP 0E0h
ORL 03h, A       (ORL R3, A  …. Not allowed)
```

5. Assume that bit P2.2 is used to control an outdoor light and bit P2.5 a light inside a building .Show how to turn on the outside light & turn off the inside one.

```
SETB C
     ORL C, P2.2
     mov P2.2, C
     CLR C
     ANL C, P2.5
     mov P2.5 ,C
```

6. Assume that registers A has packed BCD. Write a pgm to convert packed BCD to two ASCII numbers & place them in R2 & R6.

```
mov A, # 29h
        mov R2, A
        ANL A, # 0Fh
        ORL A, #30h
        mov R6, A
        mov A, R2
        ANL A, # 0F0h
        RRA
         RRA
         RRA
         RRA
         ORL A, # 30h
          mov R2, A
```

Arithmetic Instructions

<u>Incrementing and decrementing</u>

INC A ; operand may be any addressing except Immediate

INC DPTR

DEC A :operand may be any addressing except Immediate

There is no "DEC DPTR"

- No flags are affected

Addition and subtraction:

- ADD A, Source byte [OV, AC, CY]
- ADDC A, Source byte [OV, AC, CY]
- SUBB A, source byte [OV, AC, CY]

Subtract with borrow:

(A) ← (A) - source byte -CY

'A' register is the destination, & source may be any addressing mode

Examples

Unsigned addition: This make use of the carry flag to detect when the result of an ADD operation is a number larger than FF h.

```
                00 to 255d
   95d    =   01011111 b          =5Fh
  189d    =   10111101 b          =BDh
  284d      100011100 b            11Ch
```

Signed addition: If unlike signed numbers are added, then it is not possible for the result to be larger than -128 d to +127 d, and the result will always be correct.

```
Ex1:   -001d    = 11111111 b                  = FF h
       +027d   = 00011011 b                  = 1Bh
       +026d    100011010 b =+ 026d           11Ah
```

Adding two +ve numbers, result may exceed +128d

```
Ex2:   +100d      =01100100b                 =64h
       +050d      = 00110010b                 32h
        150d       010010110b  = -106d       096h      correct result= +22d
                                                  OV=1


Ex3:   +045d      =00101101b              =2Dh
       +075d      =01001011b              =4Bh
       +120d       001111000b =120d       078h    OV=0 (result not exceeded)
```

The result of adding two –ve numbers together for a sum that does not exceed the –ve limit.

```
Ex:  - 030d     = 11100010b       = E2h
     - 050d       11001110b       = CEh
     - 080d      110110000b         1B0h


          OV=0
```

Adding Two –ve numbers whose sum does exceed -128d

```
- 070d     = 10111010b                = BA h
- 070d     = 10111010b                =  BAh
- 140d     =101110100b = +116d     174h
```

OV= 1 (correct result= -12d)

Flags		Action
CY	OV	
0	0	none
0	1	compliment the sign
1	0	none
1	1	compliment the sign

Unsigned subtraction:

Subtraction of a larger number from a smaller number.

```
  015d      = 00001111b   = 0F0h
  100d      = 01100100b   = 064h
- 085d      =110101011b
```

The carry flag is set to 1 & OV=0.

2’s compliment of the result = 085d.

```
  100d   =  01100100b     = 64h
  015d   =  00001111b     = 0Fh
  085d   =001010101b      =055h
```

C=0, OV=0 (Magnitude of the result is in true form).

Signed subtraction:

When numbers of like sign are subtracted it is impossible for the result to exceed positive or negative magnitude limits of +127 or -128.

+100d =01100100b(carry flag is 0 before SUBB)=64h

sub +126d = 01111110b = 7Eh

- 026d =111100110b = 1E6h

cy=1 , OV=0

Using two negative numbers

-061d = 11000011b (CY=0 before SUBB) = C3h

Subb $\underline{\text{-116d}}$ = $\underline{\text{10001100}}$b = $\underline{\text{8C}}$h

+055d 000110111b 037h

CY=0, OV=0

An overflow is possible when subtracting numbers of opposite sign

- 099d =10011101b(cy=0 before SUBB) =9Dh

+100d =01100100b =64h

-199d =000111001b = +057d =039h

OV= 1 , cy =0

Because the overflow flag is set to 1, the result must be adjusted so that 2's compliment is 71d

```
+ 087d   = 01010111b(cy=0 before SUBB)  = 57h
- 052d   = 11001100b                    = cch
------------------------------------------------
+139d    =110001011b = -177d            =18bh
```

OV=CY=1

The magnitude can be interpreted as +011d

The general rule is that if the overflow flag is set to 1, then compliment the sign bit . The overflow flag also signals that the result is greater then - 128d or + 127d

Multiple byte Arithmetic

+32767d

+00004d

+87654d

+78659d

ADDC A, source byte

1) Write a pgm to add two 16 bit numbers

```
CLR C
mov A, # 0E7h
ADD A, # 8Dh
mov R6, A
mov A, # 3Ch
ADDC A, #3Bh
 mov R7, A
```

2) Write a pgm to subtract two 16 bit numbers.

```
CLR C
mov A, # 62h
SUBB A, # 96h
mov R7, A
mov A, # 27h
SUBB a, # 12h
mov R6 , A
```

MUL AB [OV, CY]

(B : A) ← A x B **Always Clears CY**

OV = 1 if Results > FF (not an error, signals that the result is larger than 8-bit)

DIV AB

(A / B) **Quotient in (A)**

Reminder in (B)

Divide by 0 → OV = 1 : Invalid result

DA A **Decimal Adjust after addition**

(CY flag is set if the adjusted No. exceeds 99 BCD & reset otherwise)

1.Add the unsigned numbers found in the internal RAM locations 25h,26h,&27h together & put the result in RAM locations 31(MSB) & 30h(LSB)

```
 Mov 31h,#00h
    Mov A,25h              for BCD numbers
    ADD A,26h      ………… DAA
    Mov R0,A
    Mov A,# 00h
   ADDC A,31h
   Mov 31h,A
   Mov A,R0
   ADD A,27h      ……………..  DAA
   Mov 30h,A
   Mov A,#00h
   ADDC A,31h
   Mov 31h,A
```

2. **Multiply the unsigned number in register R3 by the unsigned number on port 2 & put the result in external RAM locations 10h(MSB) & 11h(LSB)**

```
Mov  A,0A0h
Mov 0F0h,R3
MUL AB
mov R0, #11h
Movx @R0,A
DEC R0
Mov A,0F0H
Movx @R0,A
```

3. Write a pgm to get a byte of hex data from P1 & convert it to decimal.

```
mov A, # 0FFh
        mov P1, A
        mov A, P1
        mov B, # 0Ah
              DIV AB
        mov R7, B
        mov B, # 0Ah
        DIV AB
        mov R6, B
        mov R5, A
```

Branching Instructions

Jump & CALL instructions

- These can replace the contents of PC with a new program address
- The difference in bytes of the present & the new address is called the RANGE.

There are 3 ranges:

- Relative range (+127 bytes to -128 bytes)
- Absolute range (2K byte pages)
- Long range (from 0000h to FFFF h)

Absolute range may be divided into a series of pages of any convenient binary size such as 256bytes, 2K, 4K, and so on….

In 8051 it has 2K page size giving a total of 32 pages.

The upper 5 bits of the PC hold the page number & lower 11 bits hold the address with in each page.

page	address (HEX)
00	0000 - 07FF
01	0800 – 0FFF
02	1000 – 17FF
.	.
.	.
1F	F800 – FFFF

Conditional Jumps

BIT JUMPS

- JC radd
- JNC radd
- JB bit, radd (Jump if direct Bit is set)
- JNB bit, radd (Jump if direct Bit is Notset)
- JBC bit, radd (Jump if direct Bit is set & Clear it)

BYTE JUMPS

- **JZ radd** **Jump if Acc = 00 (no zero flag)**
- **JNZ radd**

C for Compare : D for Decrement

1. CJNE Rn, # data, rel

   Compare immediate data to Register : Jump if not equal
2. CJNE @Ri, # data, rel

   Compare immediate data to indirect : Jump if not equal
3. CJNE A, # data, rel
4. CJNE A, direct, rel

5. DJNZ Rn, rel **Dec Rn : Jump if it's not 0**
6. DJNZ direct, rel **Dec direct : Jump if not 0**

ALL CONDITIONAL JUMPS ARE relative JUMPS

Unconditional Jumps

- Do not test any bit or byte

JMP @ A + DPTR **Jump indirect relative to DPTR**

(PC) 🡐 (A) + (DPTR) **Sources are unaltered**

16 bit addition

AJMP sadd **Absolute Jump within the 2K space**

LJMP ladd Long Jump to anywhere in the 64K memory space

SJMP radd

CALLS AND SUBROUTINES

ACALL   sadd

LCALL   ladd

There are NO Conditional CALLS in 8051

RET	Return from the subroutine
RETI	Return from the Interrupt
NOP	**No Operation**

Example programs

1. Place any number in internal RAM location 3Ch and increment it until the number equals 2Ah

```
One: CLR C                              One:  INC 3Ch
        mov A, #2Ah                            mov A, #2Ah
        SUBB A, 3Ch        OR                  CJNE A, 3Ch
        JZ done                                NOP
        INC  3Ch
        SJMP  one
Done: NOP
```

2. A number A6h is placed somewhere in external RAM between locations 0100h and 0200h. Find the address of that location and put that address in R6(LSB) & R7(MSB).

```
mov 20h, #0A6h
mov DPTR, # 00FFh
Back: INC DPTR
movX a, @DPTR
CJNE a, 20h, Back
mov R7, 83h
 mov R6, 82h
```

3. Find the address of the first two internal RAM locations between 20h and 60h which contain consecutive numbers. If so set the carry flag to 1, else clear the flag.

```
          mov 81h, #65h
          mov R0,   #20h
next:     mov A, @R0
            Inc A
            mov 1Fh, A
            Inc R0
            Acall Done
            JNC Through
            mov A, @ R0
            CJNE  A, 1Fh, next
            SETB 0D7h
Through: Sjmp through
   Down  : CLR C
               mov A, # 61h
                XRL A, R0
              JNZ  Back
              RET
Back :  CPL C
               RET
```

4. **Assume that RAM location 40-44 have the values 7D,EB,C5,5B & 3Ch respectively. Write a program to find the sum of the values.**

```
        mov R0, # 40h
        mov R2, # 5
        CLR A
        mov R7 ,A
Again:  ADD A, @ R0
        JNC next
        INC R7
next :  INC R0
        DJNZ R2, Again
```

5. Write a pgm to find the sum of the 10 BCD numbers stored in RAM locations storing at 40h.

```
            mov R0, #40h
                   mov R2, #0Ah
                    CLR A
                   mov R7, A
        Again:     ADD A, @R0
                    DA A
                    JNC Next
                    INC R7
    Next :          INC R0
                     DJNZ R2 , Again
```

6) Write a pgm that finds the number of 1's in a given byte.

```
        mov P1, # 0
              mov R7, # 8
              mov A, # 97h
Again: RLC A
              JNC Next
              INC R1
Next  :  DJNZ R7, Again
```

7) Write a pgm to add 3 to the accumulator 10 times

```
              mov A, # 0
              mov R2, # 10
Again: ADD A, # 03
             DJNZ R2, Again
            mov R5, A
```

8. Find the sum of the values 79h,F5h & E2h, put the sum in registers R0 & R5.

```
        mov A, # 00
              mov R5, A
              ADD  A, # 79h
              JNC N1
               INC R5
N1:            ADD A, # 0F5h
               JNC N2
               INC R5
N2:             ADD A, # 0E2h
                JNC over
                INC R5
over:           mov R0, A
```

9. **A washing machine is designed for a voltage range of 180 - 240v. If the voltage is above 240v or below 180v, the washing machine will shut down by turning OFF a relay connected to P1.0. Assume that the voltage can be read at port 0 in the range 0-255v. Write a pgm to implement this operation.**

```
        . ORG 100h
      Input:  mov P0, # 0FFh
              mov A, P0
              SUBB A, # 180
              JC off
             mov A, P0
             SUBB A, # 240
              JNC off
             SJMP input
             Off   : CLR P1.0
             SJMP input
```

10. Write a pgm to separate an 8bit 2’s complement number into magnitude & sign bit.

```
          mov P0. #00h
          mov P2, #00h
          mov R0, # 0FEh
          mov A, R0
         JB Acc.7, convert   ……. Check the sign bit
         mov A, R0
DBP:      mov P0, A
Loop:     SJMP loop
Convert: SETB P2.0
              CPL A
              INC A
              SJMP DBP
```

11. Write a pgm to find the square root of a number.

Program:

```
             mov R3, # 36
             mov R0, # 00h
              mov R1, # 01h
Loop1:  CLR C
              mov A, R3
              SUBB A, R1
               mov R3, A
               JNC square
               mov A, R0
               mov P0, A
  loop:     SJMP loop
Square:  INC R0
                mov A, R1
                ADD A, # 02h
                mov R1, A
                SJMP loop1
```

Logic:

R0	R1	N=N-odd number
0	1	36-1=35
1	3	35-3=32
2	5	27
3	7	20
4	9	11
5	11	00
6	13	00-13=-13

8051Instruction Set

By Dr. Naveen B

Instruction Set

Data Transfer Instructions

MOVE   Destination, Source

MOV   Rn, A

MOV   Rn, direct

MOV DPTR, # data 16
MOVC A, @ A + DPTR
MOVC A, @ A+PC

MOVX   A, @DPTR

MOVX   @DPTR, A

MOVX   A, @Ri

**** No flags are affected**

Logical Operations

Byte level Logical operations

The operations are done in each individual bit of the source & destination bytes.

ANL **A , Rn**

ORL **A , @Rp**

XRL **A , #27h**

- **Destination is Accumulator or direct addressing & source may be any addressing mode.**
- **Use port as a source but not as a destination**

CLR	A	**Clear Acc**
CPL	A	**Complement Acc**

** No flags are affected

Bit level Logical Operations

- It is very convenient to alter a single bit of a byte.

CPL C (Complement)

CPL bit

ANL C, bit AND direct bit to CY

ANL C, /bit

ORL C, bit

ORL C, /bit

EX:SETB 00h …Bit 0 of RAM byte 20h = 1

ANL C, /00h …C=0; bit 0 of RAM byte 20h =1

BOOLEAN OPERATIONS

CLR C	**Clear CY**
CLR bit	**Clear direct bit**
SETB C	**Set CY**
SETB bit	**Set direct bit (SETB P2.4)**
MOV C, bit	**Move direct bit to CY**
MOV bit, C	**Move CY to direct bit**
MOV 7Fh, C	**…. Bit 7 of RAM byte 2Fh =1..Assume C=1**

Note:

- CLR A --- it is for byte & only 'A' reg. no other registers or addressing modes

CLR Acc.0 for individual bit in 'A' reg.

CLR b bits in SFRs & bit addressable area.

Rotate and Swap Operations

RL A	**Rotate Acc Left**	**:**	**$b_{n+1} \leftarrow b_n$**
RR A	**Rotate Acc Right**	**:**	**$b_n \leftarrow b_{n+1}$**
RLC A	**Rotate Acc Left through Carry**		
RRC A	**Rotate Acc Right through Carry**		

SWAP A **Exchange between the nibbles**

* Only CY flag is affected in RRC A & RLC A

Example programs:

1) Double the number in Reg. R2 & put the result in R3 & R4.

```
CLR C
MOV A, R2
RLC A
MOV R4, A
CLR A
RLC A
MOV R3, A
```

2) OR the contents of ports 1 & 2, put the result in external RAM location 0100h

```
MOV A, 90h
ORL A, 0A0h
MOV DPTR, #0100h
MOVX @DPTR, A
```

3) Configure P1 to read switches at P1.0 & P1.1. If P1.0 is high, turn ON a relay connected to P2.5 by sending a logic high o/p. If P1.0 is low, clear P2.5. If the status of the switch at P1.1 is high, turn OFF the relay connected to P2.6 by sending a logic low o/p. If P1.1 is low, set P2.6 to high state.

Logic:

i/p's at P1	o/p's at P2
XXXXXX00	X10XXXXX
XXXXXX01	X11XXXXX
XXXXXX10	X00XXXXX
XXXXXX11	X01XXXXX

Program:

```
mov p1, #0FFh
 Mov p2, #00h
 Mov A, p1
 Anl A, #03h
 Cpl Acc.1
 Rl a
 Rl a
 Rl a
 Rl a
 Rl a
 Mov p2, A
```

4) Swap every even numbered bit of register R3 in bank 0 woth the odd numbered bit to its left. Swap bit 0 with bit 1, bit 2 with bit 3, and so on until bit 6 is swapped with bit 7

```
MOV A, R3
RL A
ANL A, #0AAh
PUSH 0E0h
MOV A, R3
RR A
ANL A, #55h
MOV R3, A
POP 0E0h
ORL 03h, A       (ORL R3, A  …. Not allowed)
```

5. Assume that bit P2.2 is used to control an outdoor light and bit P2.5 a light inside a building .Show how to turn on the outside light & turn off the inside one.

```
SETB C
        ORL C, P2.2
        mov P2.2, C
        CLR C
        ANL C, P2.5
        mov P2.5 ,C
```

6. Assume that registers A has packed BCD. Write a pgm to convert packed BCD to two ASCII numbers & place them in R2 & R6.

```
mov A, # 29h
        mov R2, A
        ANL A, # 0Fh
        ORL A, #30h
        mov R6, A
        mov A, R2
        ANL A, # 0F0h
        RRA
         RRA
         RRA
         RRA
         ORL A, # 30h
          mov R2, A
```

Arithmetic Instructions

<u>Incrementing and decrementing</u>

INC A ; operand may be any addressing except Immediate

INC DPTR

DEC A :operand may be any addressing except Immediate

There is no "DEC DPTR"

- No flags are affected

Addition and subtraction:

- ADD A, Source byte [OV, AC, CY]
- ADDC A, Source byte [OV, AC, CY]
- SUBB A, source byte [OV, AC, CY]

Subtract with borrow:

(A) ← (A) - source byte -CY

‘A’ register is the destination, & source may be any addressing mode

Examples

Unsigned addition: This make use of the carry flag to detect when the result of an ADD operation is a number larger than FF h.

```
                00 to 255d
   95d    =   01011111 b           =5Fh
  189d    =   10111101 b           =BDh
  284d     100011100 b              11Ch
```

Signed addition: If unlike signed numbers are added, then it is not possible for the result to be larger than -128 d to +127 d, and the result will always be correct.

```
Ex1:   -001d    = 11111111 b                    = FF h
       +027d   = 00011011 b                    = 1Bh
       +026d   100011010 b  =+ 026d          11Ah
```

Adding two +ve numbers, result may exceed +128d

```
Ex2:   +100d      =01100100b                =64h
       +050d      = 00110010b                32h
        150d       010010110b  = -106d      096h      correct result= +22d
                                                 OV=1

Ex3:   +045d      =00101101b              =2Dh
       +075d      =01001011b              =4Bh
       +120d       001111000b =120d       078h   OV=0 (result not exceeded)
```

The result of adding two –ve numbers together for a sum that does not exceed the –ve limit.

```
Ex:  - 030d     = 11100010b        = E2h
     - 050d       11001110b        = CEh
     - 080d      110110000b          1B0h

          OV=0
```

Adding Two –ve numbers whose sum does exceed -128d

- 070d	= 10111010b		= BA h
- 070d	= 10111010b		= BAh
- 140d	=101110100b	= +116d	174h

OV= 1 (correct result= -12d)

Flags		Action
CY	OV	
0	0	none
0	1	compliment the sign
1	0	none
1	1	compliment the sign

<u>**Unsigned subtraction**:</u>

Subtraction of a larger number from a smaller number.

```
 015d       = 00001111b   = 0F0h
 100d       = 01100100b   = 064h
- 085d      =110101011b
```

The carry flag is set to 1 & OV=0.

2’s compliment of the result = 085d.

```
 100d   =  01100100b     = 64h
 015d   =  00001111b     = 0Fh
 085d   =001010101b      =055h
```

C=0, OV=0 (Magnitude of the result is in true form).

Signed subtraction:

When numbers of like sign are subtracted it is impossible for the result to exceed positive or negative magnitude limits of +127 or -128.

```
   +100d  =01100100b(carry flag is 0 before
                                        SUBB)=64h
sub +126d  = 01111110b        = 7Eh
    -  026d =111100110b       = 1E6h
              cy=1 , OV=0
```

Using two negative numbers

-061d = 11000011b (CY=0 before SUBB) = C3h

Subb $\underline{\text{-116d}}$ = $\underline{\text{10001100}}$b = $\underline{\text{8C}}$h

+055d 000110111b 037h

CY=0, OV=0

An overflow is possible when subtracting numbers of opposite sign

```
 - 099d  =10011101b(cy=0 before SUBB)   =9Dh
+100d  =01100100b                       =64h
-199d  =000111001b = +057d              =039h
```

OV= 1 , cy =0

Because the overflow flag is set to 1, the result must be adjusted so that 2's compliment is 71d

```
+ 087d   = 01010111b(cy=0 before SUBB)  = 57h
- 052d    = 11001100b                     = cch
-------------------------------------------------
+139d    =110001011b = -177d             =18bh
```

OV=CY=1

The magnitude can be interpreted as +011d

The general rule is that if the overflow flag is set to 1, then compliment the sign bit . The overflow flag also signals that the result is greater then - 128d or + 127d

Multiple byte Arithmetic

+32767d

+00004d

+87654d

+78659d

ADDC A, source byte

1) Write a pgm to add two 16 bit numbers

```
CLR C
mov A, # 0E7h
ADD A, # 8Dh
mov R6, A
mov A, # 3Ch
ADDC A, #3Bh
 mov R7, A
```

2) Write a pgm to subtract two 16 bit numbers.

