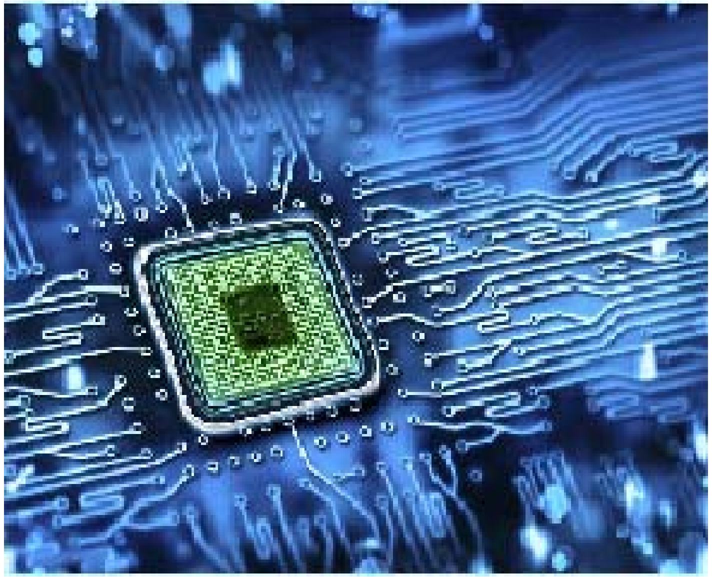
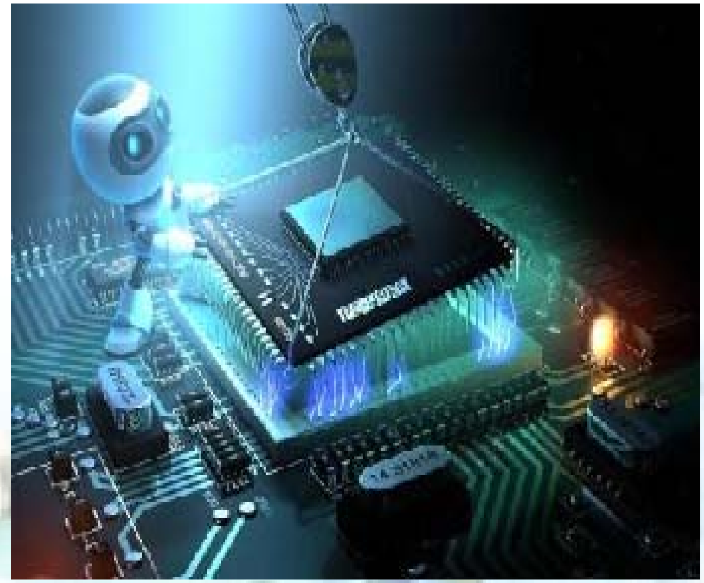
16CS402 - Embedded Systems

VIGNAN'S
Foundation for Science, Technology & Research
(Deemed to be UNIVERSITY)
-Estd. u/s 3 of UGC Act 1956

4th Year 1st Semester



CSE
Unit-2



Dr. V. VIJAYARAGHAVAN, M.E., Ph.D.,

Assistant Professor - ECE, VFSTR, Guntur, AP.

UNIT - II

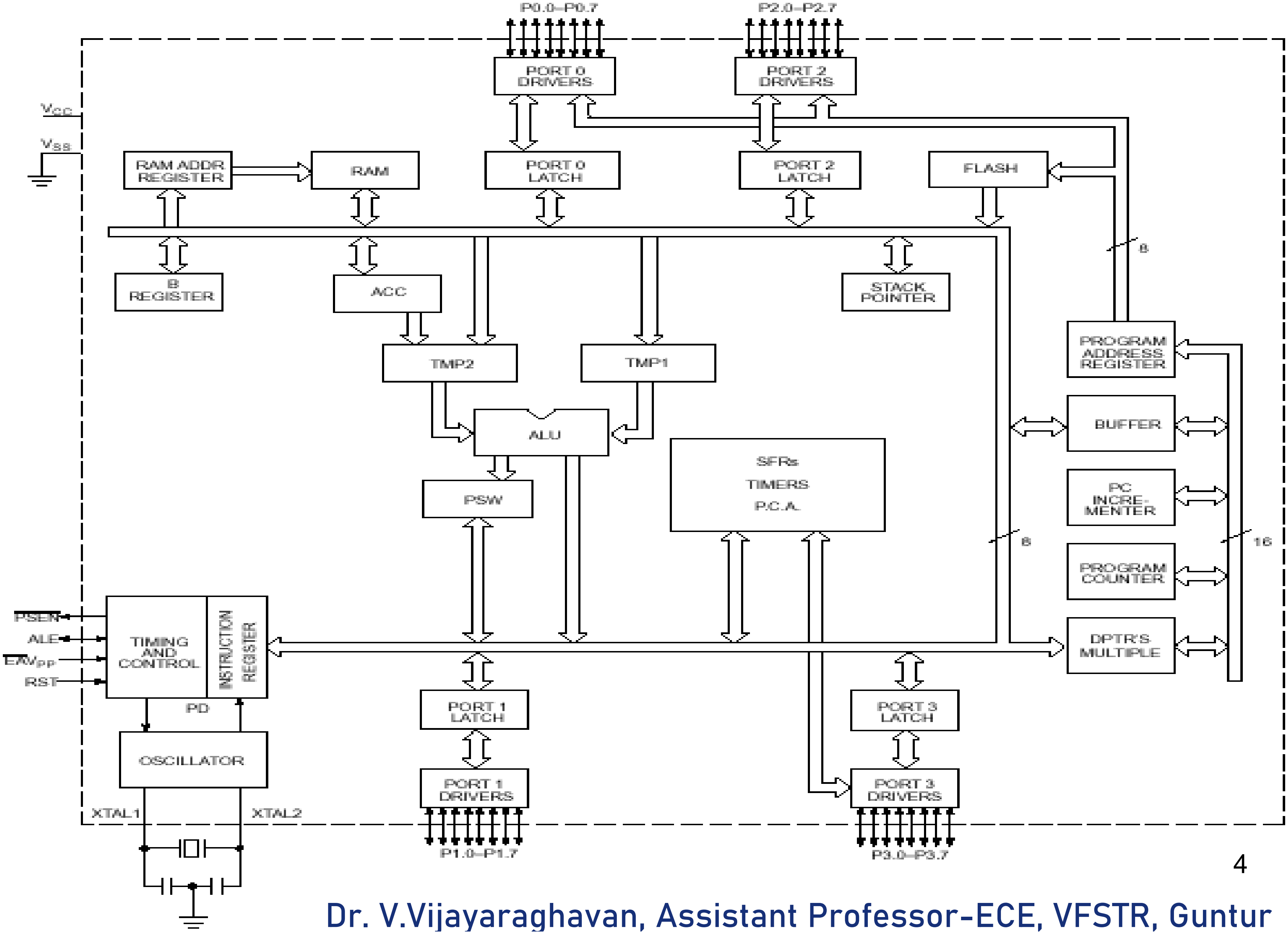
Architecture of 8051 and its Programming with C