```
CLR C
mov A, # 62h
SUBB A, # 96h
mov R7, A
mov A, # 27h
SUBB a, # 12h
mov R6 , A
```

MUL AB [OV, CY]

(B : A) ← A x B **Always Clears CY**

OV = 1 if Results > FF (not an error, signals that the result is larger than 8-bit)

DIV AB

(A / B) **Quotient in (A)**

Reminder in (B)

Divide by 0 → OV = 1 : Invalid result

DA A **Decimal Adjust after addition**

(CY flag is set if the adjusted No. exceeds 99 BCD & reset otherwise)

1. Add the unsigned numbers found in the internal RAM locations 25h,26h,&27h together & put the result in RAM locations 31(MSB) & 30h(LSB)

```
Mov 31h,#00h
   Mov A,25h              for BCD numbers
   ADD A,26h     ………… DAA
   Mov R0,A
   Mov A,# 00h
  ADDC A,31h
  Mov 31h,A
  Mov A,R0
  ADD A,27h     ……………..  DAA
  Mov 30h,A
  Mov A,#00h
  ADDC A,31h
  Mov 31h,A
```

2. **Multiply the unsigned number in register R3 by the unsigned number on port 2 & put the result in external RAM locations 10h(MSB) & 11h(LSB)**

```
Mov  A,0A0h
Mov 0F0h,R3
MUL AB
mov R0, #11h
Movx @R0,A
DEC R0
Mov A,0F0H
Movx @R0,A
```

3. Write a pgm to get a byte of hex data from P1 & convert it to decimal.

```
mov A, # 0FFh
      mov P1, A
      mov A, P1
      mov B, # 0Ah
            DIV AB
      mov R7, B
      mov B, # 0Ah
      DIV AB
      mov R6, B
      mov R5, A
```

Branching Instructions

Jump & CALL instructions

- These can replace the contents of PC with a new program address
- The difference in bytes of the present & the new address is called the RANGE.

There are 3 ranges:

- Relative range (+127 bytes to -128 bytes)
- Absolute range (2K byte pages)
- Long range (from 0000h to FFFF h)

Absolute range may be divided into a series of pages of any convenient binary size such as 256bytes, 2K, 4K, and so on….

In 8051 it has 2K page size giving a total of 32 pages.

The upper 5 bits of the PC hold the page number & lower 11 bits hold the address with in each page.

page	address (HEX)
00	0000 - 07FF
01	0800 – 0FFF
02	1000 – 17FF
.	.
.	.
1F	F800 – FFFF

Conditional Jumps

BIT JUMPS

- JC radd
- JNC radd
- JB bit, radd (Jump if direct Bit is set)
- JNB bit, radd (Jump if direct Bit is Notset)
- JBC bit, radd (Jump if direct Bit is set & Clear it)

BYTE JUMPS

- **JZ radd** **Jump if Acc = 00 (no zero flag)**
- **JNZ radd**

C for Compare : D for Decrement

1. CJNE Rn, # data, rel
   Compare immediate data to Register : Jump if not equal
2. CJNE @Ri, # data, rel
   Compare immediate data to indirect : Jump if not equal
3. CJNE A, # data, rel
4. CJNE A, direct, rel
5. DJNZ Rn, rel **Dec Rn : Jump if it's not 0**
6. DJNZ direct, rel **Dec direct : Jump if not 0**

ALL CONDITIONAL JUMPS ARE relative JUMPS

Unconditional Jumps

- Do not test any bit or byte

JMP @ A + DPTR **Jump indirect relative to DPTR**

(PC) ← (A) + (DPTR) **Sources are unaltered**

16 bit addition

AJMP sadd **Absolute Jump within the 2K space**

LJMP ladd Long Jump to anywhere in the 64K memory space

SJMP radd

CALLS AND SUBROUTINES

ACALL sadd

LCALL ladd

There are NO Conditional CALLS in 8051

RET	Return from the subroutine
RETI	Return from the Interrupt
NOP	**No Operation**

Example programs

1. Place any number in internal RAM location 3Ch and increment it until the number equals 2Ah

```
One: CLR C                          One:  INC 3Ch
      mov A, #2Ah                          mov A, #2Ah
      SUBB A, 3Ch        OR                CJNE A, 3Ch
      JZ done                              NOP
      INC  3Ch
      SJMP  one
Done: NOP
```

2. A number A6h is placed somewhere in external RAM between locations 0100h and 0200h. Find the address of that location and put that address in R6(LSB) & R7(MSB).

```
mov 20h, #0A6h
mov DPTR, # 00FFh
Back: INC DPTR
movX a, @DPTR
CJNE a, 20h, Back
mov R7, 83h
 mov R6, 82h
```

3. Find the address of the first two internal RAM locations between 20h and 60h which contain consecutive numbers. If so set the carry flag to 1, else clear the flag.

```
              mov 81h, #65h
              mov R0,   #20h
       next:    mov A, @R0
                Inc A
                mov 1Fh, A
                Inc R0
                Acall Done
                JNC Through
                mov A, @ R0
                CJNE  A, 1Fh, next
                SETB 0D7h
       Through: Sjmp through
         Down  : CLR C
                    mov A, # 61h
                     XRL A, R0
                   JNZ  Back
                   RET
       Back :  CPL C
                    RET
```

4. Assume that RAM location 40-44 have the values 7D,EB,C5,5B & 3Ch respectively. Write a program to find the sum of the values.

```
        mov R0, # 40h
        mov R2, # 5
        CLR A
        mov R7 ,A
Again:  ADD A, @ R0
        JNC next
        INC R7
next :  INC R0
        DJNZ R2, Again
```

5. Write a pgm to find the sum of the 10 BCD numbers stored in RAM locations storing at 40h.

```
        mov R0, #40h
                mov R2, #0Ah
                 CLR A
                mov R7, A
    Again:      ADD A, @R0
                 DA A
                 JNC Next
                 INC R7
Next :           INC R0
                  DJNZ R2 , Again
```

6) Write a pgm that finds the number of 1's in a given byte.

```
    mov P1, # 0
             mov R7, # 8
             mov A, # 97h
Again: RLC A
             JNC Next
             INC R1
Next  :  DJNZ R7, Again
```

7) Write a pgm to add 3 to the accumulator 10 times

```
             mov A, # 0
             mov R2, # 10
Again: ADD A, # 03
            DJNZ R2, Again
           mov R5, A
```

8. Find the sum of the values 79h,F5h & E2h, put the sum in registers R0 & R5.

```
            mov A, # 00
                mov R5, A
                ADD  A, # 79h
                JNC N1
                 INC R5
      N1:        ADD A, # 0F5h
                 JNC N2
                 INC R5
      N2:         ADD A, # 0E2h
                  JNC over
                  INC R5
      over:       mov R0, A
```

9. **A washing machine is designed for a voltage range of 180 - 240v. If the voltage is above 240v or below 180v, the washing machine will shut down by turning OFF a relay connected to P1.0. Assume that the voltage can be read at port 0 in the range 0-255v. Write a pgm to implement this operation.**

```
        . ORG 100h
      Input:  mov P0, # 0FFh
              mov A, P0
              SUBB A, # 180
              JC off
             mov A, P0
             SUBB A, # 240
              JNC off
             SJMP input
             Off   : CLR P1.0
             SJMP input
```

10. Write a pgm to separate an 8bit 2’s complement number into magnitude & sign bit.

```
         mov P0. #00h
         mov P2, #00h
         mov R0, # 0FEh
         mov A, R0
        JB Acc.7, convert   ……. Check the sign bit
        mov A, R0
DBP:      mov P0, A
Loop:     SJMP loop
Convert: SETB P2.0
              CPL A
              INC A
              SJMP DBP
```

11. Write a pgm to find the square root of a number.

Program:

```
             mov R3, # 36
             mov R0, # 00h
              mov R1, # 01h
Loop1:  CLR C
              mov A, R3
              SUBB A, R1
               mov R3, A
               JNC square
               mov A, R0
               mov P0, A
  loop:      SJMP loop
Square:  INC R0
                mov A, R1
                ADD A, # 02h
                mov R1, A
                SJMP loop1
```

Logic:

R0	R1	N=N-odd number
0	1	36-1=35
1	3	35-3=32
2	5	27
3	7	20
4	9	11
5	11	00
6	13	00-13=-13

8051 Interfacing & Applications

By Dr. Naveen B

LCD Interfacing

Liquid Crystal Displays (LCDs)

- cheap and easy way to display text
- Various configurations (1 line by 20 X char up to 8 lines X 80)
- Integrated controller
- The display has two register
  - data register
  - Command code register
- By RS you can select register
- Data lines (DB7-DB0) used to transfer data and commands

Pin Description

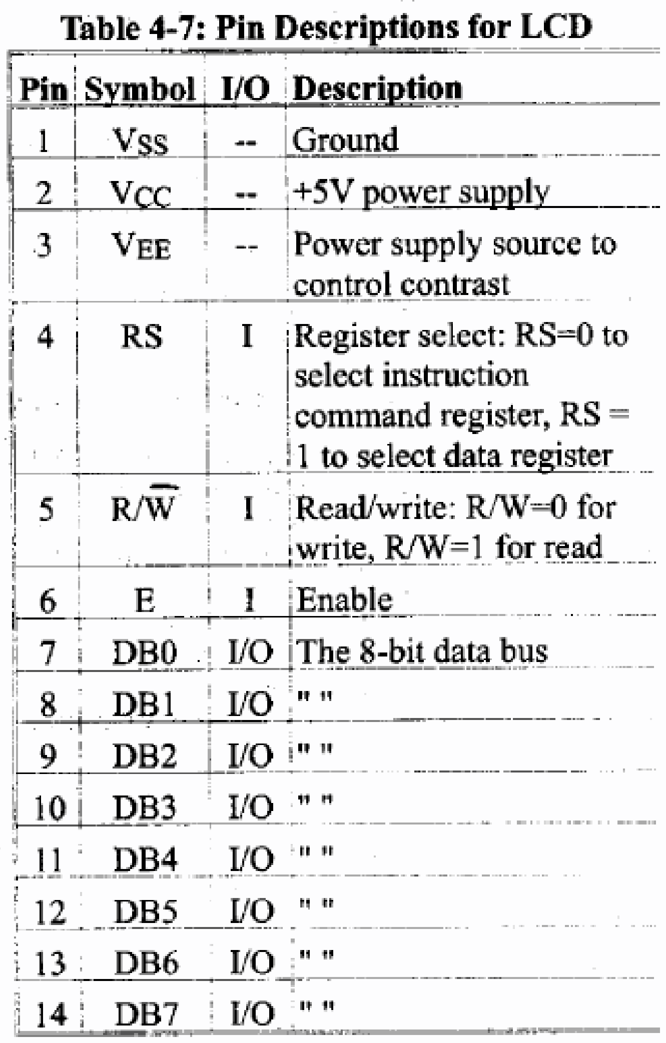


Table 4-7: Pin Descriptions for LCD

Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5V power supply
3	V_{EE}	--	Power supply source to control contrast
4	RS	I	Register select: RS=0 to select instruction command register, RS = 1 to select data register
5	$R/\overline{W}$	I	Read/write: R/W=0 for write, R/W=1 for read
6	E	I	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	" "
9	DB2	I/O	" "
10	DB3	I/O	" "
11	DB4	I/O	" "
12	DB5	I/O	" "
13	DB6	I/O	" "
14	DB7	I/O	" "

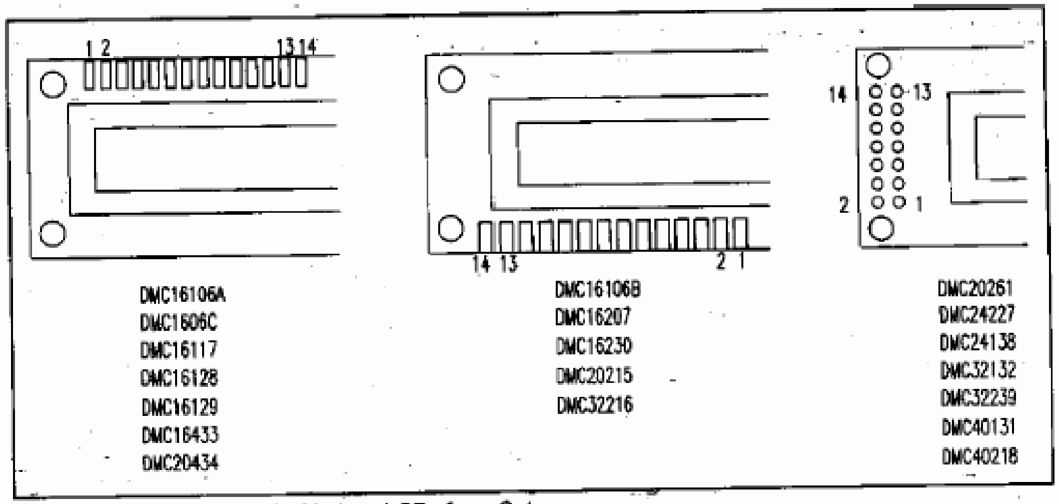
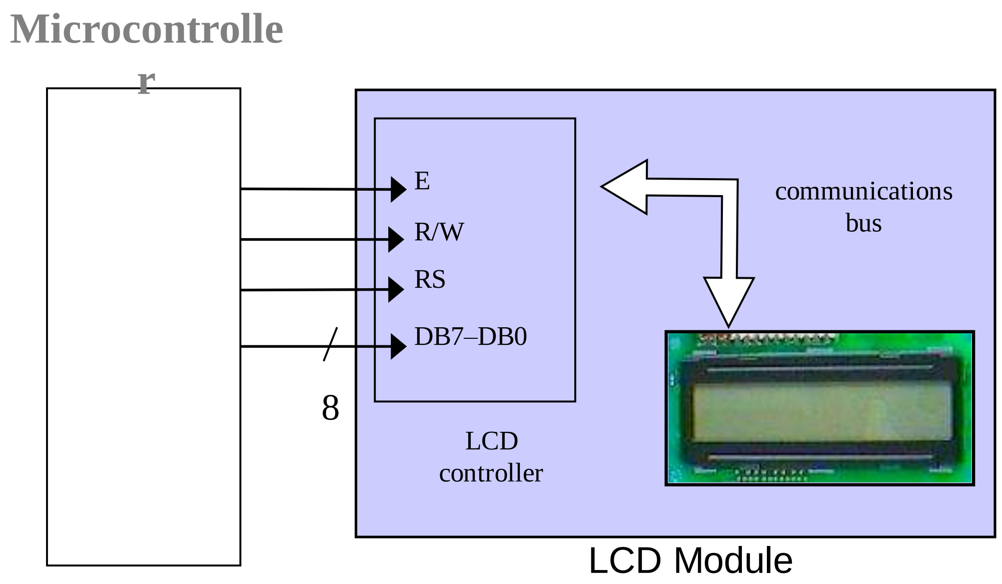


Figure 4-34. Pin Positions for Varĭous LCDs from Optrex

Alphanumeric LCD Interfacing



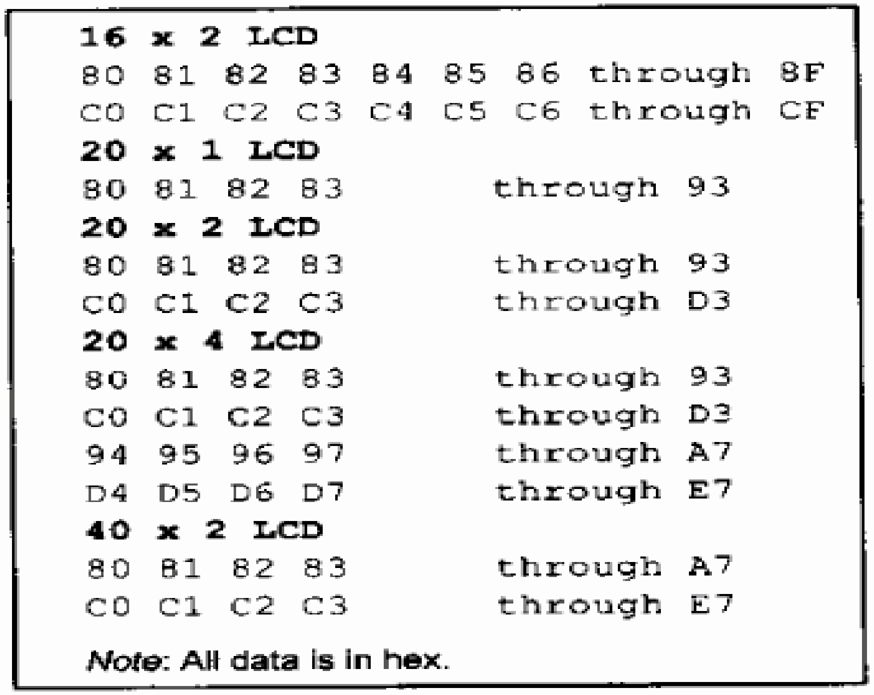
- Pinout
  - 8 data pins D7 - D0
  - RS: Data or Command Register Select
  - R/W: Read or Write
  - E: Enable (Latch data)
- RS – Register Select
  - RS = 0 → Command Register
  - RS = 1 → Data Register
- R/W = 0 → Write , R/W = 1 → Read
- E – Enable
  - Used to latch the data present on the data pins.
- D0 – D7
  - Bi-directional data/command pins.
  - Alphanumeric characters are sent in ASCII format.

LCD Commands

- The LCD's internal controller can accept several commands and modify the display accordingly. Such as:
  - Clear screen
  - Return home
  - Decrement/Increment cursor

- After writing to the LCD, it takes some time for it to complete its internal operations. During this time, it will not accept any new commands or data.
  - We need to insert time delay between any two commands or data sent to LCD

LCD Addressing

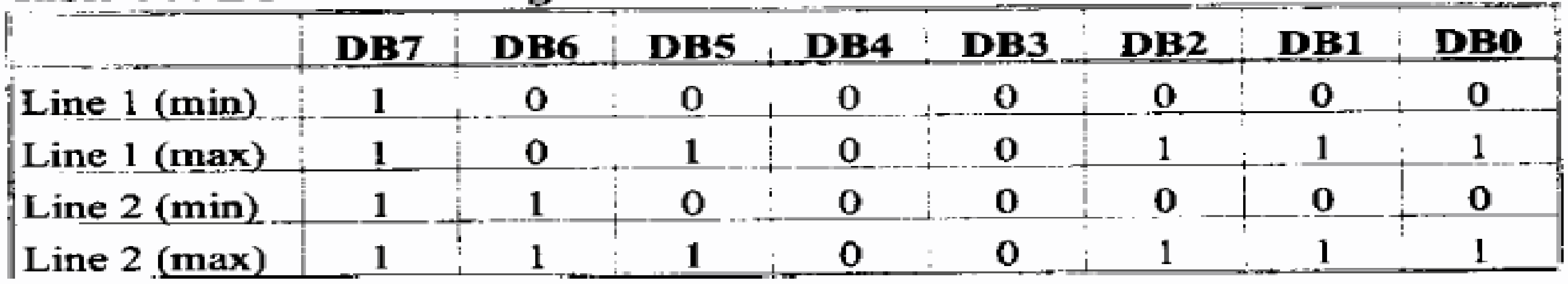