- 8051 Microcontroller Hardware
- Input, Output Ports and Circuits
- External Memory
- Counters and Timers, Serial Data Input/Output
- Interrupts, Programming 8051 using Embedded C

8051 Basic Component **(Features)**

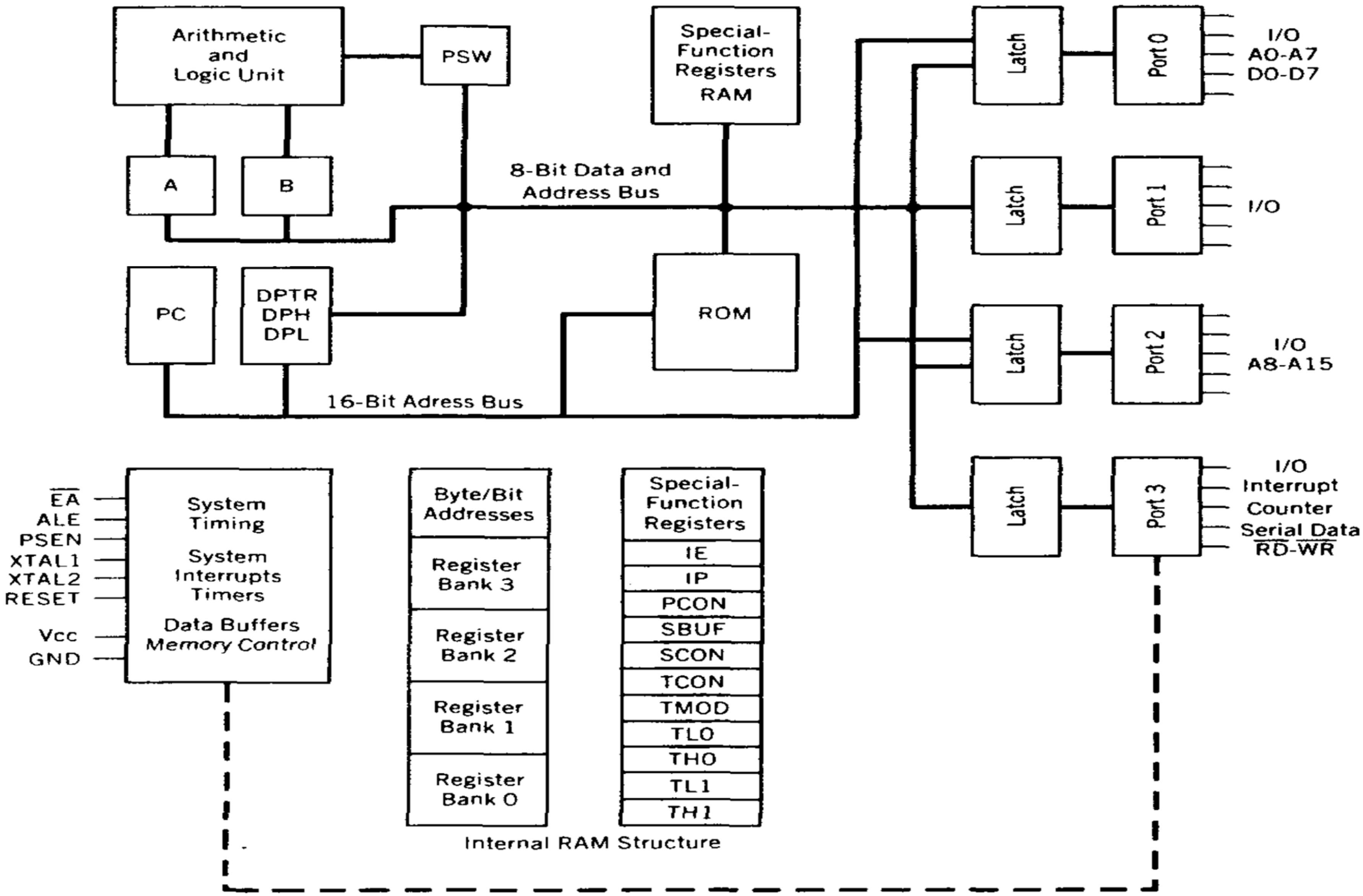
- 8 bit Microcontroller from Intel in 1983.
- 128 bytes internal RAM **(Data Memory)**
- 4K bytes internal ROM **(Program Memory)**
- 32bit bidirectional I/O ports->Four 8-bits **(P0,P1,P2,P3)**.
- Two 16-bit timers/counters **(T0, T1)**
- One serial interface **(UART)**
- 5 Interrupt sources **($\overline{\text{INT0}}$, $\overline{\text{INT1}}$, TF0, TF1, TI/RI)**
- Clock frequency is 11.0592MHz
- Externally it has 8bit data bus & 16 bit address bus

8051 Internal Block Diagram

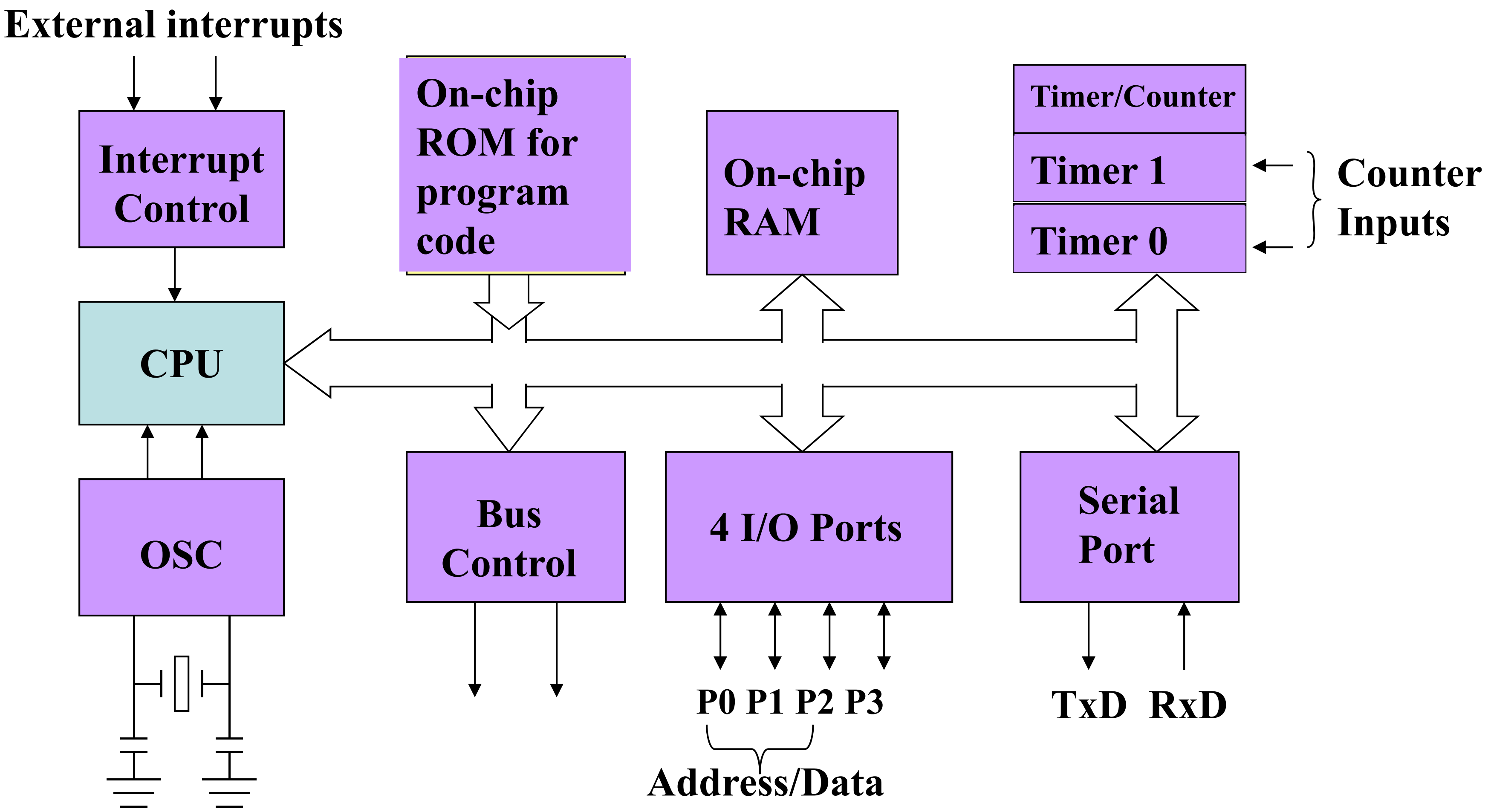


8051 Internal Block Diagram

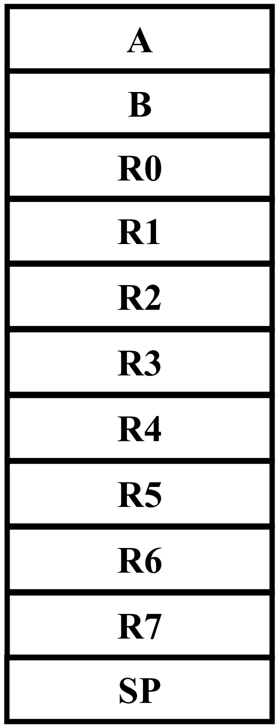
FIGURE 2.1a 8051 Block Diagram



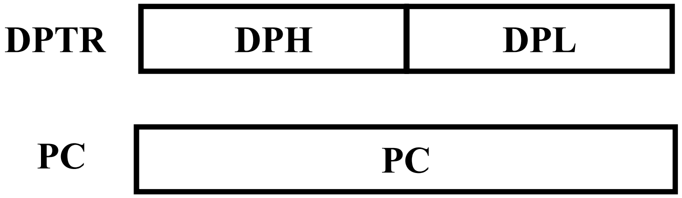
Block Diagram



8051 CPU Registers



Some 8-bit Registers



Some 8051 16-bit Register

Accumulator (A) and B Register

- ***A*** register is a 8 bit most versatile CPU register and is used for many operations, including addition, integer multiplication and division, and Boolean bit manipulations.
- ***A*** register is also used for all data transfer between the 8051 and any external memory.
- ***B*** register is used with the ***A*** register for multiplication and division operations.

  (eg. MUL AB   DIV AB)

Data Pointer Register

- ***DPTR*** is a 16-bit register
- ***DPTR*** is made up of two 8-bit registers: ***DPH*** and ***DPL***
- ***DPTR*** holds the memory addresses for internal and external code access and external data access

  (eg. MOVC A,@A+DPTR MOVX A,@DPTR MOVX @DPTR,A)
- ***DPTR*** is under the control of program instructions and can be specified by its 16-bit name, or by each individual byte name, ***DPH*** and ***DPL***
- ***DPTR*** does not have a single internal address; ***DPH*** and ***DPL*** are each assigned an address (83H and 82H)

Program Counter

- ***PC*** is a 16-bit register
- ***PC*** is the only register that does not have an internal address
- Holds the address of the next executable instructions.
- Program ROM may be on the chip at addresses 0000H to 0FFFH (4Kbytes), external to the chip for addresses that exceed 0FFFH
- Program ROM may be totally external for all addresses from 0000H to FFFFH
- ***PC*** is automatically incremented (+1) after every instruction byte is fetched

Stack Pointer

- ***SP*** is a 8-bit register used to hold an internal RAM address that is called the "***bottom of the stack***"
- ***Stack*** refers to an area of internal RAM that is used in to store and retrieve data quickly
- ***SP*** holds the internal RAM address where the last byte of data was stored by a stack operation
- When data is to be placed on the stack, the ***SP*** increments before storing data on the stack so that the stack ***grows up*** as data is stored
- As data is retrieved from the stack, the byte is read from the stack, and then the ***SP*** decrements to point to the next available byte of stored data
- ***SP = 07H*** after reset.