```
16 x 2 LCD
80 81 82 83 84 85 86 through 8F
C0 C1 C2 C3 C4 C5 C6 through CF
20 x 1 LCD
80 81 82 83          through 93
20 x 2 LCD
80 81 82 83          through 93
C0 C1 C2 C3          through D3
20 x 4 LCD
80 81 82 83          through 93
C0 C1 C2 C3          through D3
94 95 96 97          through A7
D4 D5 D6 D7          through E7
40 x 2 LCD
80 81 82 83          through A7
C0 C1 C2 C3          through E7
```

Note: All data is in hex.

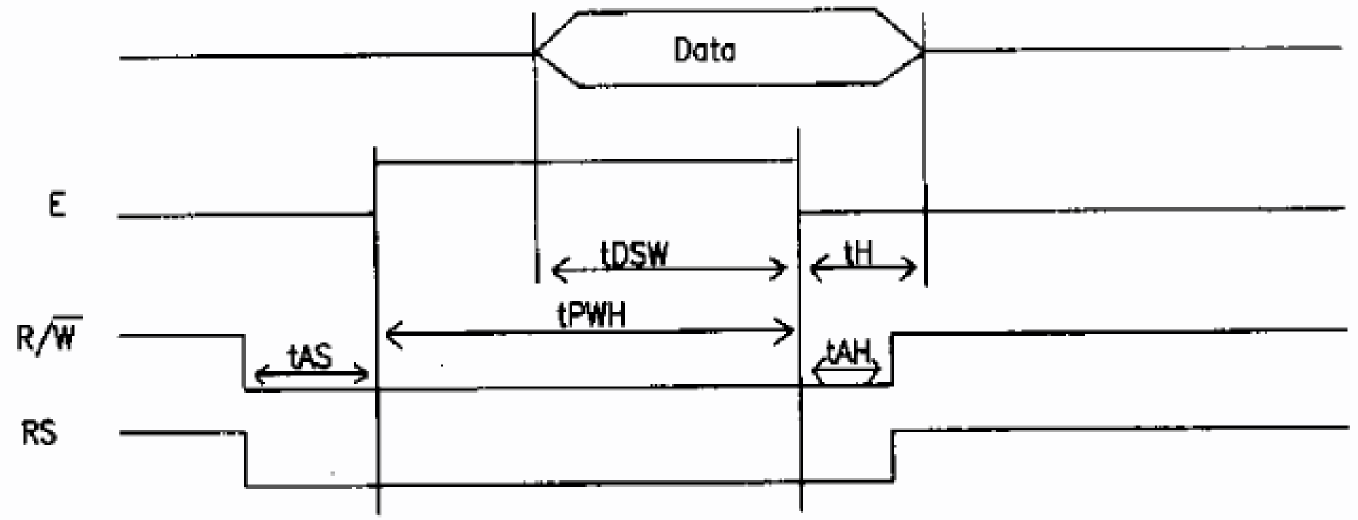
Figure 4-36. Cursor Addresses for Some LCDs

Table 4-9: LCD Addressing



	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Line 1 (min)	1	0	0	0	0	0	0	0
Line 1 (max)	1	0	1	0	0	1	1	1
Line 2 (min)	1	1	0	0	0	0	0	0
Line 2 (max)	1	1	1	0	0	1	1	1

LCD Timing



tPWH = Enable pulse width = 450 ns (minimum)
tDSW = Data set up time = 195 ns (minimum)
tH = Data hold time = 10 ns (minimum)
tAS = Set up time prior to E (going high) for both RS and R/W = 140 ns (minimum)
tAH = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

Figure 4-37. LCD Timing

Command Codes

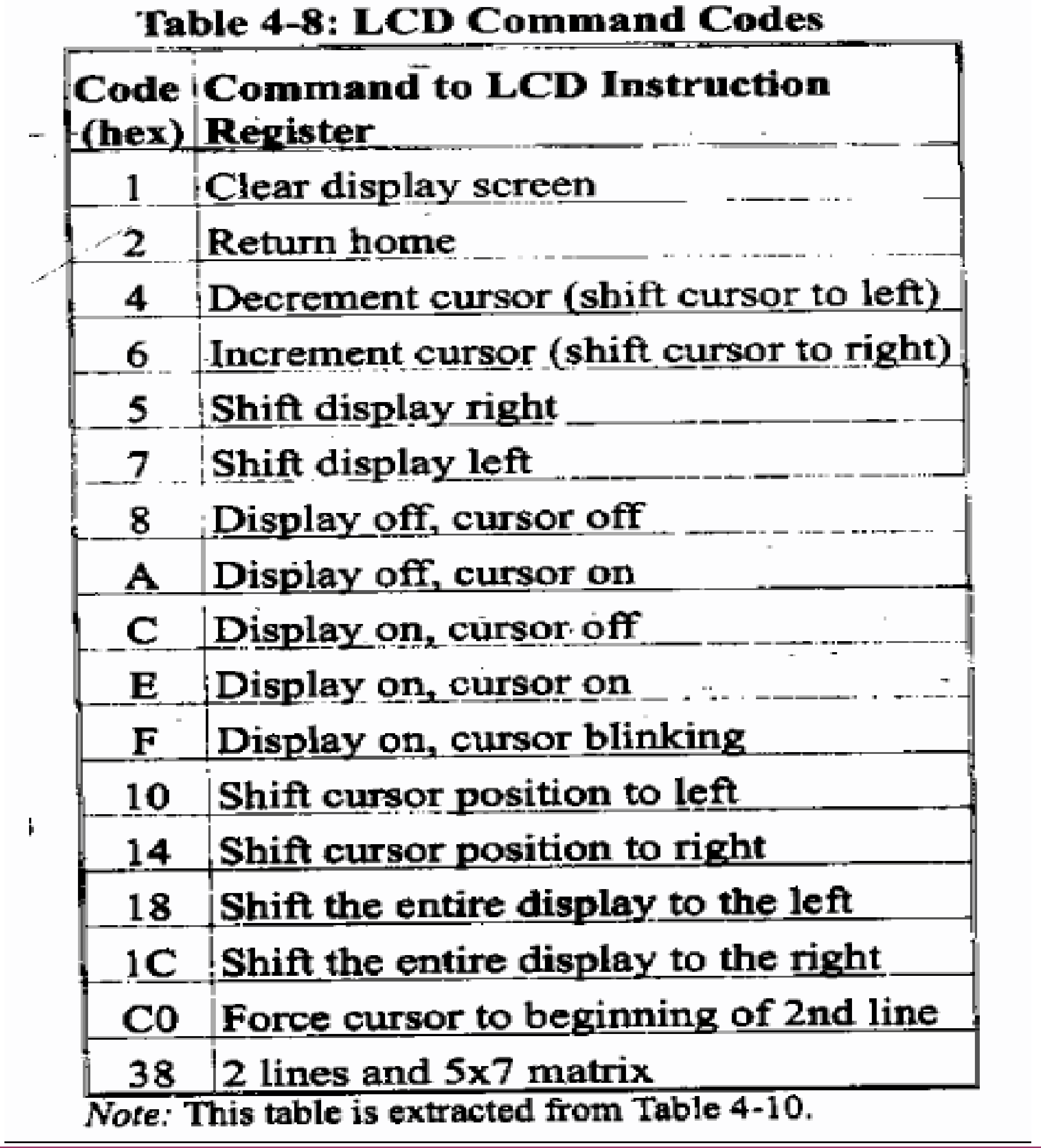


Table 4-8: LCD Command Codes

Code (hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor on
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

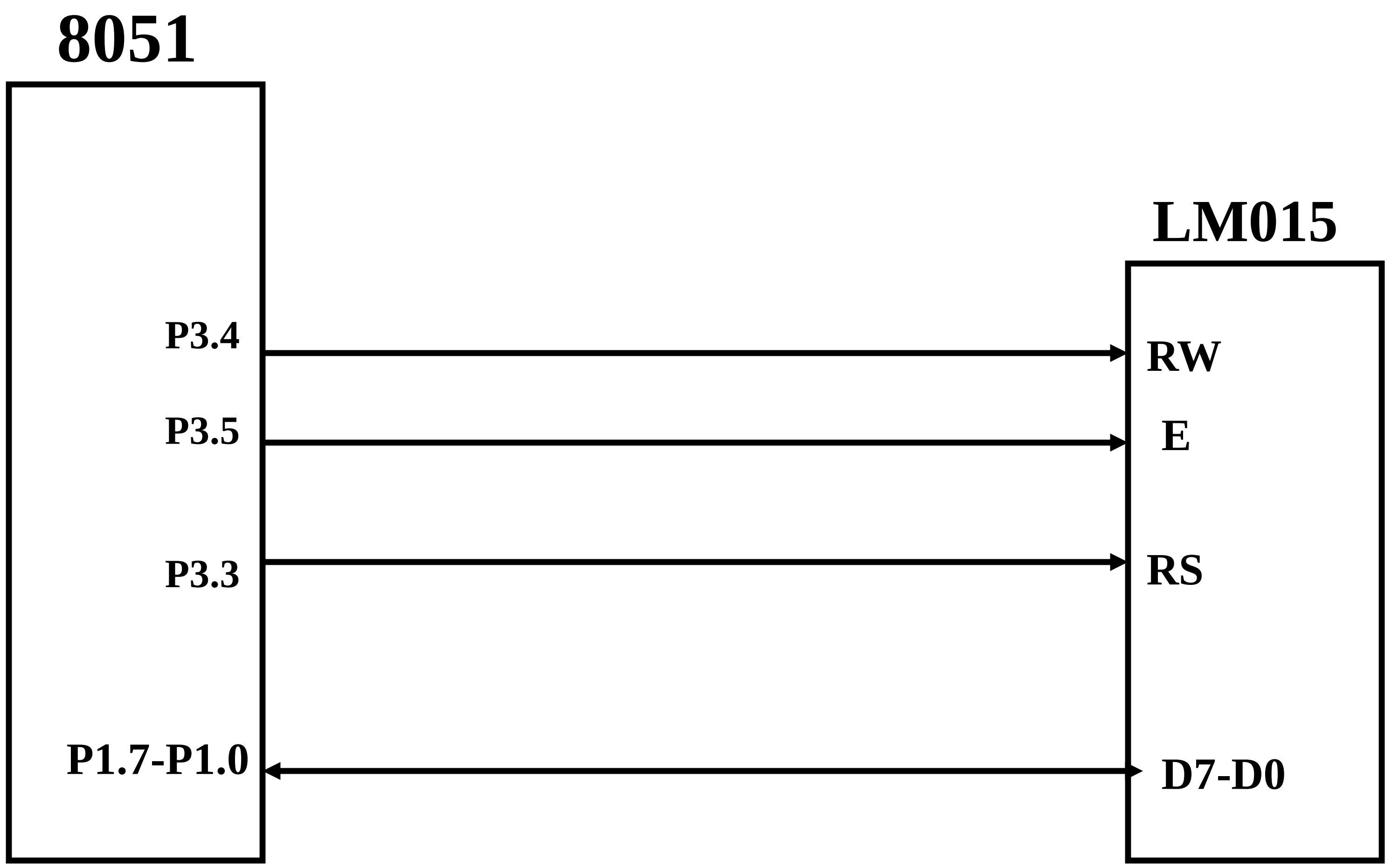
Note: This table is extracted from Table 4-10.

Table 4-10: List of Instructions (Courtesy of Optrex Corporation)

Instruction	Code: RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max)
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter.	1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	--	Sets DD RAM address 0 as address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40 µs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of entire display (D), cursor ON/OFF (C), and blink of cursor position character (b).	40 µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	--	--	Moves cursor and shifts display without changing DD RAM contents.	40 µs
Function Set	0	0	0	0	1	DL	N	F	--	--	Sets interface data length (DL), number of display lines (L) and character font (F).	40 µs
Set CG RAM Address	0	0	0	1	AGC						Sets CG RAM address. CG RAM data is sent and received after this setting.	40 µs
Set DD RAM Address	0	0	1	ADD							Sets DD RAM address. DD RAM data is sent and received after this setting.	40 µs
Read Busy Flag & Address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	40 µs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40 µs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 µs

- Also use RS=0, to check the busy flag bit to see if the LCD is ready to receive information
- Busy flag is D7
- When R/W=1 & RS=0, D7=1(busy flag), the LCD is busy taking care of internal operations and will not accept any new information.
- When D7=0, LCD is ready to receive new information

Interfacing LCD with 8051



Example

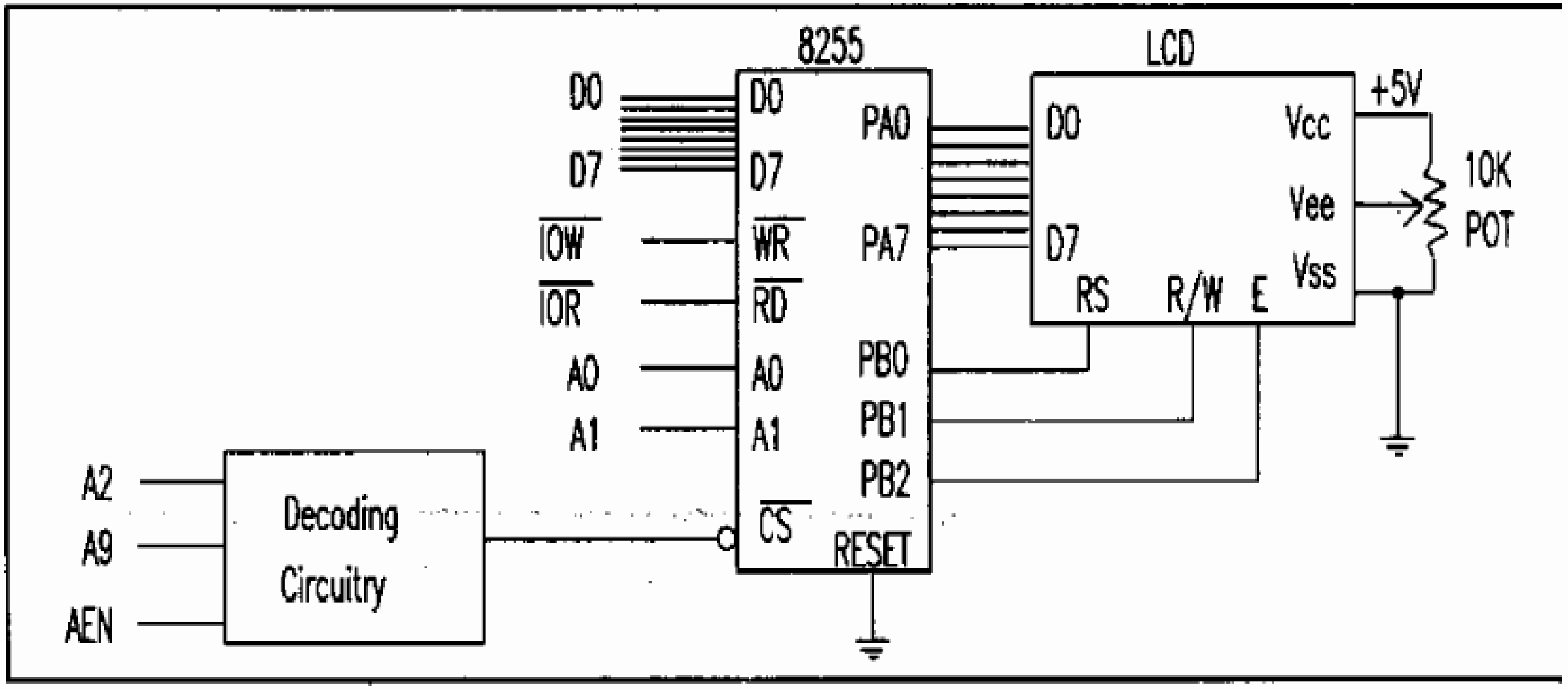


Figure 4-35. 8255-to-PC Interface Connection to LCD

Sending commands and data to LCDs with a time delay

```
;calls a time delay before sending next data/command
; P1.0 – P1.7 are connected to LCD data pins  D0 – D7
;P2.0  is connected to RS pin of LCD
;P2.1 is connected to R/W pin of LCD
;P2.2 is connected to E pin pf LCD

org 0h
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 , cursur on
ACALL COMNWRT            ; call command subroutine
ACALL DELAY              ; give LCD some time
Mov A, #01               ; clear LCD
ACALL COMNWRT            ;  call command subroutine
ACALL DELAY              ; give LCD some time
Mov A, # 06h             ; shift cursor right
ACALL COMNWRT            ; call command subroutine
```

```
          ACALL DELAY             ; give LCD some time
          Mov A, # 84h            ;cursor at line 1 , pos. 4
          ACALL COMNWRT           ; call command subroutine
          ACALL DELAY             ; give LCD some time
          Mov A ,#'N'             ; display letter N
          ACALL DATAWRT           ; call display subroutine
          ACALL DELAY             ; give LCD some time
          MOv A, # 'o'            ;  display letter o
          ACALL DATAWRT           ; call display subroutine
AGAIN: SJMP AGAIN                 ; stay here
COMNWRT :                         ; send command to LCD
          Mov P1, A               ; copy reg A to port1
          CLR P2.0                ; RS=9 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 – L pulse
            RET
DATAWRT:                      ; write data to LCD
            Mov P1, a         ; copy reg A to Port1
            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 – L pilse
            RET
DELAY: Mov R3, #50     ; 50 or higher for fast CPU's
HERE2 :Mov R4 , #255  ;  R4=255
HERE:   DJNZ  R4, HERE      ; stay untill R4 become 0
           DJNZ R3 , HERE2
           RET
           END
```

Sending code or data to the LCD with checking busy flag

; check busy flag before sending data , command to LCD

; P1=data pin , P2.0=Rs , P2.1=R/W, P2.2=E pins

```
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, #01              ; clear LCD command
```

```
ACALL COMMAND    ;  issue command

Mov A, # 06h                ; shift cursor right
ACALL COMMAND    ; issue command
Mov A, # 84h      ;cursor at line 1 , pos. 6
ACALL COMMAND   ; issue command
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 cod
                CLR P2.0     ; RS=0 for command
                CLR P2.1     ; R/W =0 to write to LCD
                SETB P2.2    ; E=1 for H – L pulse
                CLR P2.2     ; E=0 , latch in
                RET
DATA _DISPLAY:
                ACALL DELAY  ; is LCd ready?
                Mov P1, A     ; issue data
```

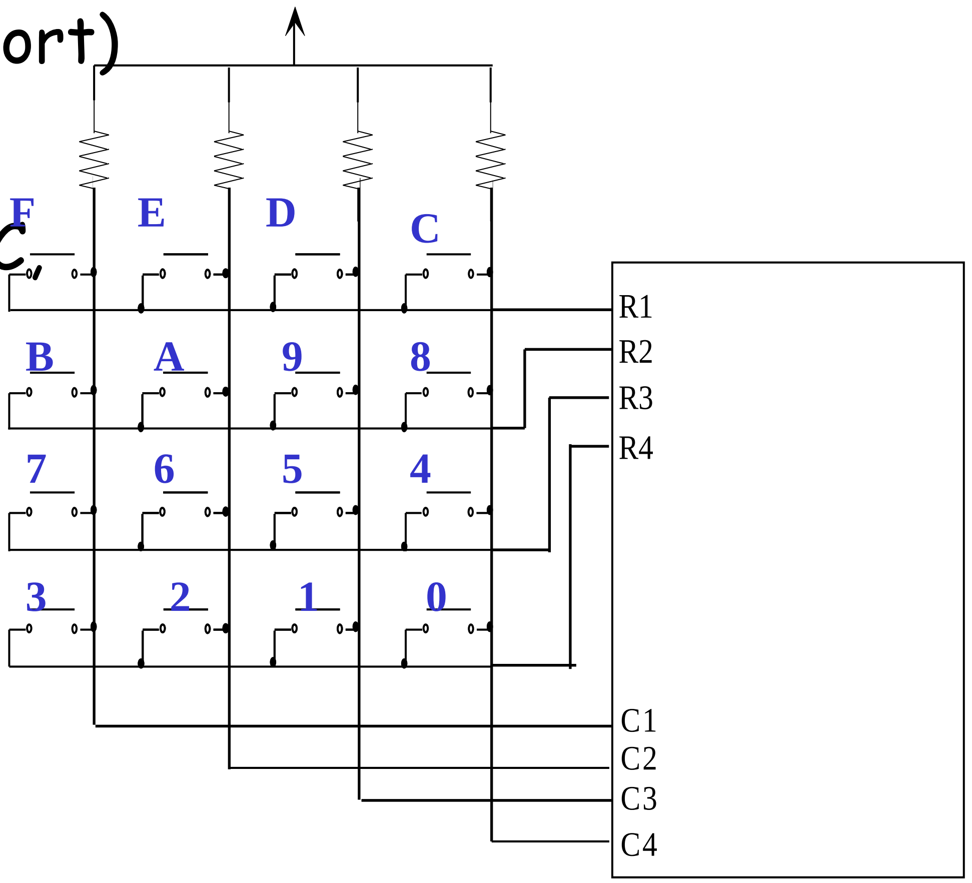
```
        SETB  P2.0          ; RS =1 for data
        CLR P2.1            ; R/W= 0 to write  data to LCD
        SETB P2.2           ; E=1 for H – L pulse
        ACALL DELAY         ; give LCD some time
        CLR P2.2            ; E=0 , latch in
        RET
READY: SETB P1.7            ; make P1.7 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:  CLR P2.2             ; E=0 for L – h pulse
        ACALL DELAY         ; give LCD some time
        SETB P2.2           ; E=1 L – H pulse
        JB P1.7 , BACK      ; stay until busy flag =0
        RET
        END
```

Keyboard interfacing

16 keys arranged as a 4X4 matrix

- Place a 0 on R0 port (i/o Port)
- Read C port (o/p Port)
- C port is connected to VCC,

  ie. port is high
- If there is a 0 bit then the button at the column/row intersection has been pressed.
- Otherwise, try next row
- Repeat constantly



DAC Interfacing

Two methods:

a) Binary weighted

b) R – 2R ladder……higher degree of precision

The number of data bit inputs decides the resolution of the DAC since the number of analog output levels is equal to 2^n

^{n} is the data inputs, common ones are 8, 10 and 12 bits

Example programs

Stepper motor interfacing

- A stepper motor is a widely used device that translates electrical pulses into mechanical movement used for position control.
- The most common stepper motors have four stator windings that are paired with a center tapped common.
- Conventional motor shaft runs freely, the stepper motor shaft moves in a fixed repeatable increment, which allows move it to a precise position.

- The step angle is the minimum degree of rotation associated with a single step.

table 17.4

Steps per second =

(rpm x steps per revolution)/60

EXAMPLE 17.3

Sensor interfacing

- Transducers convert physical data such as temperature, light intensity and speed to electrical signals.
- Depends on the transducer, the o/p produced is in the form of voltage, current, resistance or capacitance

Temperature sensors

Temperature is converted into electrical signals using a transducer called **thermistor**.

A thermistor responds to temperature change by changing resistance

TABLE: 13.8

- Response is not linear
- Complexity associated with writing a software for nonlinear devices.
- This leads to a linear sensors like LM34 & LM35

Converting the common transducers o/p like voltage, current, capacitance, resistance to voltage in order to send i/p to an A- to D converter called **Signal Conditioning**.

Figure 13.21

Example 13.1

ADC interfacing

- In this physical world everything is analog.
- A/D converters translate analog signal to digital
- so that MC can read & process them.
- **Step size** is the smallest change that can be discerned by an ADC
- **Conversion time** is the time it takes the ADC to convert the analog i/p to a digital number.

8051 Interrupts

BY Dr. Naveen B

Definition of 'Interrupt'

Event that disrupts the normal execution of a program and causes the execution of special instructions

Interrupts VS polling

A single MC can serve several devices. There are two ways to do it: a) polling
b) Interrupt

Used in the previous chapter

JNB TF, target -> polling method (wait until the timer rolls over, & we could not do anything else)

Steps in executing an interrupt

- Finish current instruction and saves the PC on stack.
- Jumps to a fixed location in memory depend on type of interrupt
- Starts to execute the interrupt service routine until RETI (return from interrupt)
- Upon executing the RETI the microcontroller returns to the place where it was interrupted. Get pop PC from stack

Interrupt Overheads

Interrupt arrives
Complete current instruction
Save essential register information
Vector to ISR
Save additional register information

Interrupt Latency

Execute body of ISR

Restore other register information
Return from interrupt and restore essential
    registers
Resume task

Interrupt Termination

Interrupt Sources

Original 8051 has 6 sources of interrupts

- Reset
- Timer 0 overflow
- Timer 1 overflow
- External Interrupt 0
- External Interrupt 1
- Serial Port events (buffer full, buffer empty, etc)

Interrupt Vectors

Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.