Program Status Word (PSW) - Flag Register

PSW contains the math flags, user program flag ***F0***, and the register select bits (RS1, RS0) that identify which of the four general-purpose register banks is currently in use by the program

D7	D6	D5	D4	D3	D2	D1	D0
CY	**AC**	**F0**	**RS1**	**RS0**	**OV**	**--**	**P**

Bit	**Symbol**	**Function**
7	CY	Carry Flag; used in arithmetic, JMP, ROTATE, and BOOLEAN instructions
6	AC	Auxiliary carry flag; used for BCD arithmetic
5	F0	User flag 0
4	RS1	Register bank select bit 1
3	RS0	Register bank select bit 0
2	OV	Overflow flag; used in arithmetic instructions
1	--	Reserved for future use
0	P	Parity flag; shows parity of register A after ALU

Special Function Registers (SFR)

- 8051 has 21 ***SFR***s which occupy the addresses from 80H to FFH (128bytes)
- Not all of the addresses from 80H to FFH are used for ***SFR***s
- Attempt to use the “empty” addresses may get unpredictable result

Bit addressable

F8								
F0	B							
E8								
E0	ACC							
D8								
D0	PSW							
C8								
C0								
B8	IP							
B0	P3							
A8	IE							
A0	P2							
98	SCON	SBUF						
90	P1							
88	TCON	TMOD	TL0	TL1	TH0	TH1		
80	PO	SP	DPH	DPL				PCON

8051
Foot Print

IMPORTANT PINS:

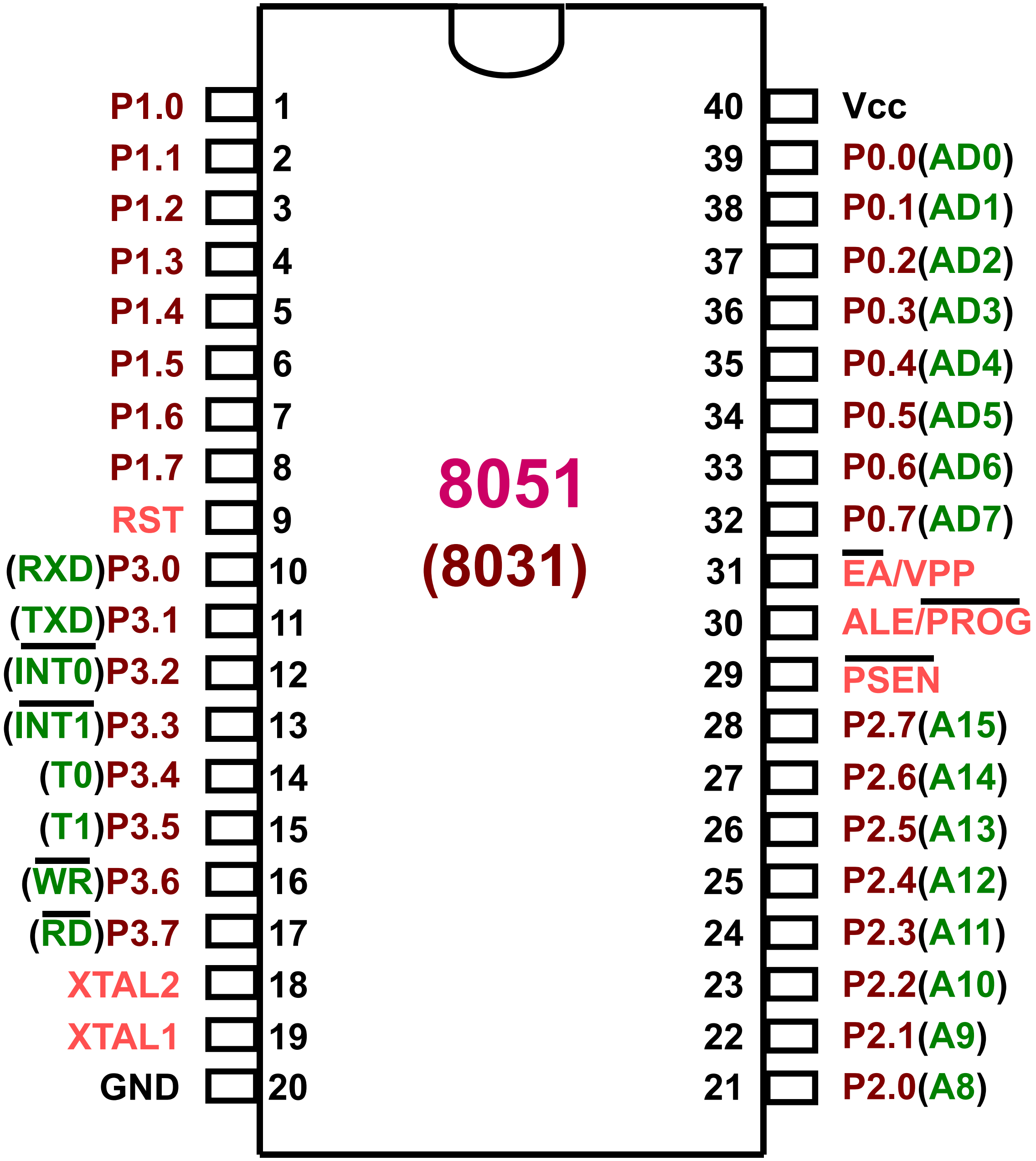
PSEN’ (out): Program Store Enable, the read signal for external program memory (active low).

ALE (out): Address Latch Enable, to latch address outputs at Port0 and Port2

EA’ (in): External Access Enable, active low to access external program memory locations 0 to 4K

RXD,TXD: UART pins for serial I/O on Port 3

XTAL1 & **XTAL2**: Crystal inputs for internal oscillator.