```
External Interrupt 0:     0003h
Timer 0 overflow:      000Bh
External Interrupt 1: 0013h
Timer 1 overflow:      001Bh
Serial :                    0023h
```

Note: that there are only 8 memory locations between vectors.

ISRs and Main Program in 8051

```
        SJMP   main
                ORG   03H
        ljmp int0sr
        ORG   0BH
        ljmp t0sr
        ORG   13H
        ljmp int1sr
        ORG   1BH
        ljmp t1sr
        ORG   23H
        ljmp serialsr
        ORG   30H
main:     …
        END
```

Figure 23. IE: Interrupt Enable Register

(MSB)							(LSB)
EA	—	ET2	ES	ET1	EX1	ET0	EX0

Enable bit = 1 enables the interrupt.

Enable bit = 0 disables it.

Symbol	Position	Function
EA	IE.7	Global enable/disable. Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	Undefined/reserved.
ET2	IE.5	Timer 2 Interrupt enable bit (AT89C52).
ES	IE.4	Serial Port Interrupt enable bit.
ET1	IE.3	Timer 1 Interrupt enable bit.
EX1	IE.2	External Interrupt 1 enable bit.
ET0	IE.1	Timer 0 Interrupt enable bit.
EX0	IE.0	External Interrupt 0 enable bit.

User software should never write 1s to unimplemented bits, since they may be used in future AT89 Series products.

Enabling and disabling an interrupt

- by bit operation
- Recommended in the middle of program

```
SETB   EA     SETB   IE.7        ;Enable All
SETB   ET0    SETB   IE.1        ;Enable Timer0 ovrf
SETB   ET1    SETB   IE.3        ;Enable Timer1 ovrf
SETB   EX0    SETB   IE.0        ;Enable INT0
SETB   EX1    SETB   IE.2        ;Enable INT1
SETB   ES     SETB   IE.4        ;Enable Serial port
```

- by mov instruction
- Recommended in the first of program

```
MOV IE, #10010110B
```

Programming timer interrupts

JNB TF , target --→polling method

In interrupt method, if the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised , & the microcontroller is interrupted in whatever it is doing , & jump to ISR.

Place all the initialization codes in memory starting at 30h, because to avoid using the memory space allocated to the interrupt vector table.

Example

- A 10khz square wave with 50% duty cycle

```
       ORG   0            ;Reset entry point
       LJMP MAIN          ;Jump above interrupt
       ORG   000BH        ;Timer 0 interrupt vector
T0ISR:CPL  P1.0           ;Toggle port bit
       RETI             ;Return from ISR to Main program
       ORG 0030H          ;Main Program entry point
MAIN:MOV   TMOD,#02H ;Timer 0, mode 2
       MOV   TH0,#50     ;50 us delay
       SETB TR0           ;Start timer
       MOV   IE,#82H      ;Enable timer 0 interrupt
       SJMP $             ;Do nothing just wait
       END
```

Examples

Write a program that continuously gets 8bit data from P0 sends it to P1 while simultaneously creating a square wave of 200µs period on pin p2.1.Use timer 0 to create the square wave.

```
           Org 0000
          LJMP MAIN
          Org 000BH            ;Timer 0 interrupt vector table
          CPL P2.1
          RETI
          Org 0030h
MAIN:      Mov TMOD , #02h
           Mov P0, #0FFh
           Mov TH0, # -92
           Mov IE ,#82h
           SETB TR0
BACK:      Mov A, P0
          Mov P1, A
          SJMP BACK
          END
```

Write a program to generate a square wave of 50Hz frequency on pinP1.2

```
            Org 0
         LJMP main
         Org 000Bh
         CPL P1.2
         Mov TL0, #00
         Mov TH0, # 0DCh
         RETI

            Org 30h
main:    Mov TMOD , #00000001B
         Mov TL0 ,#00
         Mov TH0 ,# 0DCh
         Mov  IE, # 82h
         SETB TR0
Here:   SJMP Here
         END
```

Programming external hardware interrupts

- The two external hardware interrupts are ---
  INT0(P3.2) and INT1(P3.3)
- Two types of activation

-> level triggered (default mode)

-> edge triggered

In the level triggered mode , INT0 & INT1 pins are normally high & if a low level signal is applied to them , it triggers the interrupt.

TCON register selects edge trigger or level trigger, bit 0 for IT0 & bit 2 for IT1

TCON.0=1 -→edge triggered & 0 for level triggered ,by default it is a level triggered.

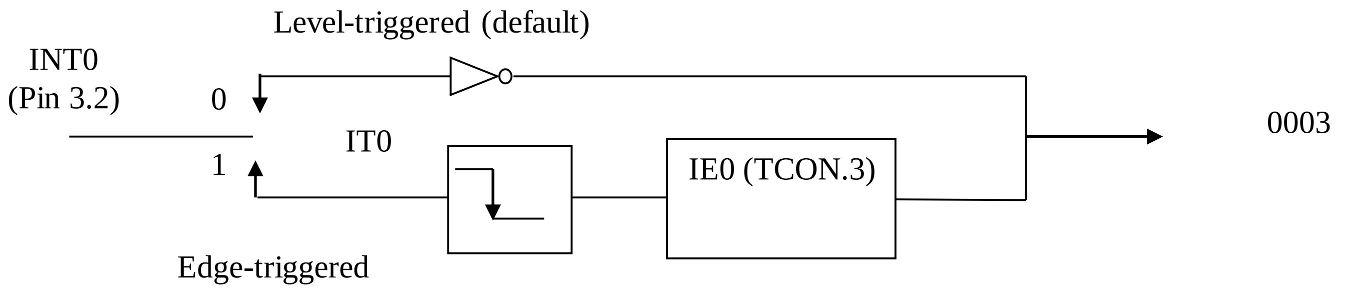
External interrupt type control

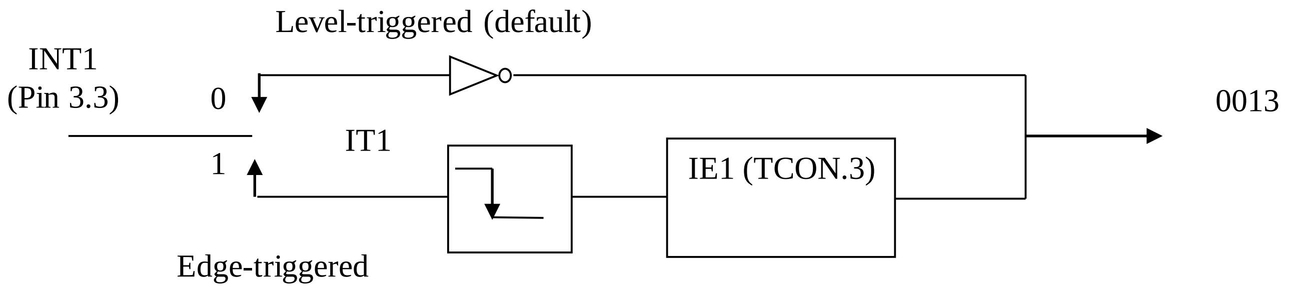
- By low nibble of Timer control register TCON
- IE0 (IE1): External interrupt 0(1) edge flag.
  - set by CPU when external interrupt edge (H-to-L) is detected.
  - Does not affected by H-to-L while ISR is executed (no int on int)
  - Cleared by CPU when RETI executed.
  - does not latch low-level triggered interrupt
- IT0 (IT1): interrupt 0 (1) type control bit.
  - Set/cleared by software
  - IT=1 edge trigger
  - IT=0 low-level trigger

(MSB) (LSB)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1		Timer0		for Interrupt			

External Interrupts





Example of external interrupt

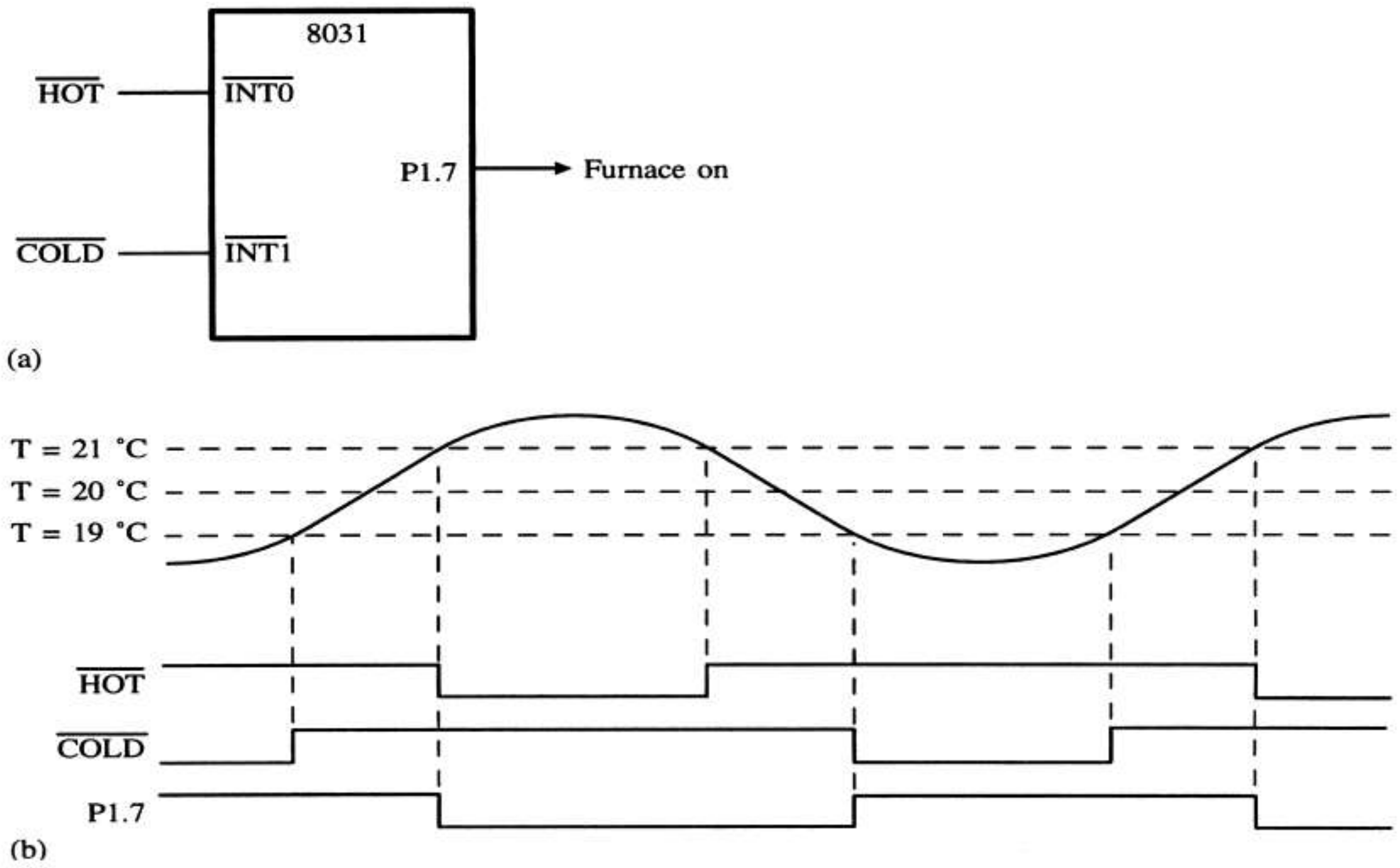


FIGURE 6–5
Furnace example. (a) Hardware connections (b) Timing.

Interrupt Priorities

- What if two interrupt sources interrupt at the same time?
- The interrupt with the highest PRIORITY gets serviced first.
- All interrupts have a power on default priority order.
  1. External interrupt 0 (INT0)
  2. Timer interrupt0 (TF0)
  3. External interrupt 1 (INT1)
  4. Timer interrupt1 (TF1)
  5. Serial communication (RI+TI)
- Priority can also be set to “high” or “low” by IP reg.

Interrupt Priorities (IP) Register

---	---	PT2	PS	PT1	PX1	PT0	PX0

IP.7: reserved

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: external interrupt 1 priority bit

IP.1: timer 0 interrupt priority bit

IP.0: external interrupt 0 priority bit

Interrupt Priorities Example

---	---	PT2	PS	PT1	PX1	PT0	PX0

- `MOV IP , #00000100B` **or** `SETB IP.2` **gives priority order**
  1. Int1
  2. Int0
  3. Timer0
  4. Timer1
  5. Serial
- `MOV IP , #00001100B` **gives priority order**
  1. Int1
  2. Timer1
  3. Int0
  4. Timer0
  5. Serial

Interrupt inside an interrupt

---	**---**	**PT2**	**PS**	**PT1**	**PX1**	**PT0**	**PX0**

- A high-priority interrupt can interrupt a low-priority interrupt
- All interrupt are latched internally
- Low-priority interrupt wait until 8051 has finished servicing the high-priority interrupt

Assume that the INTI pin is connected to a switch that is normally high. Whenever it goes low, it should turn on LED. The LED is connected to P1.3 & 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.

```
        Org 0000h
        LJMP main
        Org 0013h
        SETB P1.3                   ; INTI ISR
        Mov R3, #255                ; load counter
Back:   DJNZ R3, Back
        CLR P1.3
        RETI
        Org 30h
Main:   Mov IE ,# 10000100B
Here:   SJMP Here
        END
```

Upon reset the 8051 makes INTO & INT1 low level triggered interrupts. To make them edge triggered interrupts, we must program the bits of the TCON register.

Assuming that pin 3.3 (INTI) 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 LED.

```
          Org 0000h
         LJMP main
         Org oo13h
         SETB P1.3
         Mov R3 ,#255
Back:    DJNZ R3, Back
         CLR p1.3
         RET

          Org 30h
Main:     SETB TCON .2
          Mov IE, #10000100B
Here:     SJMP , Here
          END
```

Difference between RET and RETI

Both perform the same actions of popping off the top two bytes of the stack Into the program counter and making the 8051 return to where it left off, however RETI also performs clearing the interrupt in service flag, indicating that the servicing of the interrupt is over & the 8051 now can accept a new interrupt on that pin.

If RET is used instead of RETI , it simply block any new interrupt on that pin, It indicates that the interrupt is still being serviced.

Programming the serial communication interrupt

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

```
        org 0
       LJMP MAIN

     org 23h
       LJMP SERIAL          ; jump to serial interrupt ISR
       Org 30h
MAIN:  Mov P1, #0FFh         ; make P1 an input port
       Mov TMOD , #20h     ; timer 1, mode 2(auto-reload)
       Mov TH1 , #0FDh      ; 9600 baud rate
       Mov SCON ,# 50h     ; 8bit, 1 stop , REN enabled
```

```
        Mov IE , #10010000B  ; enabled  serial interrupt
        SETB TR1              ;  start timer 1
BACK: Mov A , P1              ; read data from port1
Mov SBUF ,A                   ; Give a copy to SBUF
        Mov P2, A             ; send it to P2
        SJMP BACK             ; stay in loop indefinitely
        Org  100h
SERIAL: JB  TI, TRANS         ; jump if T1 is high
        Mov A , SBUF  ; otherwise  due to receive
        CLR   RI      ; clear RI since CPU does not
        RETI          ; return from ISR
TRANS: CLR TI         ; clear TI since CPU does not
        RETI          ; return from ISR
        END
```

2. Write a program in which the 8051 gates data from P1 and sends it to P2 continuously while incoming data from the serial port is send to P0 .Assume that XTAL =11.0592MHz. Set the baud rate at 9600.

```
        Org 0
        LJMP MAIN
        Org 23h
        LJMP SERIAL             ; jump to serial ISR
        Org 30h
MAIN :  Mov P1 , #0FFh          ; make P1 an input port
        Mov TMOD , #20h         ; timer 1, mode 2(auto-reload)
        Mov TH1. #0FDh          ; 9600 baud rate
        Mov SCON , #50h         ; 8-bit , 1 stop, REN enabled
        Mov IE, #10010000B      ; enable serial interrupt
        SETB TR1                ; start timer 1
```

```
     BACK:  Mov A, P1       ; read data from port 1
            Mov P2, A       ; send it to P2
            SJMP BACK   ; stay in loop indefinitely
            Org 100h
SERIAL:     JB T1, TRANS ;  jump if T1 is high
           Mov A, SBUF    ; otherwise due to receive
           Mov P0 , A         ; send incoming data to P0
           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
```

3. Write a program using interrupts to do the following :

(a) Receive data serially and send it to p0,

(b) Have port P1 read and transmitted serially, and a copy given to P2,

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

Assume that XTAL =11.0592MHz .Set the baud rate at 4800

```
          Org 0
           LJMP MAIN
           Org 000Bh                    ; ISR for timer 0
           CPL P0.1                      ; toggle P0.1
           RETI                          ; return from ISR
           Org 23h
           LJMP SERIAL                   ; jump to serial int.ISR

                  Org 30h
MAIN :     Mov P1, #0FFh                 ; make P1 an input port
           Mov TMOD , #22h               ; timer 0 & 1, mode 2, auto_reload
           Mov TH1, #0F6h                ; 4800 baud rate
           Mov SCON , #50h               ; 8-bit , 1 stop , REN enabled
           Mov TH0  , # -92h             ; for 5 KHz wave
           Mov IE , # 10010010 B         ; enable serial , timer 0 int
```

```
            SETB TR1                          ; start timer 1
            TR0                               ; START TIMER 0
BACK:        Mov A , P1                       ; read data from port 1
            Mov SBUF, A                       ; give a copy to SBUF
            Mov  P2 , A                       ; write it to P2
             SJMP BACK
              org 100h
SERIAL :      JB TI , TRANS
             Mov A, SBUF
             Mov P0, A                        ; send serial data to P0
             CLR RI                           ; clear RI since CPU does not
             RETI                             ;  return from ISR
            TRANS:   CLR TI                   ; clear TI since CPU does not
            RETI                              ;  return from ISR
            END
```

Serial example(2)

An example for serial port interrupt

```
        ORG 0000H
        LJMP MAIN
;jump to serial ISR
        ORG 23H
        LJMP ISR
;main program
        ORG 30H
;1-initializtion
MAIN:   MOV P0,#0FFH
        MOV TMOD,#20H
        MOV TH1,#-13
        MOV SCON,#50H
        MOV IE,#90H
;2-begin
        SETB TR1
AGAIN:  MOV A,P0
        MOV P1,A
        SJMP AGAIN
```

```
;ISR for reading from serial port
ISR:     PUSH ACC
         JB TI,TRANSM
         MOV A,SBUF
         MOV P2,A
         CLR RI
         SJMP ISREND
TRANSM:  CLR TI
ISREND:  POP ACC
         RETI
         END
```

Serial example(3)

```
an example for serial port interrupt
;for transmitting
        ORG 0000H
        LJMP MAIN
;jump to serial ISR
        ORG 23H
        LJMP ISR
;main program
        ORG 30H
;initializtion
MAIN:   MOV P0,#0FFH
        MOV TMOD,#20H
        MOV TH1,#-13
        MOV SCON,#50H
        MOV IE,#90H
;2-begin
        SETB TR1
AGAIN: SJMP AGAIN
```

```
;ISR for receive from serial to p0
;transmitting to serial from p1
ISR:    JB TI,TRANSM
        MOV A,SBUF
        mov P0,A
        CLR RI
        RETI
TRANSM: MOV A,P1
        MOV SBUF,A
        CLR TI
        RETI
        END
```

Serial example(4)

```
        ORG     0000
;Initialize serial port & timer
INIT:   MOV     SCON,#52H       ;Serial port mode 1
        MOV     TMOD,#20H       ;Timer 1, mode 2
        MOV     TH1,#-13        ;Reload count for 2400 baud
        SETB    TR1             ;Start timer 1

;move character 'B' to accumulator for transmitting
        MOV     A,#'B'

;Transmit characters by serial port
OUTCHR:         MOV     C,P     ;Put parity bit in C flag
        CPL     C               ;Change to odd parity
        MOV     ACC.7,C         ;Add to character code
        END
```