8051
(8031)

P1.0	1	40	Vcc
P1.1	2	39	P0.0(AD0)
P1.2	3	38	P0.1(AD1)
P1.3	4	37	P0.2(AD2)
P1.4	5	36	P0.3(AD3)
P1.5	6	35	P0.4(AD4)
P1.6	7	34	P0.5(AD5)
P1.7	8	33	P0.6(AD6)
RST	9	32	P0.7(AD7)
(RXD)P3.0	10	31	$\overline{\text{EA}}$/VPP
(TXD)P3.1	11	30	ALE/$\overline{\text{PROG}}$
($\overline{\text{INT0}}$)P3.2	12	29	$\overline{\text{PSEN}}$
($\overline{\text{INT1}}$)P3.3	13	28	P2.7(A15)
(T0)P3.4	14	27	P2.6(A14)
(T1)P3.5	15	26	P2.5(A13)
($\overline{\text{WR}}$)P3.6	16	25	P2.4(A12)
($\overline{\text{RD}}$)P3.7	17	24	P2.3(A11)
XTAL2	18	23	P2.2(A10)
XTAL1	19	22	P2.1(A9)
GND	20	21	P2.0(A8)

8051 IO PORTS

Input, Output Ports and Circuits (P0 - P3)

One of the most useful features of the 8051 is that it consists of 4 I/O ports (P0 - P3)

- All ports are bit addressable
- On **RESET** all the ports are configured as *output*
- When a bit latch is to be used as an *input*, a "**1**" *must be* written to the corresponding latch by the program to configure it as input (eg. MOV P1, #0FFH)

Port 0 （pins 32-39） ：P0（P0.0～P0.7）
8-bit R/W - General Purpose I/O
Or acts as a multiplexed low byte address and data bus for external memory design

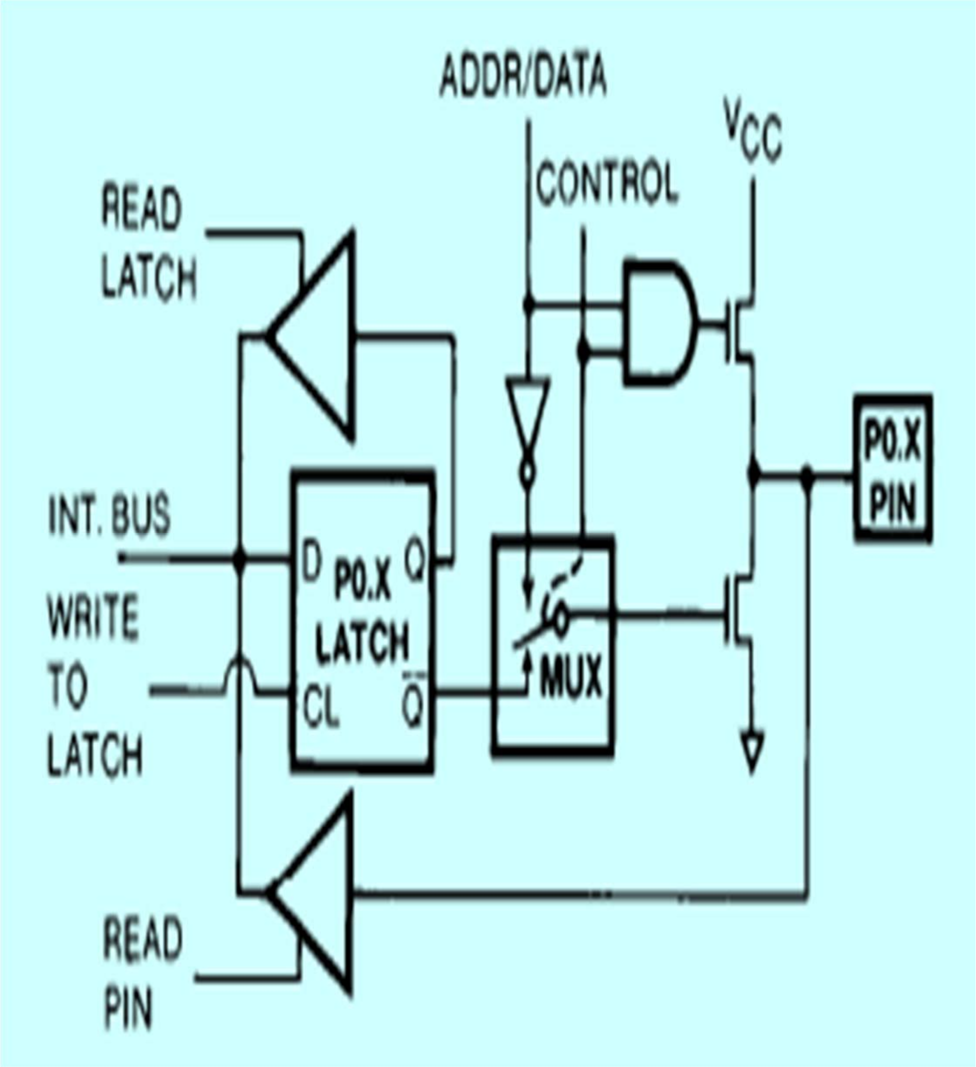
Port 1 （pins 1-8） ：P1（P1.0～P1.7）
Only 8-bit R/W - General Purpose I/O

Port 2 （pins 21-28） ：P2（P2.0～P2.7）
8-bit R/W - General Purpose I/O
Or high byte of the address bus for external memory design

Port 3 （pins 10-17） ：P3（P3.0～P3.7）
General Purpose I/O
Or acts as 8 different function (serial, external interrupts, external timers)

(Port 0 Configurations)

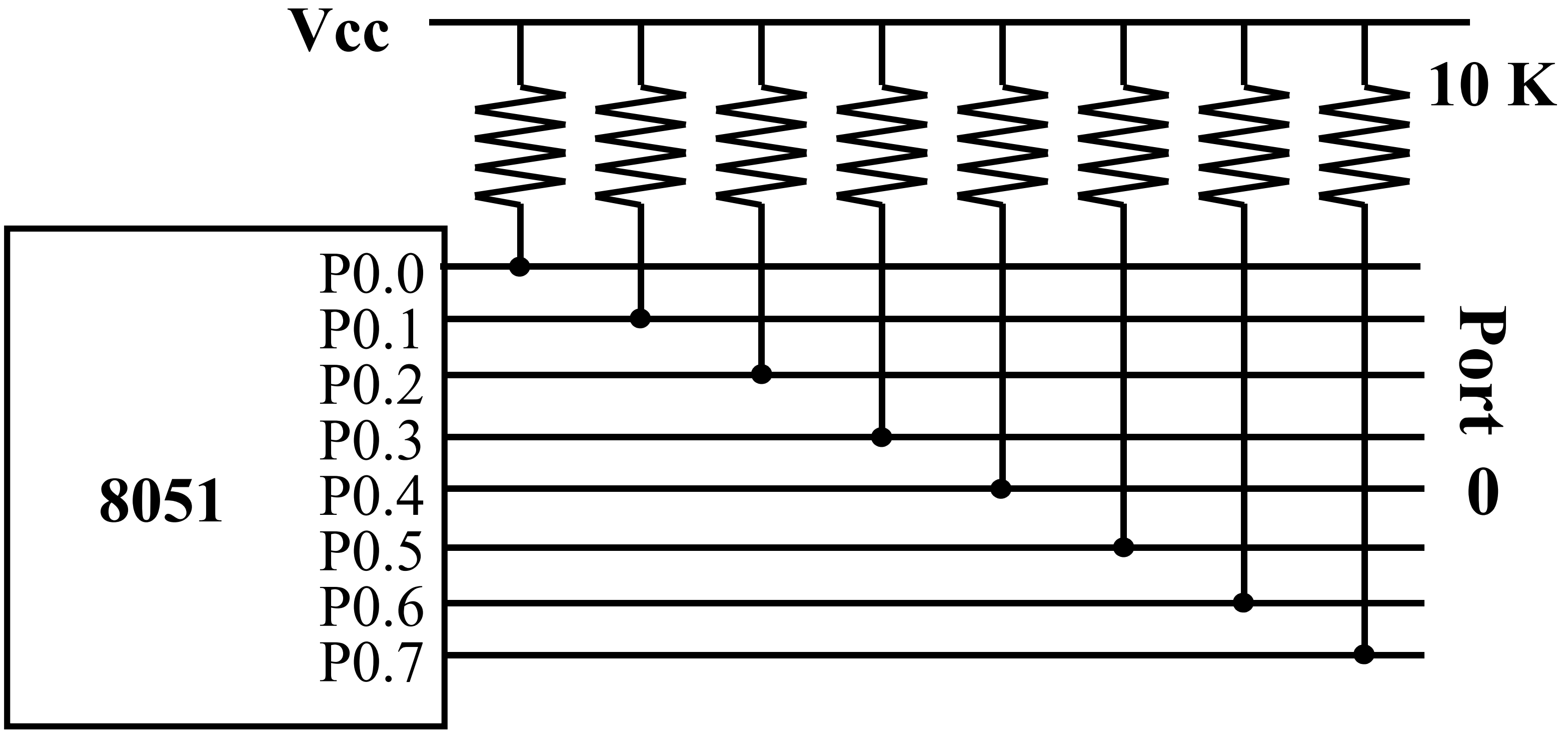
- Occupies a total of 8 pins (Pins 32-39)
- Can be used for :
  - Input only
  - Output only
  - Input and output at the same time (i.e. some pins for input and the others for output)
- Can be used to handle both address and data
- Need pull-up resistors



Hardware Structure of I/O Pin

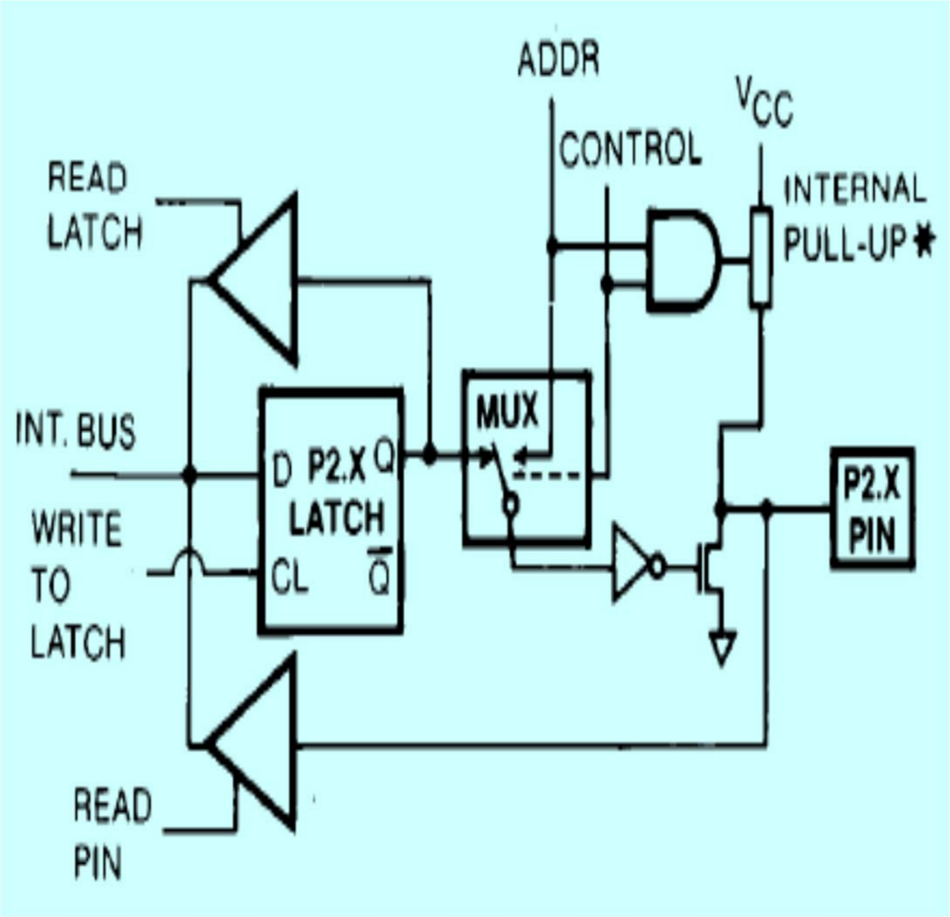
- Each pin of I/O ports
  - Internally connected to CPU bus
  - A D latch store the value of this pin
    - Write to latch＝1：write data into the D latch
  - 2 Tri-state buffer：
    - TB1: controlled by “Read pin”
      - Read pin＝1：really read the data present at the pin
    - TB2: controlled by “Read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