```
AGAIN:JNB TI,AGAIN ;Buffer empty? no:check again
      CLR TI          ;Yes:clear falg and
      MOV SBUF,A      ;send character
      CLR ACC.7     ;Strip off parity bit
      JMP $
```

MICROCONTROLLER AND APPLICATIONS (18EC 42)

Prepared by
Dr. Naveen B

CHAPTER 1

Microcontroller

- Introduction (Historical Background)
- Microprocessors & Microcontrollers
- A Microcontrollers Survey
- Development Systems for Microcontrollers
- RISC & CISC CPU Architectures
- Harvard & Von-Neumann CPU architecture

Historical Background

- Blaise Pascal invented a calculator in 1642 that was constructed of gears and wheels. Each gear contained 10 teeth.
- Charles Babbage began to create what he called his **Analytical Engine**. This machine was to generate navigation tables for the Royal Navy.
- The engine stored 1000 20-digit decimal numbers and variable program.

The development of transistor in 1948 at Bell Labs. In 1958 invent the integrated circuit by Jack Kilby of Texas Instruments. The IC led to the development of digital integrated circuits (RTL, or resistor-to-transistor logic) in the 1960s and the first microprocessor at Intel Corporation in 1971.
Marcian E. Hoff, developed the 4004 microprocessor.

- Intel introduced microprocessors in 1971
  - ∗ 4-bit microprocessors
    - ➢ 4004
    - ➢ 4040

    It addressed a mere 4096 4-bit wide memory locations.

    The problems with 4004 are :

    Speed, width, Memory size
  - ∗ 8-bit microprocessors
    - » 8008
    - » 8080
    - » 8085
- In 1973, Intel corporation released the 8008, an extended 8-bit version of the 4040 microprocessor.
- The memory size are 16K bytes
- The instructions are 48

- Intel introduced the 8080 microprocessor in 1973. The first modern 8-bit microprocessors.
- Motorola Corporation introduced its MC6800 microprocessor .

8080 address memory with 64K bytes than the 8008 with 16K bytes.

In 1977, Intel corporation introduced an update version of the 8080—the 8085. the last 8-bit microprocessor developed by Intel. The main advantages of the 8085 were its internal clock generator, internal system controller, and higher clock frequency.

- 16-bit processors
  - 8086 introduced in 1978
    - 20-bit address bus, 16-bit data bus
  - 8088 is a less expensive version
    - Uses 8-bit data bus
  - Can address up to 4 segments of 64 KB
  - Referred to as the real mode

8086 and 8088 addressed 1M bytes of memory.

A small 4- or 6-byte instruction cache or queue that prefetched a few instructions before they were executed.

- 80186
  - A faster version of 8086
  - 16-bit data bus and 20-bit address bus
  - Improved instruction set
- 80286 was introduced in 1982
  - 24-bit address bus
  - 16 MB address space
  - Enhanced with memory protection capabilities
  - Introduced protected mode
    - Segmentation in protected mode is different from the real mode

- 80386 was introduced in 1985
  - First 32-bit processor
  - 32-bit data bus and 32-bit address bus
  - 4 GB address space
  - Segmentation can be turned off (flat model)
  - Introduced paging
- 80486 was introduced in 1989
  - Improved version of 386
  - Combined coprocessor functions for performing floating-point arithmetic
  - Added parallel execution capability to instruction decode and execution units
    - Achieves scalar execution of 1 instruction/clock
  - Later versions introduced energy savings for laptops

- Pentium (80586) was introduced in 1993
  - Similar to 486 but with 64-bit data bus
  - Wider internal data paths
    - 128- and 256-bit wide
  - Added second execution pipeline
    - Superscalar performance
    - Two instructions/clock
  - Doubled on-chip L1 cache
    - 8 KB data
    - 8 KB instruction
  - Added branch prediction

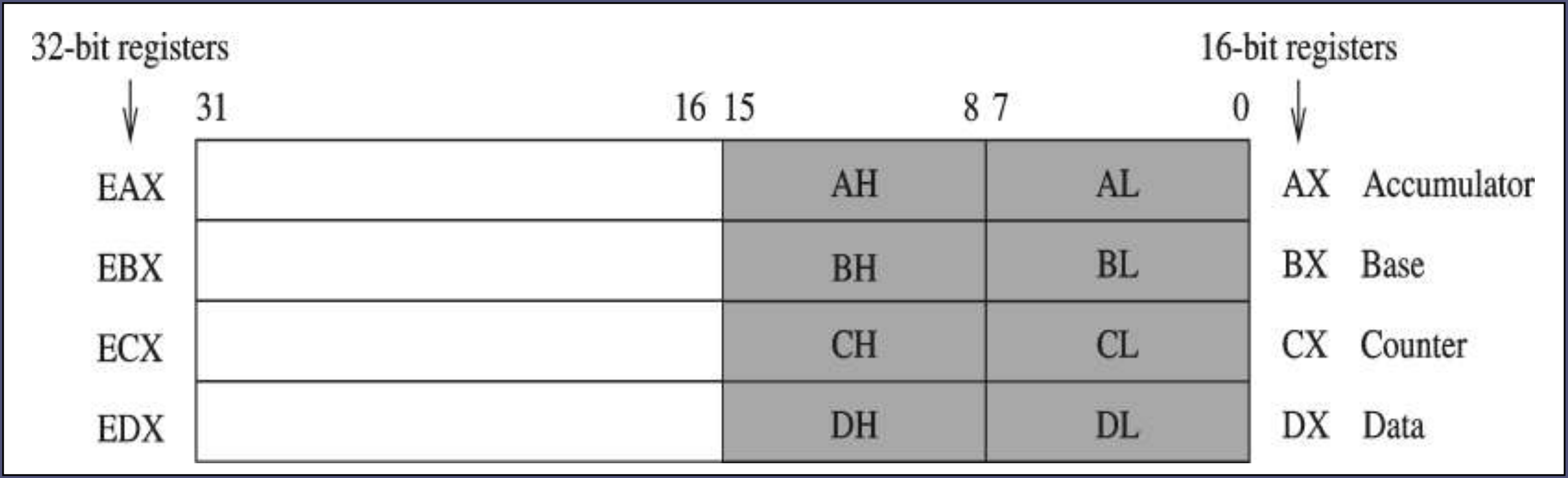
- Pentium Pro was introduced in 1995
  - Three-way superscalar
    - 3 instructions/clock
  - 36-bit address bus
    - 64 GB address space
  - Introduced dynamic execution
    - Out-of-order execution
    - Speculative execution
  - In addition to the L1 cache
    - Has 256 KB L2 cache

- Pentium II was introduced in 1997
  - Introduced multimedia (MMX) instructions
  - Doubled on-chip L1 cache
    - 16 KB data
    - 16 KB instruction
  - Introduced comprehensive power management features
    - Sleep
    - Deep sleep
  - In addition to the L1 cache
    - Has 256 KB L2 cache
- Pentium III, Pentium IV,…

- Itanium processor
  - RISC design
    - Previous designs were CISC
  - 64-bit processor
  - Uses 64-bit address bus
  - 128-bit data bus
  - Introduced several advanced features
    - Speculative execution
    - Predication to eliminate branches
    - Branch prediction

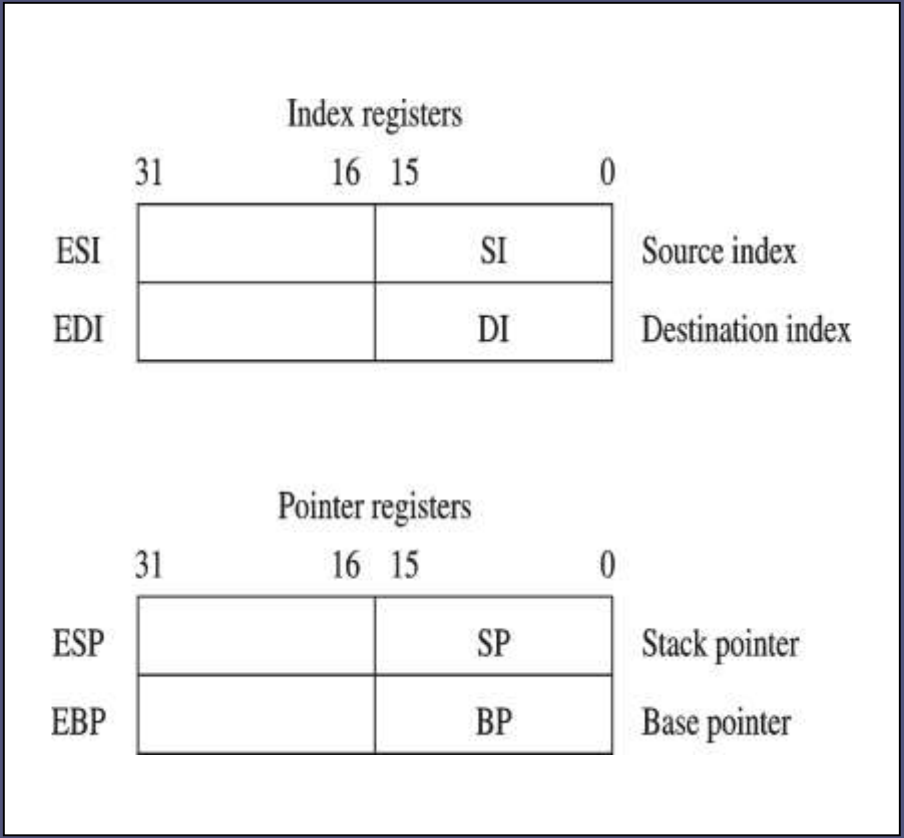
Pentium Registers

- Four 32-bit registers can be used as
  - Four 32-bit register (EAX, EBX, ECX, EDX)
  - Four 16-bit register (AX, BX, CX, DX)
  - Eight 8-bit register (AH, AL, BH, BL, CH, CL, DH, DL)
- Some registers have special use
  - ECX for count in loop instructions

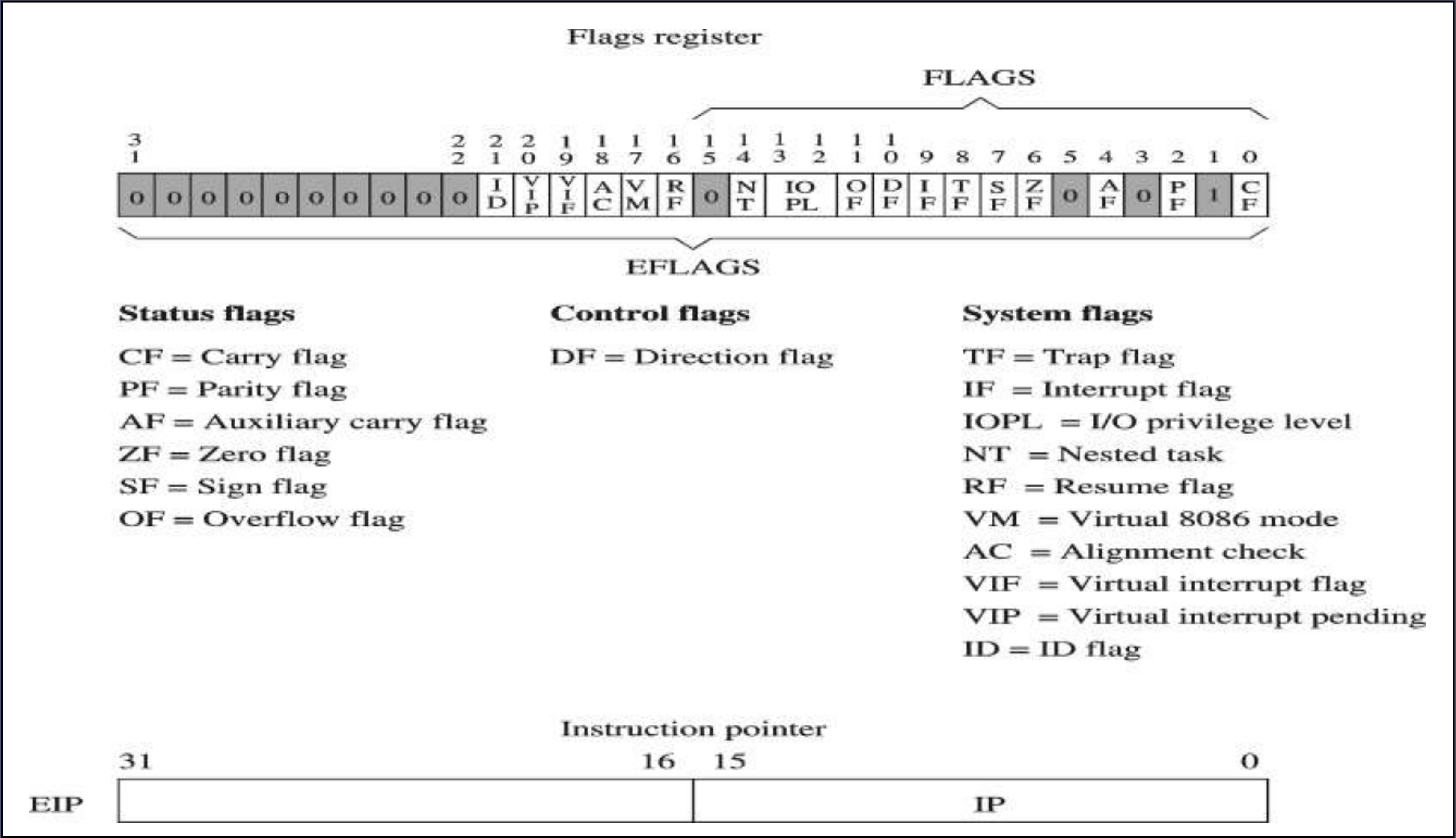


Pentium Registers (cont'd)

- Two index registers
  - 16- or 32-bit registers
  - Used in string instructions
    - Source (SI) and destination (DI)
  - Can be used as general-purpose data registers
- Two pointer registers
  - 16- or 32-bit registers
  - Used exclusively to maintain the stack



Pentium Registers (cont'd)



Status flags	Control flags	System flags
CF = Carry flag	DF = Direction flag	TF = Trap flag
PF = Parity flag		IF = Interrupt flag
AF = Auxiliary carry flag		IOPL = I/O privilege level
ZF = Zero flag		NT = Nested task
SF = Sign flag		RF = Resume flag
OF = Overflow flag		VM = Virtual 8086 mode
		AC = Alignment check
		VIF = Virtual interrupt flag
		VIP = Virtual interrupt pending
		ID = ID flag

Protected Mode Architecture

- Pentium supports two modes
  - Protected mode
    - 32-bit mode
    - Supports segmentation and paging
  - Real mode
    - Uses 16-bit addresses
    - Runs 8086 programs
    - Pentium acts as a faster 8086

Intel and Motorola microprocessors

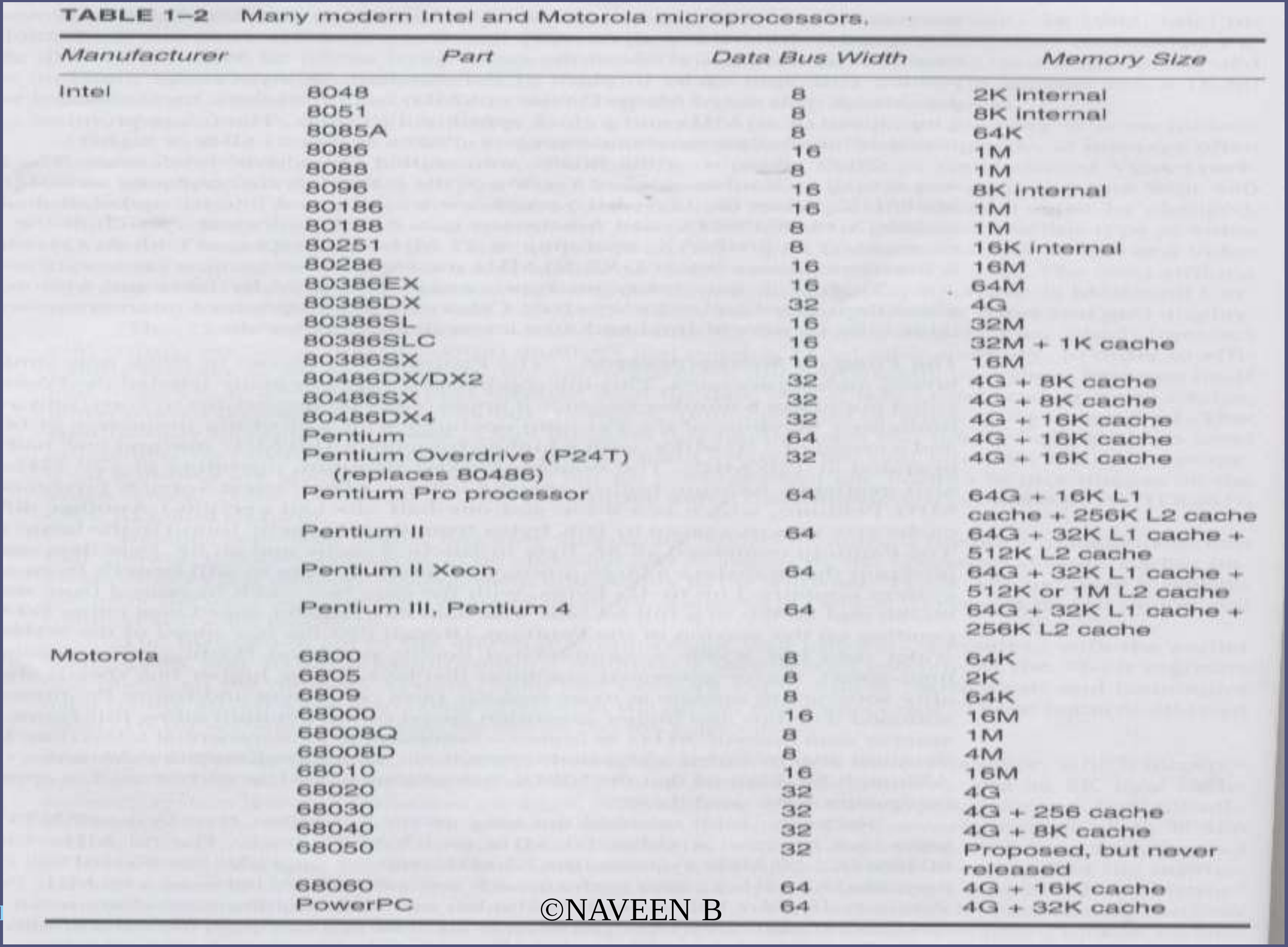


TABLE 1–2 Many modern Intel and Motorola microprocessors.

Manufacturer	*Part*	*Data Bus Width*	*Memory Size*
Intel	8048	8	2K internal
	8051	8	8K internal
	8085A	8	64K
	8086	16	1M
	8088	8	1M
	8096	16	8K internal
	80186	16	1M
	80188	8	1M
	80251	8	16K internal
	80286	16	16M
	80386EX	16	64M
	80386DX	32	4G
	80386SL	16	32M
	80386SLC	16	32M + 1K cache
	80386SX	16	16M
	80486DX/DX2	32	4G + 8K cache
	80486SX	32	4G + 8K cache
	80486DX4	32	4G + 16K cache
	Pentium	64	4G + 16K cache
	Pentium Overdrive (P24T) (replaces 80486)	32	4G + 16K cache
	Pentium Pro processor	64	64G + 16K L1 cache + 256K L2 cache
	Pentium II	64	64G + 32K L1 cache + 512K L2 cache
	Pentium II Xeon	64	64G + 32K L1 cache + 512K or 1M L2 cache
	Pentium III, Pentium 4	64	64G + 32K L1 cache + 256K L2 cache
Motorola	6800	8	64K
	6805	8	2K
	6809	8	64K
	68000	16	16M
	68008Q	8	1M
	68008D	8	4M
	68010	16	16M
	68020	32	4G
	68030	32	4G + 256 cache
	68040	32	4G + 8K cache
	68050	32	Proposed, but never released
	68060	64	4G + 16K cache
	PowerPC	64	4G + 32K cache

Applications

Home

- Appliances
- Intercom
- Telephones
- Security systems
- Garage door openers
- Answering Machine
- Fax machine
- Exercise Equipments
- Washing Machine
- Home Computers
- TVs
- Cable TV Tuner
- VCR
- Camcorder
- Remote Controls

Automobile

- Engine Control
- Air Bag
- Transmission Control
- Automobile Instrumentation
- Entertainment
- Keyless Entry

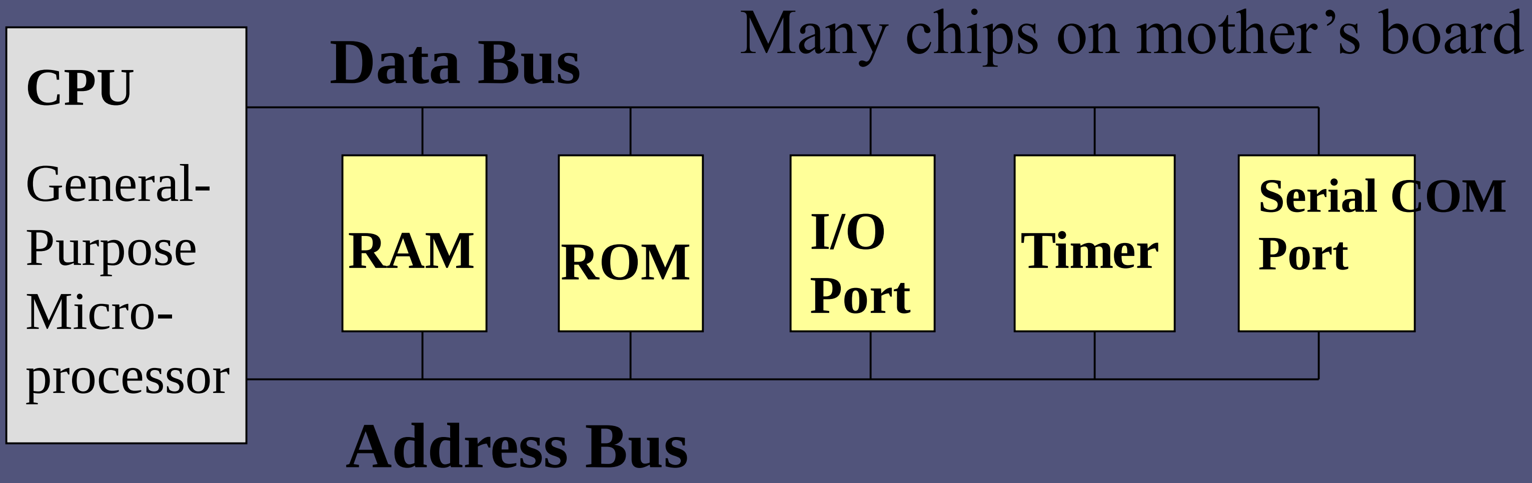
Office

- Laser Printer
- Copier
- Color Printer
- Paging

Introduction

General-purpose microprocessor

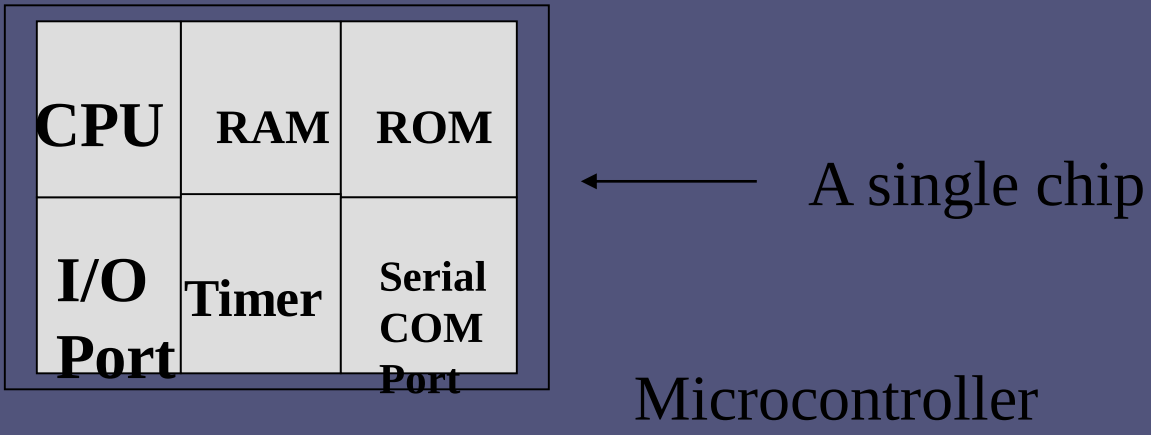
- CPU for Computers
- No RAM, ROM, I/O on CPU chip itself
- Example：Intel’s x86, Motorola’s 680x0



General-Purpose Microprocessor System

Microcontroller :

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example：Motorola's 6811, Intel's 8051, Zilog's Z8 and PIC 16X



Microprocessor vs. Microcontroller

Microprocessor

- CPU is stand-alone, RAM, ROM, I/O, timer are separate
- designer can decide on the amount of ROM, RAM and I/O ports.
- Requires more hardware
- expansive
- general-purpose
- Less multifunctional pins
- 1 or 2 bit handling Inst.
- Single memory map for Data & Code
- Many instructions to move data between memory & CPU

Microcontroller

- CPU, RAM, ROM, I/O and timer are all on a single chip
- fix amount of on-chip ROM, RAM, I/O ports
- Requires less hardware
- for applications in which cost, power and space are critical
- single-purpose
- More multifunctional pins
- More bit handling inst.
- separate memory map for Data & Code
- 1 or 2 instructions

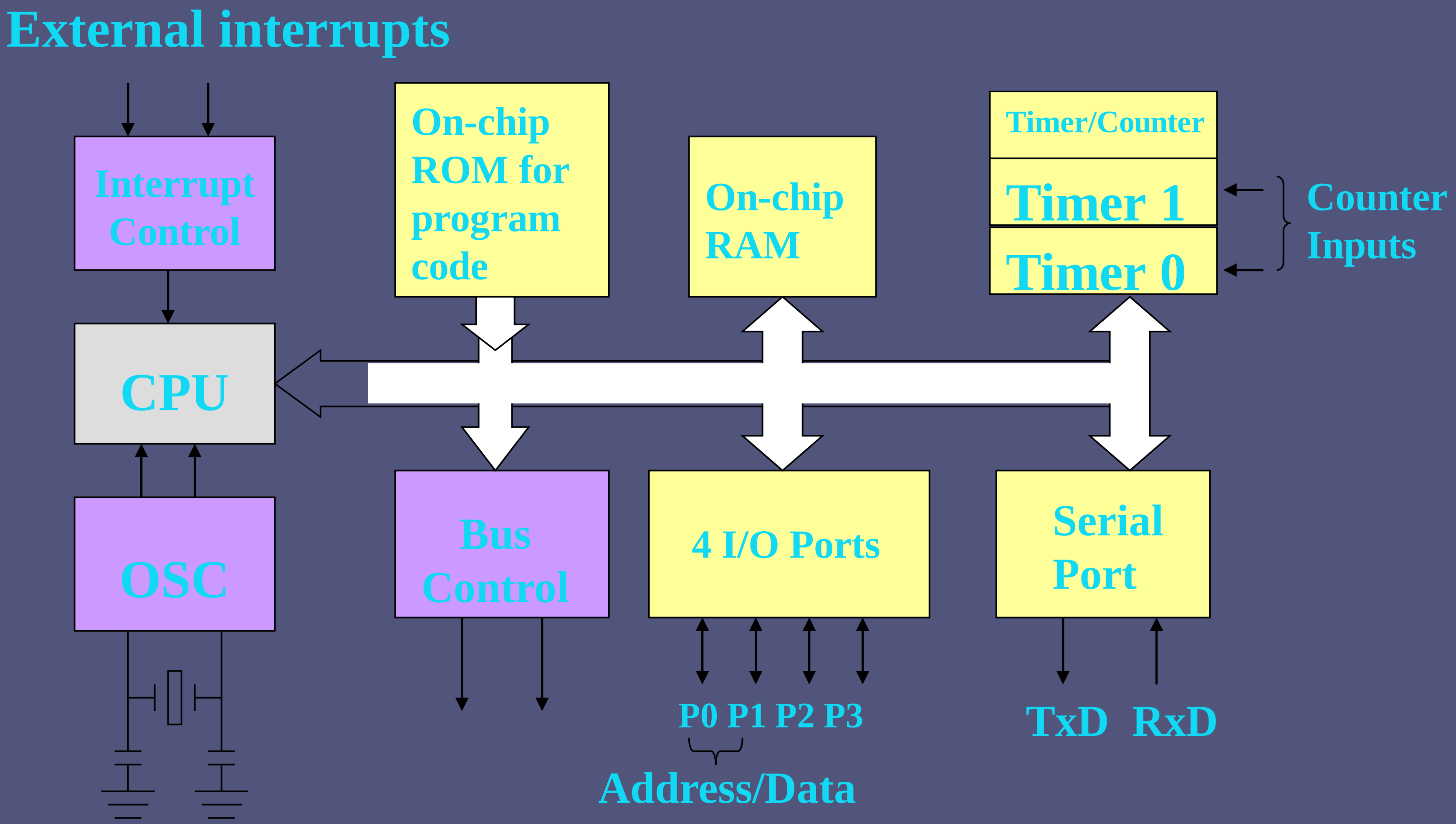
Embedded System

- Embedded system means the processor is embedded into that application.
- An embedded product uses a microprocessor or microcontroller to do one task only.
- In an embedded system, there is only one application software that is typically burned into ROM.
- Example：printer, keyboard, video game player

Three criteria in Choosing a Microcontroller

1. meeting the computing needs of the task efficiently and cost effectively
   - speed, the amount of ROM and RAM, the number of I/O ports and timers, size, packaging, power consumption
   - easy to upgrade
   - cost per unit
2. availability of software development tools
   - assemblers, debuggers, C compilers, emulator, simulator, technical support
3. wide availability and reliable sources of the microcontrollers.

Block Diagram



Comparison of the 8051 Family Members

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

A MICROCONTROLLER SURVAY

Data Bits	**MicroController IC**	**RAM / ROM**
4 bit MicroController	MCS40 (4004)	32 byte RAM, 512 byteROM
8 bit MicroController	8051 (MCS 51)	128 byte RAM, 4 K ROM
16 bit MicroController	80C196 (MCS96 Family)	256 byte RAM, 8 K ROM
32 bit MicroController	80960 (Floating point Unit, 512 Byte Instruction Cache)	

DEVELOPMENT SYSTEMS FOR MICROCONTROLLERS

The package of hardware & software will allow the MC to be programmed & connected to the application is called development system.

- Trained personnel must be available
- A device capable of programming EPROMs must be available
- Software is needed along with PC to host it

RISC & CISC CPU ARCHITECTURES

- A CISC processor has most of the following properties:
  - Richer instruction set, some simple, some very complex
  - Instructions generally take more than 1 clock to execute
  - Instructions of a variable size
  - Instructions interface with memory in multiple mechanisms with complex addressing modes
  - Microcode control
  - No pipelining
  - Work well with simpler compiler
- Segmented memory model
  - Few registers
- Crappy floating point performance
- Upward compatibility within a family

- A RISC processor has most of the following properties:
    - Simple primitive instructions and addressing modes
    - Instructions execute in one clock cycle
    - Uniformed length instructions and fixed instruction format
    - Instructions interface with memory via fixed mechanisms (load/store)
    - Hardwired control
    - Pipelining
  - Instruction set is orthogonal (little overlapping of instruction functionality)
  - Complexity pushed to the compiler
  - Superscalar and out-of-order execution
  - Large number of registers
  - Fast floating point performance

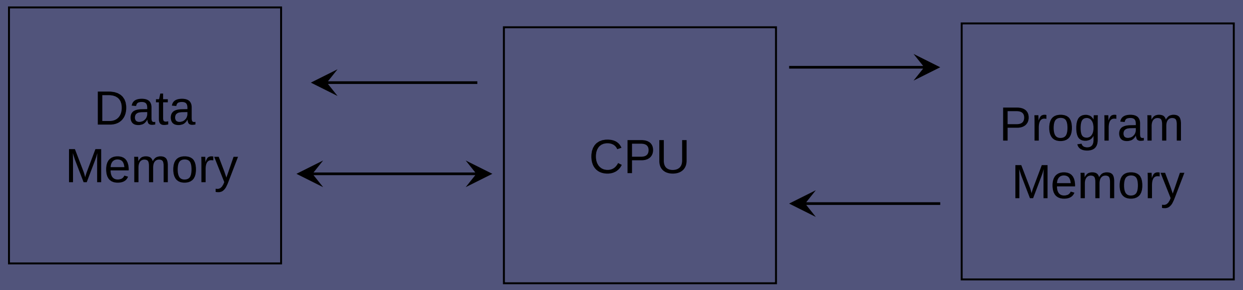
Comparison between RISC & CISC

Simple instructions taking one cycle	Complex instructions taking many cycle
Few instructions only	Many instructions
Simple addressing modes	Complex addressing modes
Few addressing modes & most instructions have register addressing mode	Many addressing modes
Very few instructions refer memory	Most of the instructions may refer memory
Instructions are executed by hardware	Executed by micro program
Fixed instruction format	Variable instruction format
Highly pipelined	Not pipelined
Complexity is in the compiler	Complexity is in the micro program
More registers	Few registers

Harvard Vs Von-Neumann

There are two ways in which the computer memory used for storing instructions may be organized.

Harvard

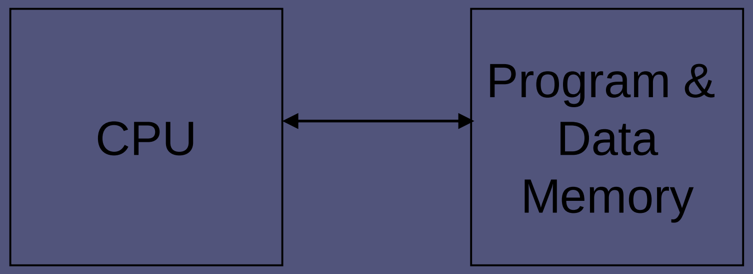


1. Separate blocks for Program & Data Memory

2. Data bus & Program bus usually of different size

8051 has Harvard Architecture which uses the same address in different memories for code and data

Von Neumann



Program & Data Memory in the same block

- In a von Neumann architecture, the instruction has first to be fetched, using the program counter; then it can be executed. Since when the instruction is executed, it may also read or write data, you often cannot load one instruction and execute another at the same time. So the basic sequence in a von-Neuman architecture system is
- fetch
  execute
  fetch
  execute
- This means that such a system may be *slower*. On the other hand, it has great advantages of *flexibility*, and the efficient use of memory caches can do a great deal to mitigate the apparent slowness of the architecture.
- For example, what happens when you load up a game from disk into your computer? The program is stored as data, for example on a disc. You have to read that data, and store it in the memory of your computer. When you have done that, you can treat it as a program, (i.e. as a set of instructions) and run (execute) it. With a von Neumann architecture, since you have only one memory system, there is no problem; but with a Harvard architecture, you cannot do it - you cannot write data to the program memory - a special system is required to load the program into program memory.

- The Harvard architecture also prevents you reading data from the program memory. For example, you may wish to include in your program tables of data, which can be used by the program; for example, messages to be displayed on the screen, or some kind of "look-up table". In a von Neumann architecture there is no problem; you can just store the table along with your program, and read it when you want to, because an instruction can read data from any address; in a Harvard architecture, data stored in the program memory cannot be read as data in the data memory. There are ways to get round this problem.
- A von Neumann architecture is used for most computers; it allows the storing and running of different programs. A Harvard architecture is more appropriate for a microcontroller; in use, it will only ever run one program which will be stored in the ROM in its program memory. Moreover, the extra speed without the complexity of a sophisticated cache controller will be useful in some circumstances.

8051 Serial Communication

By Dr. Naveen B

Advantages of serial communication over parallel communication

- Single Data line is used
- Cheaper
- Communicate over a longer distance

Modem is used to convert data from 0’s & 1’s to audio signals and vice-versa

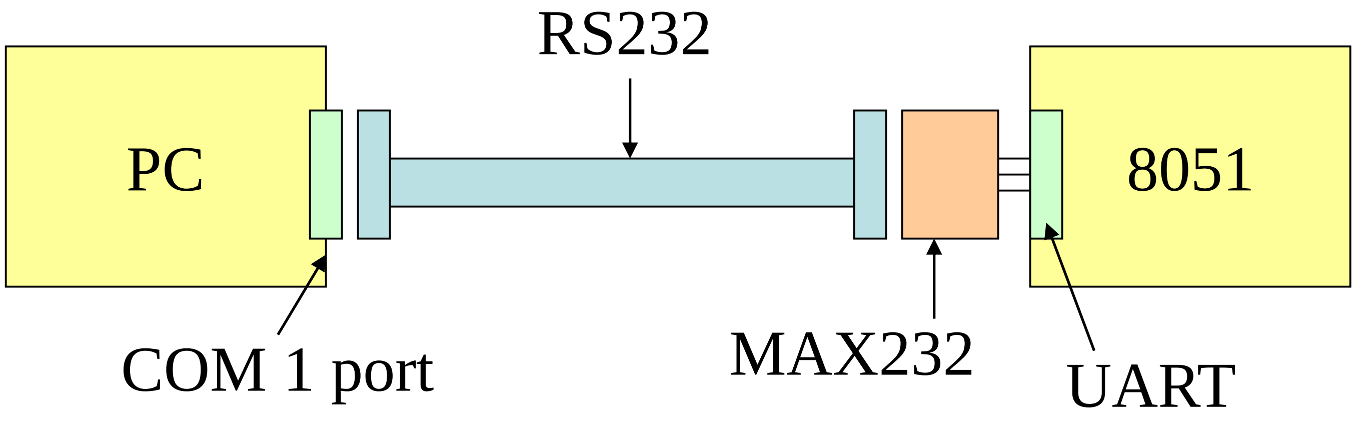
Parallel in serial out & serial in parallel out registers are used.

8051 and PC

- The 8051 module connects to PC by using RS232.

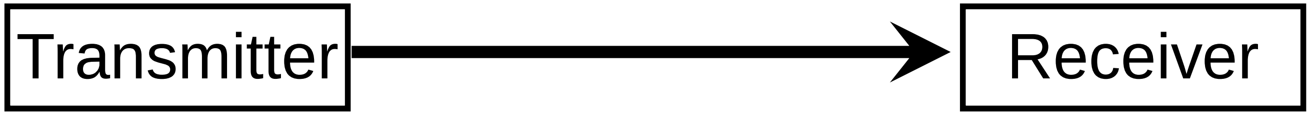
- RS232 is a protocol which supports half-duplex, synchronous/asynchronous, serial communication.

We discuss these terms in following sections.

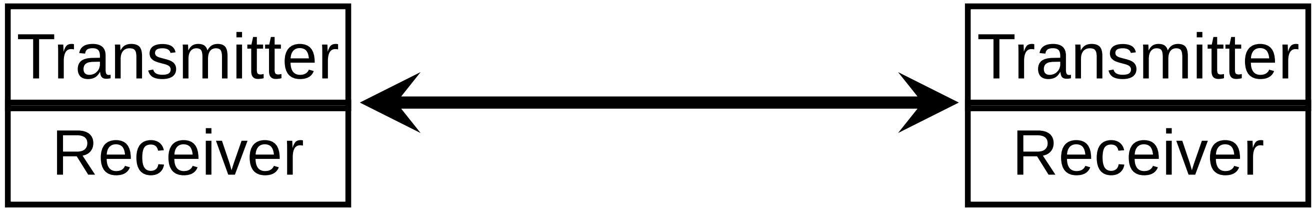


Simplex vs. Duplex Transmission

- Simplex transmission: the data can sent in one direction.
  - Example: the computer only sends data to the printer.

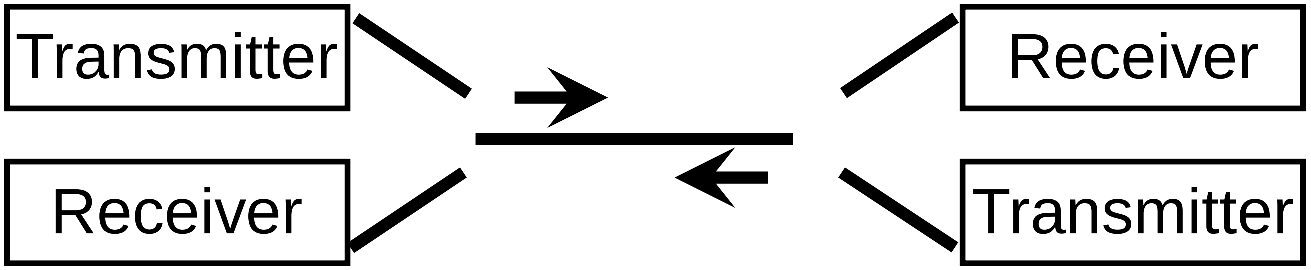


- Duplex transmission: the data can be transmitted and receive

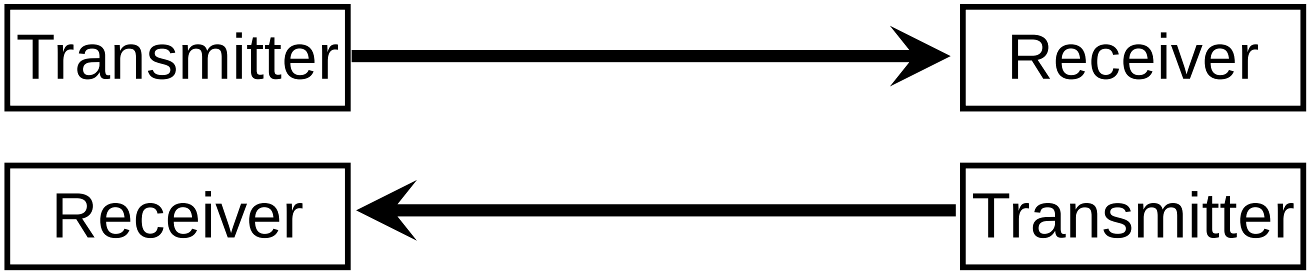


Half vs. Full Duplex

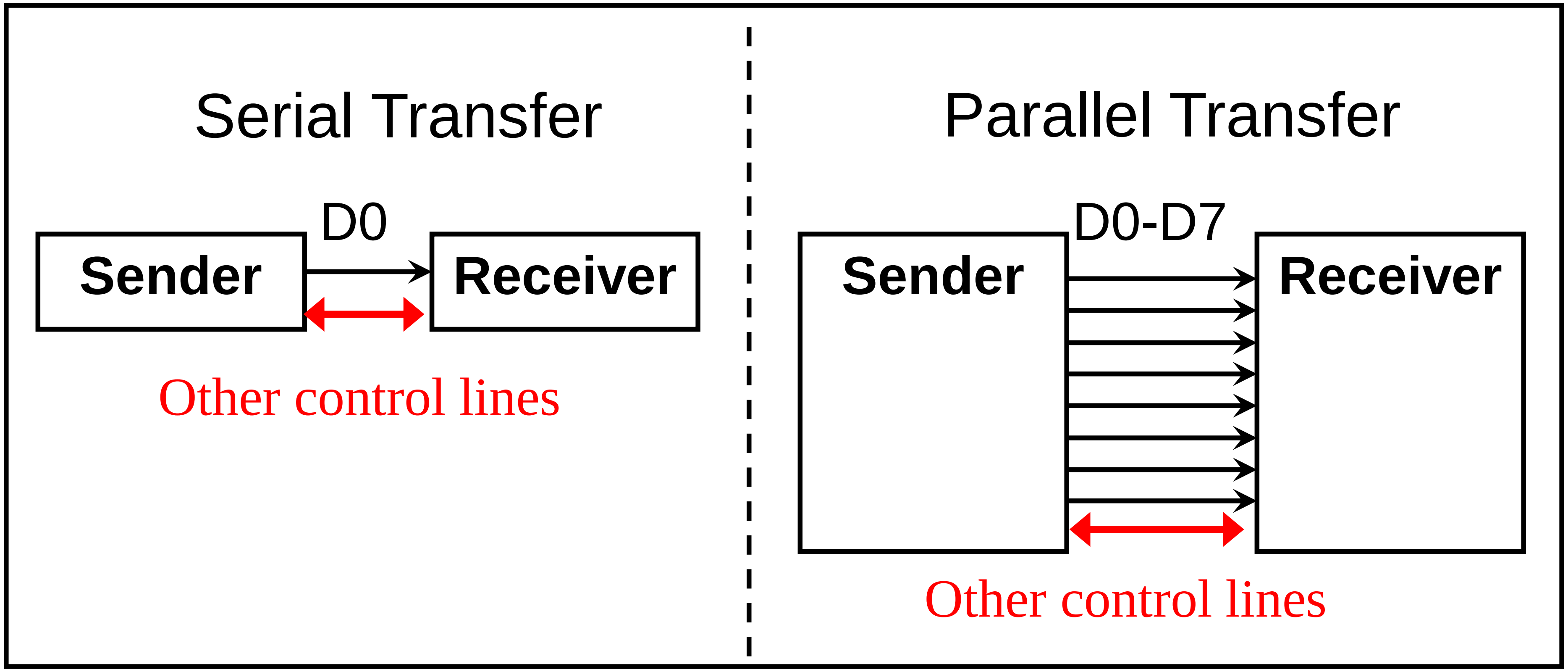
- Half duplex: if the data is transmitted one way at a time.



- Full duplex: if the data can go both ways at the same time.
  - Two wire conductors for the data lines.

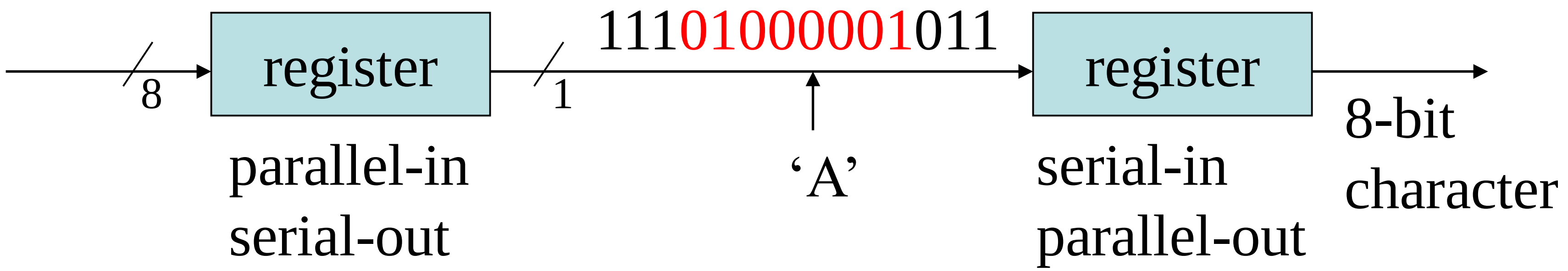


Serial vs Parallel Data Transfer



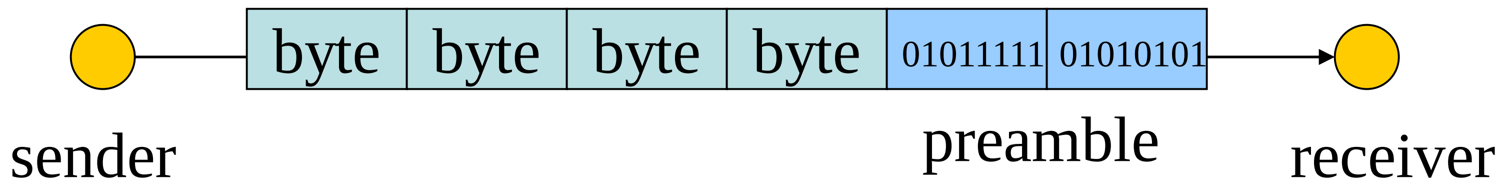
Serial Communication

- How to transfer data?
  - Sender:
    - The byte of data must be converted to serial bits using a parallel-in-serial-out shift register.
    - The bit is transmitted over a single data line.
  - Receiver
    - The receiver must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte.

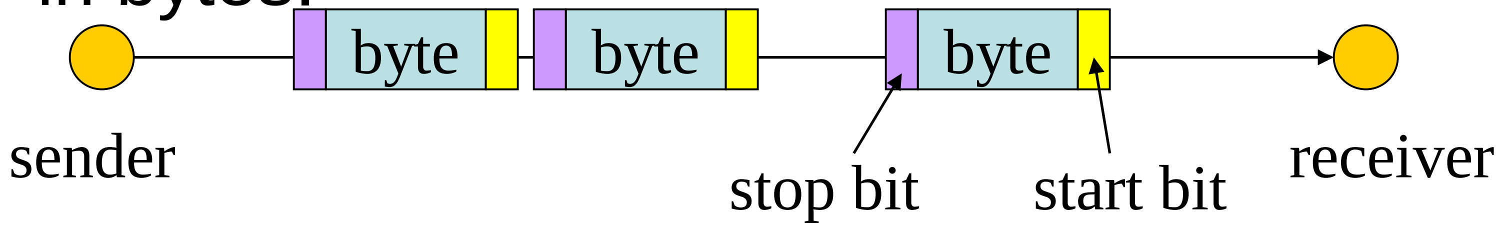


Asynchronous vs. Synchronous

- Serial communication uses two methods:
  - In synchronous communication, data is sent in blocks of bytes.



  - In asynchronous communication, data is sent in bytes.

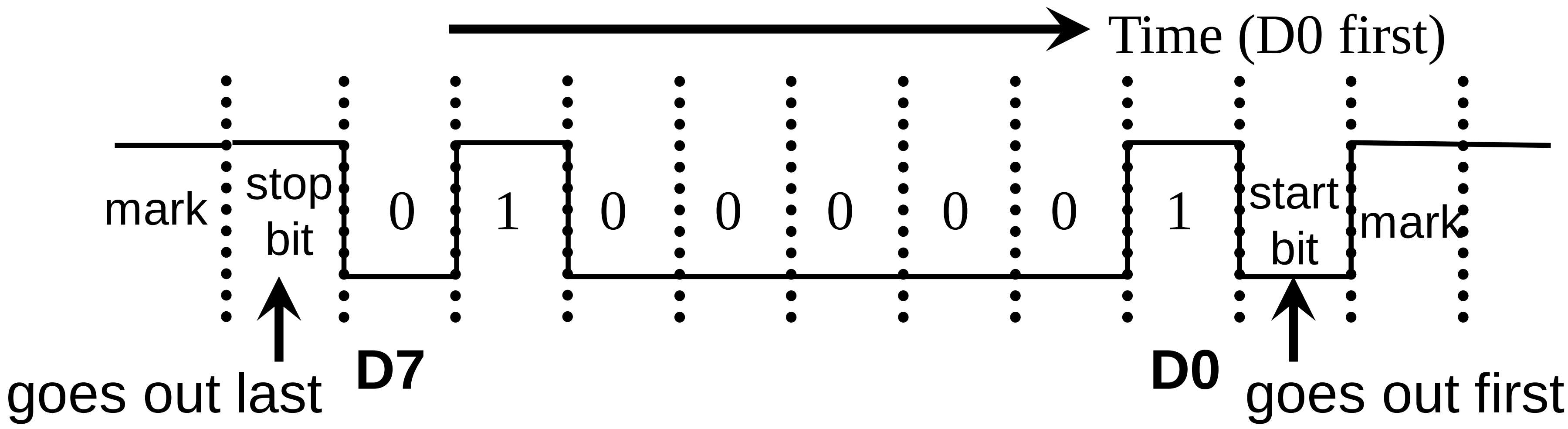


UART & USART

- It is possible to write software to use both methods, but the programs can be tedious and long.
- Special IC chips are made for serial communication:
  - USART (universal synchronous-asynchronous receiver-transmitter)
  - UART (universal asynchronous receiver-transmitter)
- The 8051 chip has a built-in UART.
  - Half-duplex
  - Asynchronous mode only

Framing (1/3)

- How to detect that a character is sent via the line in the asynchronous mode?
  - Answer: Data framing!
- Each character is placed in between start and stop bits. This is called framing.



Framing (2/3)

- The LSB is sent out first.
- The start bit is 0 (low) and always one bit.
- The stop bits is 1 (high).
- The stop bit can be one (if 8 bits used in ASCII) or two bits (if 7 bits used in ASCII).
  - In asynchronous serial communication, peripheral chips and modems can be programmed for data that is 7 or 8 bits.
- When there is no transfer, the signal is 1 (high), which is referred to as mark.

Framing (3/3)

- We have a total of 10 bits for each character:
  - 8-bits for the ASCII code
  - 2-bits for the start and stop bits
- In some systems in order to maintain data integrity, the parity bit is included in the data frame.
  - In an odd-parity bit system the total number of bits, including the parity bit, is odd.
  - UART chips allow programming of the parity bit for odd-, even-, and no-parity options.

Handshaking signals to ensure fast & reliable data transmission between Two devices

- DTR (Data Terminal Ready): when the terminal is turned ON, after going through a self test, it sends out a signal DTR to indicate that it is ready for communication. This is an i/p pin from DTE & an i/p to the modem.

- DSR (Data Set Ready): when DCE (modem) is turned on & has gone through the self test, it asserts DSR to indicate that it is ready to communicate .It is an o/p from modem & i/p to the pc (DTE).

- RTS (Request To Send): When the DTE device has a byte to transmit, it asserts RTS to signal the modem that it has a byte of data to transmit

CTS (clear to send): in response to RTS, when the modem has room for storing the data it is to receive, it sends out signal CTS to the DTE to indicate that it can receive the data now.

DCD (data carrier detect): the modem asserts signal DCD to inform the DTE that a valid carrier has been detected & that contact between it & the other modem is established

RI (ring detector): an o/p from the modem & an i/p to PC indicates that the telephone is ringing. It gives on & OFF in synchronization with the ringing sound

How to communicate 8051 to PC

- Connect TXD to RXD and RXD to TXD from pc to 8051
- Use max232 to transform signal from TTL level to RS232 level
- The baud rate of the 8051 must matched to the baud rate of the pc
- PC standard baud rates
  - 2400-4800-9600-14400-19200-28800-33600-57600
- Serial mode 1 is used
- Timer 1 is used
- The 8051 UART divides the machine cycle frequency by 32
- Machine cycle is 1/12 XTAL frequency
- We use timer1 in mode 2 (auto reload)

MAX 232

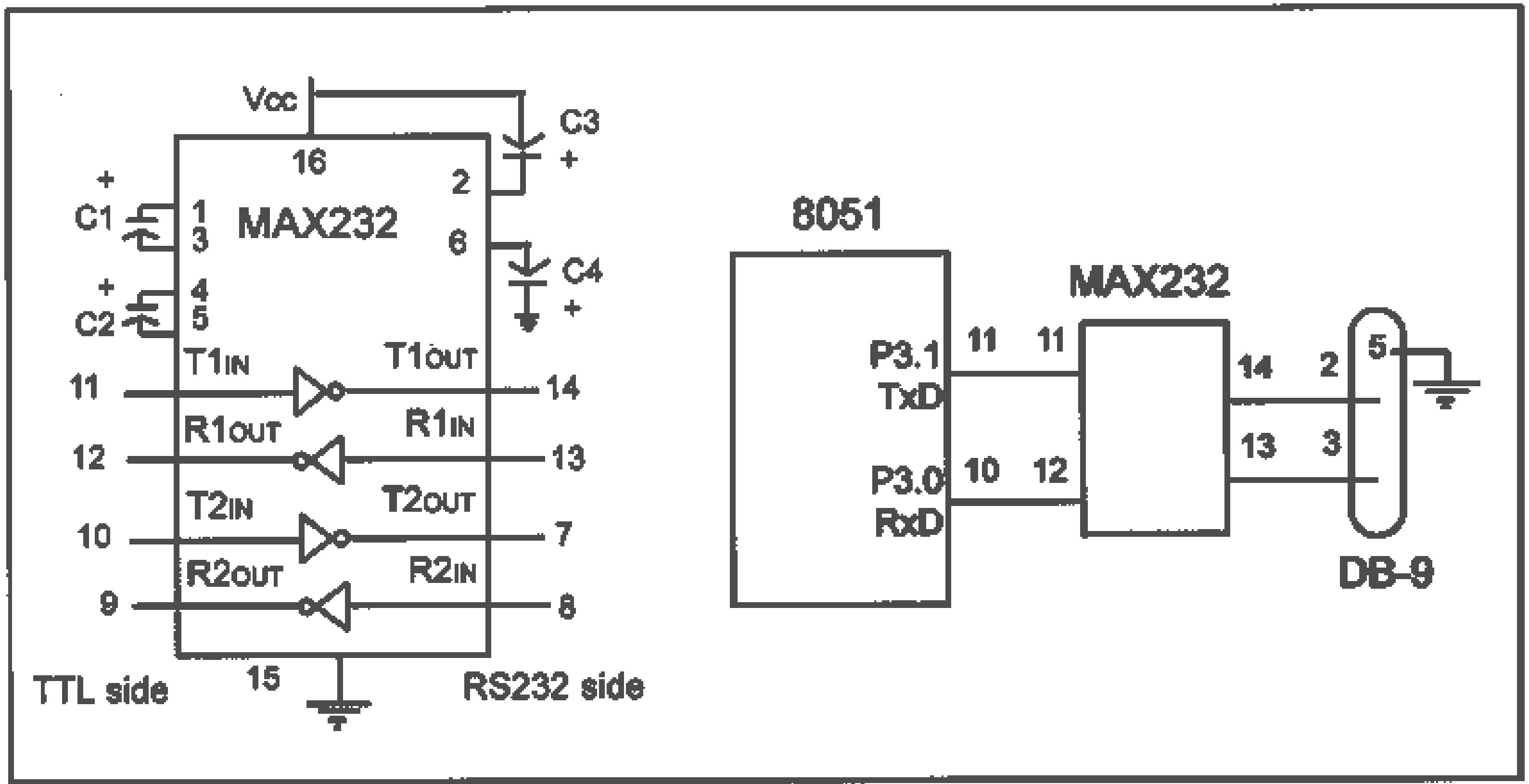


Figure 10.7: (a) Inside MAX232 and (b) its Connection to the 8051 (Null Modem)

TxD and RxD pins in the 8051

- In 8051, the data is received from or transmitted to
  - RxD: received data (Pin 10, P3.0)
  - TxD: transmitted data (Pin 11, P3.1)
- TxD and RxD of the 8051 are TTL compatible.
- The 8051 requires a line driver to make them RS232 compatible.
  - One such line driver is the MAX232 chip.

PC Baud Rates

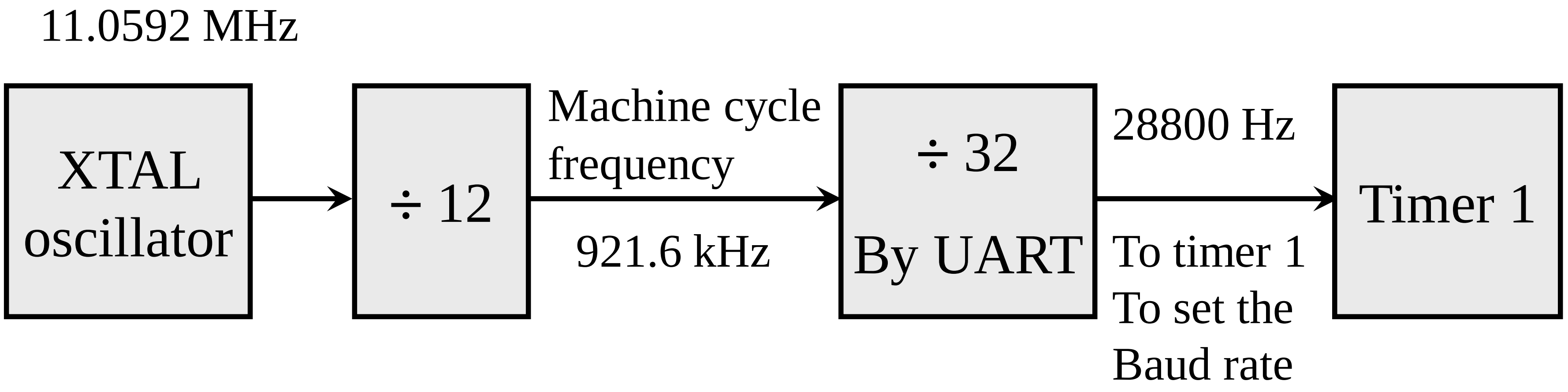
- PC supports several baud rates.
- HyperTerminal supports baud rates much higher than the ones list in the Table.

110 bps
150
300
600
1200
2400
4800
9600 (default)
19200

Note: Baud rates supported by 486/Pentium IBM PC BIOS.

Baud Rates in the 8051 (1/2)

- The 8051 transfers and receives data serially at many different baud rates by using UART.
- UART divides the machine cycle frequency by 32 and sends it to Timer 1 to set the baud rate.
- Signal change for each roll over of timer 1



Baud Rates in the 8051 (2/2)

- Timer 1, mode 2 (8-bit, auto-reload)
- Define TH1 to set the baud rate.
  - XTAL = 11.0592 MHz
  - The system frequency = 11.0592 MHz / 12 = 921.6 kHz
  - Timer 1 has 921.6 kHz/ 32 = 28,800 Hz as source.
  - TH1=FDH means that UART sends a bit every 3 timer source.
  - Baud rate = 28,800/3= 9,600 Hz

Example

With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates. (a) 9600 (b) 2400 (c) 1200

Solution:

With XTAL = 11.0592 MHz, we have:

The frequency of system clock = 11.0592 MHz / 12 = 921.6kHz

The frequency sent to timer 1 = 921.6 kHz/ 32 = 28,800 Hz

(a) 28,800 / 3 = 9600 where -3 =FD (hex) is loaded ontoTH1

(b)28,800 / 12 = 2400 where -12 = F4 (hex) is loaded intoTH1

(c)28,800 / 24 = 1200 where -24 = E8 (hex) is loaded intoTH1

Registers Used in Serial Transfer Circuit

- SBUF (Serial data buffer)
- SCON (Serial control register)
- PCON (Power control register)

SBUF Register

- Serial data register: **SBUF**