Port 0 with Pull-Up Resistors

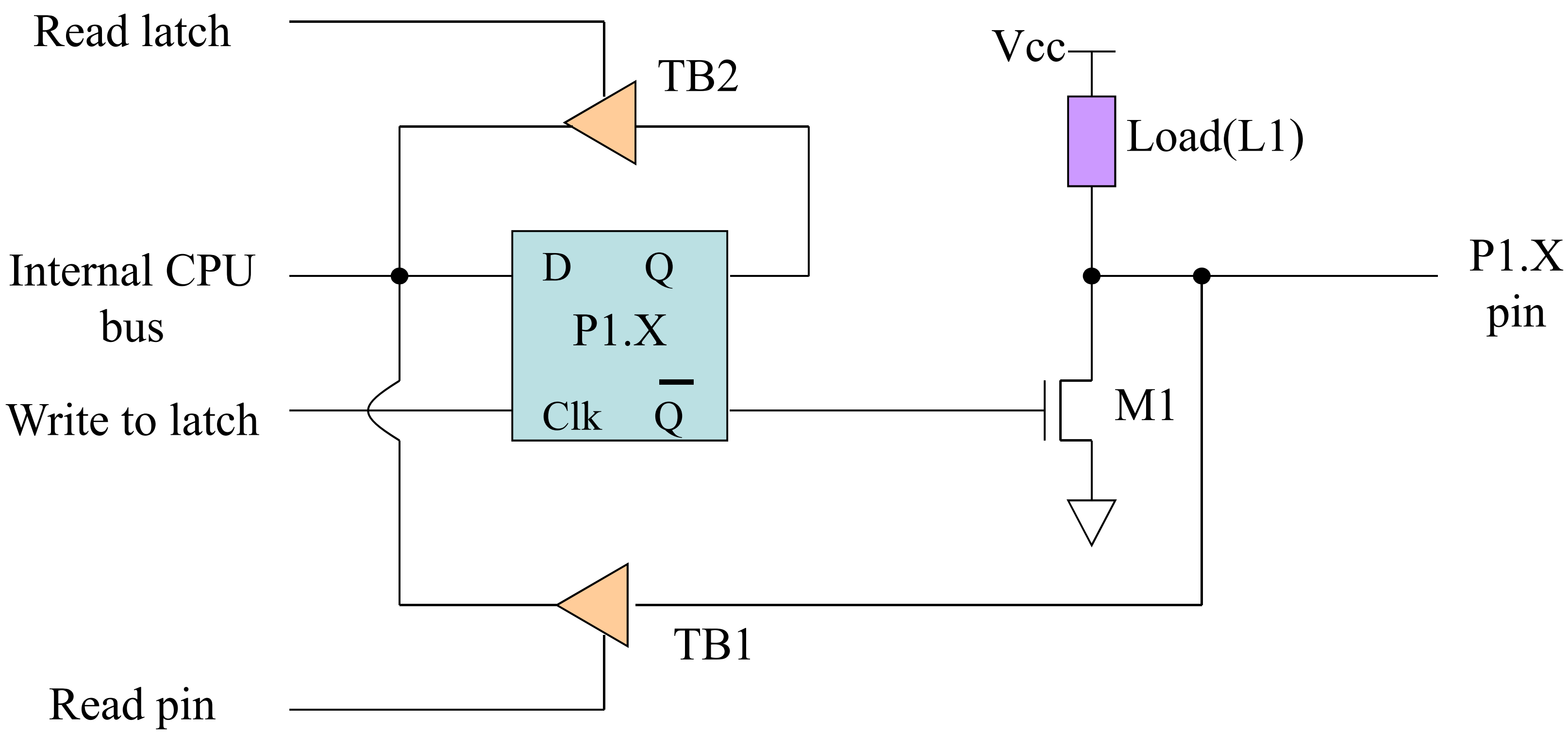


(Port 2 Configurations)

- Occupies a total of 8 pins (Pins 21-28)
- Similar function as Port 1
- Can be used as input or output
- Does not need any pull-up resistors
- Upon reset, port 2 is configured as an output port