```
MOV SBUF,#'A'     ;put char 'A' to transmit
MOV SBUF,A        ;send data from A
MOV A,SUBF        ;receive and copy to A
```

  - An 8-bit register
  - Set the usage mode for two timers
    - For a byte of data to be transferred via the TxD line, it must be placed in the SBUF.
    - SBUF holds the byte of data when it is received by the 8051;s RxD line.

Serial port block diagram

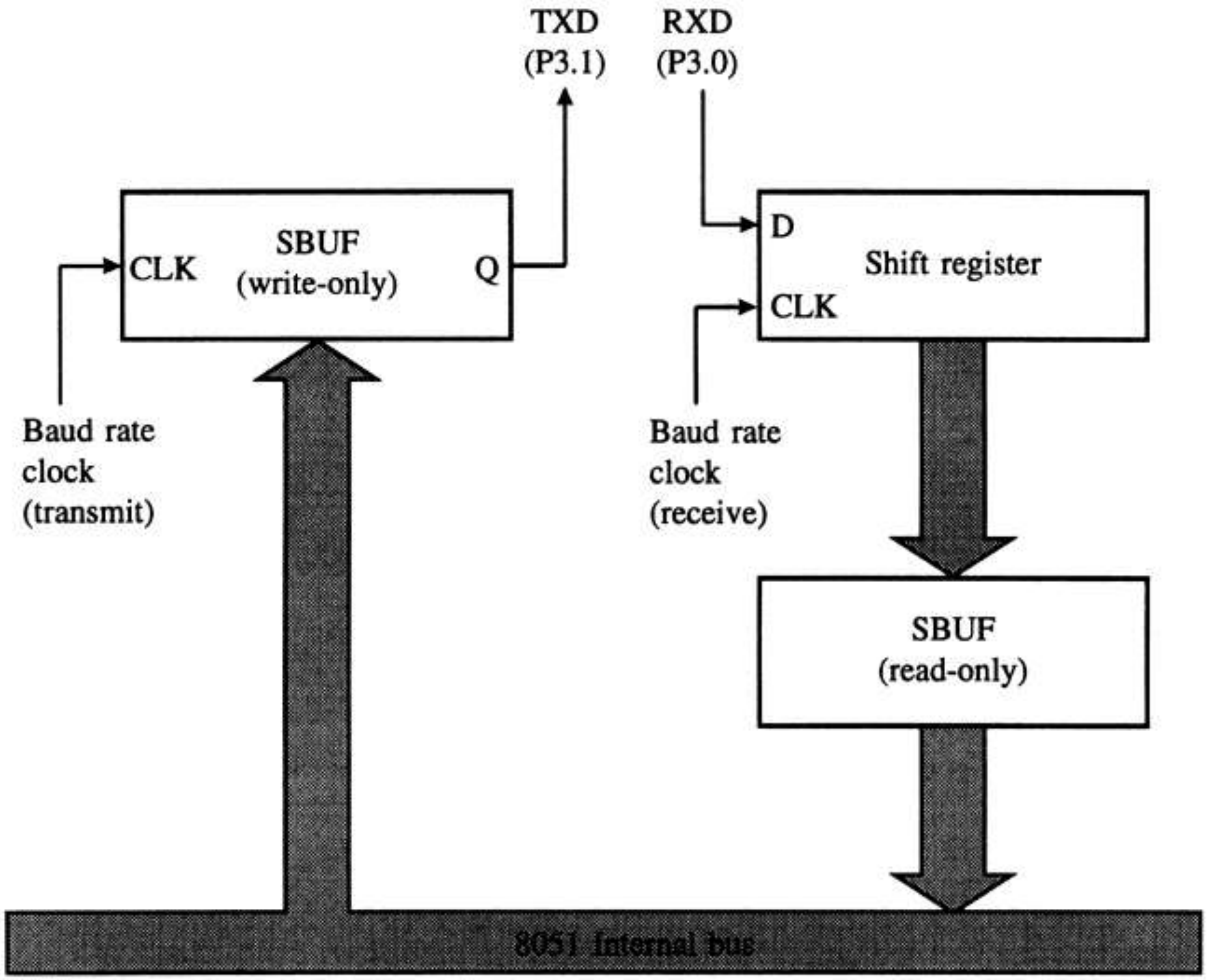


FIGURE 5–1
Serial port block diagram

SCON Register

- Serial control register: **SCON**

SM0, **SM1**	Serial port mode specifier
REN	(Receive enable) set/cleared by software to enable/disable reception.
TI	Transmit interrupt flag(set by HW & clear by SW)
RI	Receive interrupt flag (set by HW & clear by SW)

SM2 = **TB8** = R**B8** =0 (not widely used)

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

* SCON is bit-addressable.

REN (Receive Enable)

- SCON.4
- Set/cleared by software to enable/disable reception.
  - REN=1
    - It enable the 8051 to receive data on the RxD pin of the 8051**.**
    - If we want the 8051 to both transfer and receive data, REN must be set to 1.
    - **SETB SCON.4**
  - REN=0
    - The receiver is **disabled.**
    - The 8051 can not receive data.
    - **CLR SCON.4**

TB8 (Transfer Bit 8)

- SCON.3
- TB8 is used for serial modes 2 and 3.
- The 9th bit that will be transmitted in mode 2 & 3.
- Set/Cleared by software.

RB8 (Receive Bit 8)

- SCON.2
- In serial mode 1, RB8 gets a copy of the stop bit when an 8-bit data is received

TI (Transmit Interrupt Flag)

- SCON.1
- When the 8051 finishes the transfer of the 8-bit character, it raises the TI flag.
- TI is raised by hardware at the beginning of the stop bit in mode 1.
- Must be cleared by software.

RI (Receive Interrupt)

- SCON.0
- Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.
- When the 8051 receives data serially via RxD, it gets rid of the start and stop bits and place the byte in the SBUF register.
- Then 8051 rises RI to indicate that a byte.
- RI is raised at the beginning of the stop bit.

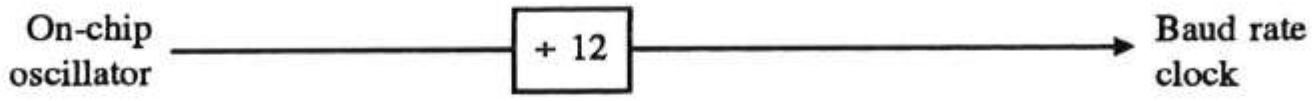
SM0, SM1

- SM1 and SM0 determine the framing of data.
  - SCON.6 (SM1) and SCON.7 (SM0)
  - Only mode 1 is compatible with COM port of PC.

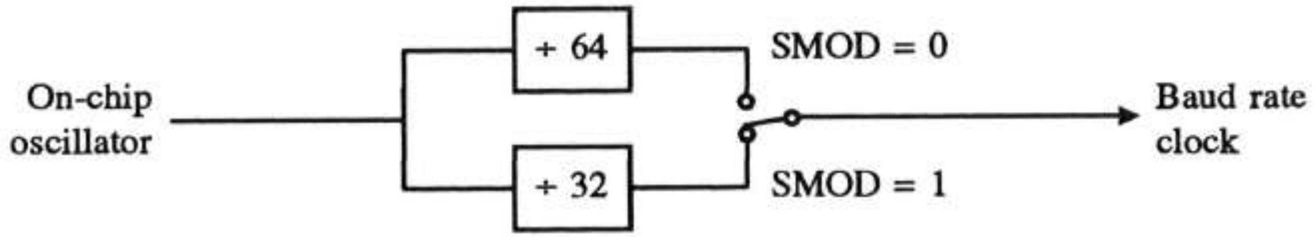
SM1	SM0	Mode	Operating Mode	Baud Rate
0	0	**0**	Shift register	Fosc./12
0	1	**1**	**8-bit UART**	**Variable by timer1**
1	0	**2**	9-bit UART	Fosc./64 or Fosc./32
1	1	**3**	9-bit UART	Variable

- Mode 0 :
  - Serial data enters and exits through RxD
  - TxD outputs the shift clock.
  - 8 bits are transmitted/received(LSB first)
  - The baud rate is fixed a 1/12 the oscillator frequency.

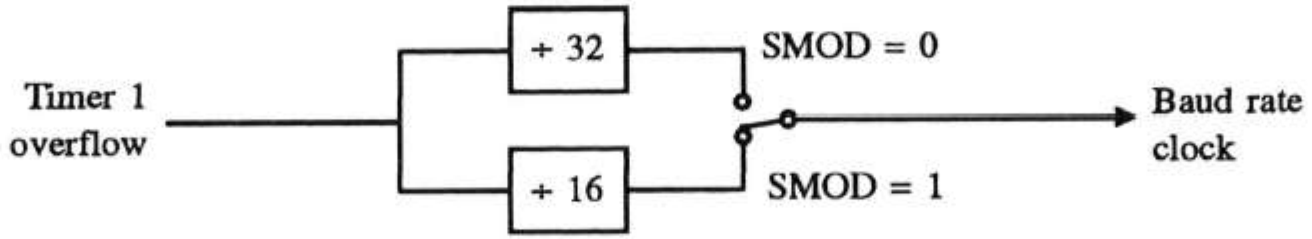
- Mode 1
  - Ten bits are transmitted (through TxD) or received (through RxD)
  - A start bit (0), 8 data bits (LSB first), and a stop bit (1)
  - On receive, the stop bit goes into RB8 in SCON
  - the baud rate is determined by the Timer 1 overflow rate.
  - Timer1 clock is 1/32 machine cycle (MC=1/12 XTAL)
  - Timer clock can be programmed as 1/16 of machine cycle
  - Transmission is initiated by any instruction that uses SBUF as a destination register.



(a) Mode 0



(b) Mode 2



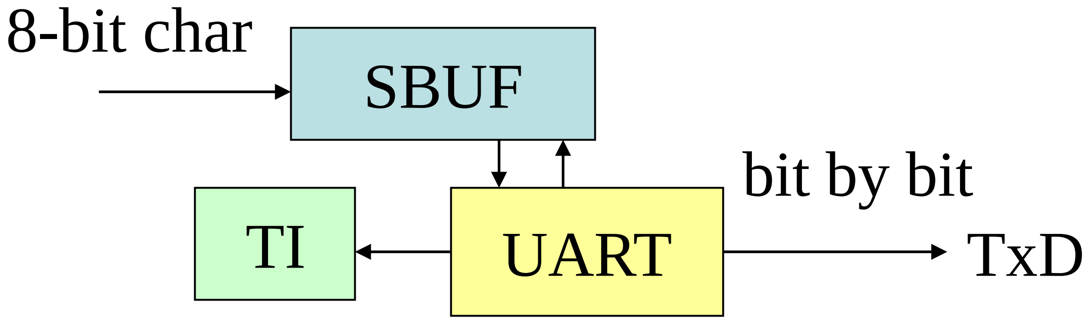
(c) Modes 1 and 3

- Mode 2 :
  - **Eleven bits are transmitted (through TxD), received (through RxD)**
    - **A start bit (0)**
    - **8 data bits (LSB first)**
    - **A programmable 9th data bit**
    - **and a stop bit (1)**
  - **On transmit, the 9th bit (TB8) can be assigned 0 or 1**
  - **On receive, the 9th data bit goes into RB8 in SCON.**
  - **the 9th can be parity bit**
  - **The baud rate is programmable to 1/32 or 1/64 the oscillator frequency in Mode 2 by SMOD bit in PCON register**

- Mode 3
  - Same as mode 2
  - But may have a variable baud rate generated from Timer 1.

Importance of TI flag (1/2)

- The following sequence is the steps that the 8051 goes through in transmitting a character via TxD:
  1. The byte character to be transmitted is written into the SBUF register.
  2. It transfers the **start bit**.
  3. The **8-bit character** is transferred one bit at a time.
  4. The **stop bit** is transferred.



- Sequence continuous:

5. During the transfer of the stop bit, the 8051 raises the TI flag, indicating that the last character was transmitted and it is ready to transfer the next character.
6. By monitoring the **TI** flag, we know whether or not the 8051 is ready to transfer another byte.
   - We will not overloading the SBUF register.
   - If we write another byte into the SBUF before TI is raised, the untransmitted portion of the previous byte will be lost.
7. After SBUF is loaded with a new byte, the TI flag bit must be cleared by the programmer.

Steps to program the 8051 to transfer data serially

1. The TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 to set the band rate.
2. The TH1 is loaded with one of the values given in table below to set the band rate.

Band rate	TH1(HEX)	TH1(Decimal)
9600	FD	- 3
4800	FA	- 6
2400	F4	- 12
1200	E8	- 24

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

Example

1. Write a program for the 8051 to transfer letter “A” serially at 4800baud, continuously.

```
        MOV   TMOD,#20H    ;timer 1, mode 2
        MOV   TH1,#-6      ;4800 baud rate
        MOV   SCON,#50H    ;8-bit,1 stop,REN enabled
        SETB  TR1          ;start timer 1
AGAIN:  MOV   SBUF,#"A"    ;letter “A” to be transferred
HERE:   JNB   TI,HERE      ;wait for the last bit
        CLR   TI           ;clear TI for next char
        SJMP  AGAIN        ;keep sending A
```

2. Write a program to transfer the message “YES” serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

```
      MOV   TMOD,#20H ;timer 1, mode 2
      MOV   TH1,#-3    ;9600 baud
      MOV   SCON,#50H
      SETB  TR1
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 last bit to transfer
      CLR   TI         ;get ready for next byte
      RET
```

3. Write a program to send a message “Global Academy Of Technology” to the PC serially.

```
          0rg 0
         mov Tmod, #20h            ;      Timer 1, mode 2
          mov TH1 , #0FAh          ;   4800 baud rate
          mov SCON, # 50h          ;  8bit, 1stop, REN enabled
          SETB TR1
          mov DPTR , # MYDATA
L1:       CLR A
          movc A, @A+DPTR
          JZ L3                    ; check for last character
          ACALL SEND
          INC DPTR
          SJMP L1
SEND:     mov SBUF , A
L2      : JNB TI, L2
          CLR TI
          RET
MYDATA DB “Global Academy Of Technology”, 0
L3       : END
```

4. Assume a switch is connected to pin P1.7. Write a program to monitor its status & send two messages to serial port continuously as follows: SW=0 send “NO”,SW=1 send “YES”.

```
          Sw1 EQU P1.7
          org 00h
Main:mov TMOD , #20h
        mov TH1, # -3                    ;9600 baud rate
        mov SCON , #50h
        SETB TR1
        SETB SW1                      ;  make SW an i/p
S1:   JB P1.7, next                   ;  if SW=0, Display “NO”
        mov DPTR, # mess1
FN:   CLR A
        movC A, @a+DPTR
        JZ S1
        ACALL SENDCOM
        INC  DPTR
        SJMP FN
```

```
Next:           mov DPTR, #mess2    ;if SW=1, display “YES”
LN:             CLR A
                movC A, @A+DPTR
                JZ S1
                 ACALL SENDCOM
                 INC DPTR
                 SJMP LN
SENDCOM: mov SBUF ,A
Here:           JNB TI, Here
                CLR TI
                RET
Mess1      DB ‘NO’, 0
Mess2      DB ‘YES’, 0
                END
```

5. A PC is connected to an 8051 system using RS232 interface. Using a 6.0MHz clock for the 8051, design the 8051 program that will transmit the word ‘impossible’ when the PC sends a word ‘mission’.

With XTAL = 6.0 MHz, we have:

The frequency of system clock = 6.0 MHz / 12 = 500 kHz

The frequency sent to timer 1 = 500 kHz/ 32 = 15625 Hz

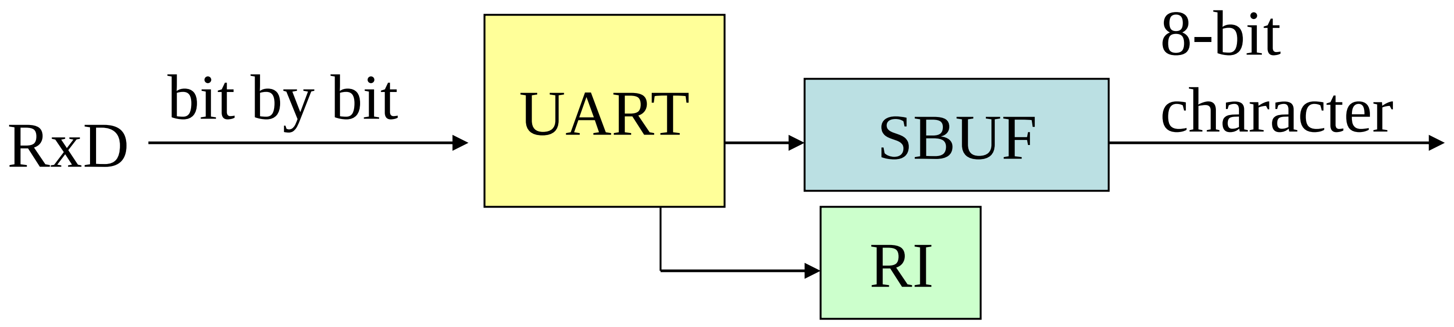
15625 / 142 = 110   where 142 =8E (hex)

```
      ORG 100
Sta: MOV SCON, #50H
      MOV TMOD, #20H
      MOV TH1, 8EH
      SETB TR1
Rec: MOV R0, #00
      MOV DPTR, #MSG1
Wait: JNB RI, wait
      MOV A, SBUF
      MOV P1,A
```

```
    CLR RI
    CLR A
    MOVC A, @A+DPTR
    CJNE A, 01, sta        ; compare values from table with received byte
    INC DPTR
    INC R0
    CJNE R0, #07, wait   ; receive 7 characters
    MOV DPTR #MSG2
    MOV R0, #00
Wr: CLR A
    MOVC A, @A+DPTR
    MOV SBUF, A
Tr1: JNB TI, tr1
    CLR TI
    INC R0
    CJNE R0, #0AH, wr   ; check if all characters are written ‘ impossible’
    SJMP rec

MSG1  DB ;mission’
MSG2  DB ‘impossible’
```

Importance of the RI flag

- The following sequence is the steps that the 8051 goes through in receiving a character via RxD:
  1. 8051 receives the **start bit** indicating that the next bit is the first bit of the character to be received.
  2. The **8-bit character** is received one bit at a time. When the last bit is received, a byte is **formed** and placed in SBUF.



- Sequence continuous:

3. The **stop bit** is received. During receiving the stop bit, the 8051 make **RI=1**, indicating that an entire character was been received and **must be picked up** before it gets **overwritten** by an incoming character.
4. By monitoring the **RI** flag, we know whether or not the 8051 has received a character byte.
   - If we fail to copy SBUF into a safe place, we risk the loss of the received byte.
5. After SBUF is copied into a safe place, the RI flag bit must be cleared by the programmer.

Steps to program the 8051 to receive data serially

Step 1 ,2 ,3 ,4 are same as transmitter.

5. RI is cleared with CLR RI instruction.

6. The RI flag bit is monitored with JNB RI,XX to see if an entire character has been received yet.

7. When RI is raised, SBUF has a byte.

8. To receive the next character , go to step 5.

Example

1. Program 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  ;timer1, mode 2 (auto reload)
       MOV   TH1,#-6    ;4800 baud
       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     ;save incoming byte in A
       MOV   P1,A       ;send to port 1
       CLR   RI         ;get ready to receive next byte
       SJMP  HERE       ;keep getting data
```

Doubling the baud rate in the 8051

Two methods:

1. To use higher frequency crystal
2. To change a bit in the PCON register is D7 (SMOD)

First one is not flexible since crystal is fixed. The new crystal may not be compatible with the IBM PC serial comports baud rate.

When the 8051 is powered up D7(SMOD bit) if the PCON register is zero. It can be set to high by software and there by double the baud rate. It is not bit addressable

```
mov A, PCON
SETB ACC.7
mov PCON,A
```

When SMOD=0, Timer frequency=$\frac{11.0592\text{MHz}}{12} = 921.6\text{KHz}$

$$\frac{921.6\text{KHz}}{32} = 28800\text{Hz}$$

When SMOD=1, Timer frequency=$\frac{11.0592\text{MHz}}{12} = 921.6\text{KHz}$

$$\frac{921.6\text{KHz}}{16} = 57600\text{Hz}$$

The following table shows the values loaded into TH1 for both cases.

TH1(Decimal)	(HEX)	SMOD=0	SMOD=1
- 3	FD	9600	19200
- 6	FA	4800	9600
- 12	F4	2400	4800
- 24	F8	1200	2400

- Find the baud rate if TH1=- 2, SMOD=1& XTAL=11.0592MHz ,IS this baud rate supports by IBM /compatible PC's?

Timer 1 frequency = 57600Hz ,with SMOD=1

The baud rate $= \frac{57600}{2} = 28800$

This baud rate is not Supported by the BIOS of the PC

Power control register

MSB							LSB
SMOD	—	—	—	GF1	GF0	PD	IDL

BIT	SYMBOL	FUNCTION
PCON.7	SMOD	Double Baud rate bit. When set to a 1 and Timer 1 is used to generate baud rate, and the Serial Port is used in modes 1, 2, or 3.
PCON.6	—	Reserved.
PCON.5	—	Reserved.
PCON.4	—	Reserved.
PCON.3	GF1	General-purpose flag bit.
PCON.2	GF0	General-purpose flag bit.
PCON.1	PD	Power-Down bit. Setting this bit activates power-down operation.
PCON.0	IDL	Idle mode bit. Setting this bit activate idle mode operation.

What is SMOD

- Bit 7 of PCON register
- If SMOD=1 double baud rate
- PCON is not bit addressable
- How to set SMOD
    - **Mov a, pcon**
    - **Setb acc.7**
    - **Mov pcon,a**

Example:

1. Write a program to send the message “The Earth is One Country” to serial port. Assume a SW is connected to pin P1.2. Monitor its status and set the baud rate as follows.

   SW=0 ; 4800 baud rate

   SW=1 ; 9600 baud rate

Assume XTAL=11.0592 MHz, 8bit data , and 1 stop bit.

```
           org 0h                     ;starting position
Main:      mov TMOD , # 20h
           mov TH1, # -6              ;4800 baud rate (default)
           mov SCON, #50h
           SETB TR1
           SETB P1.2                  ;make SW an input
S1:        JNB  P1.2, SLOWSP ; check SW status
```

```
           mov A, PCON             ;  read PCON
           SETB  ACC.7             ; set SMOD High for  9600
           mov   PCON ,A           ; write PCON
           SJMP FN                 ; send message
SLOWSP: mov A, PCON                ; read PCON
           CLR ACC.7               ; make SMOD low for 4800
           mov PCON , A            ; write PCON
FN:        CLR A
           movc A , @A+DPTR        ; read value
           JZ S1                   ; check for end of line
           ACALL SENDCOM           ; send  value to the serial port
           INC DPTR                ; move to next value
           SJMP   FN               ; Repeat
SENDCOM: mov SBUF , A              ; place value in buffer
Here:       JNB  T1, Here          ; wait until transmitted
            CLR T1                 ;clear
            RET                    ; return

Mess1  DB  “The Earth is One Country” , 0
          END
```

4. Write a program to transfer ASCII character ‘B’ continuously. Compute the frequency used by timer 1 to set the baud rate. Also find the baud rate of the data transfer.

```
Mov A, pcon
Setb Acc.7
Mov pcon,A
Mov tmod, #20h
Mov th1, #-3
Mov scon, #50h
Setb tr1
```

```
        Mov A,#’B’
Loop: clr TI
        Mov sbuf, A
   L1: jnb TI, l1
        Sjmp loop
```

With smod=1,   921.6 khz/16 = 57600hz

Baud rate of the data transfer is 57600/3 = 19200