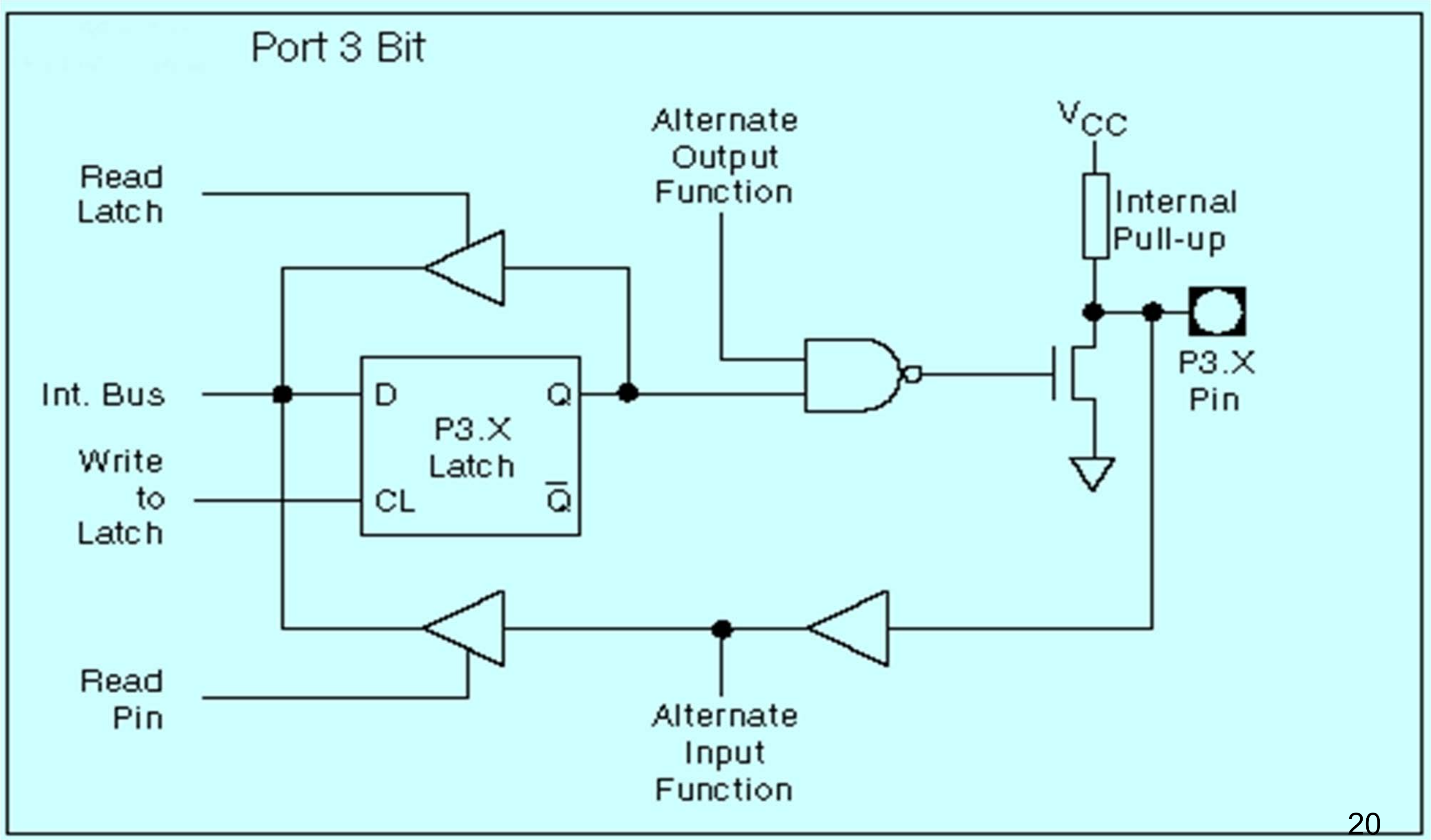


(Port 1 Configurations)

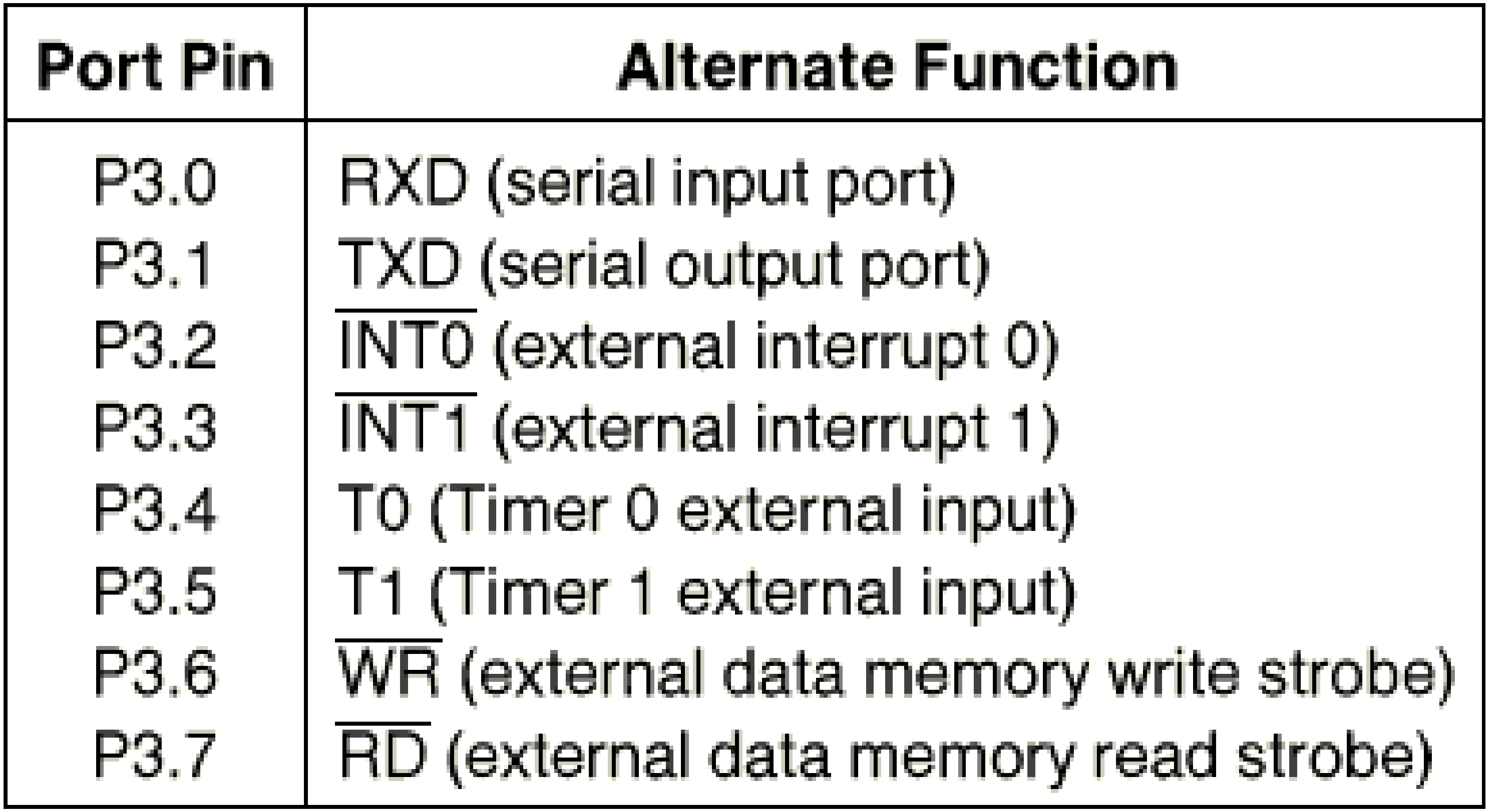


- Occupies a total of 8 pins (Pins 1-8)
- Can be used for :
  - Input only or Output only or Input and output at the same time (i.e. some pins for input and the others for output)

Port 3 Bit Latches and I/O Buffers
(Port 3 Configurations)



Port 3 Alternate Functions

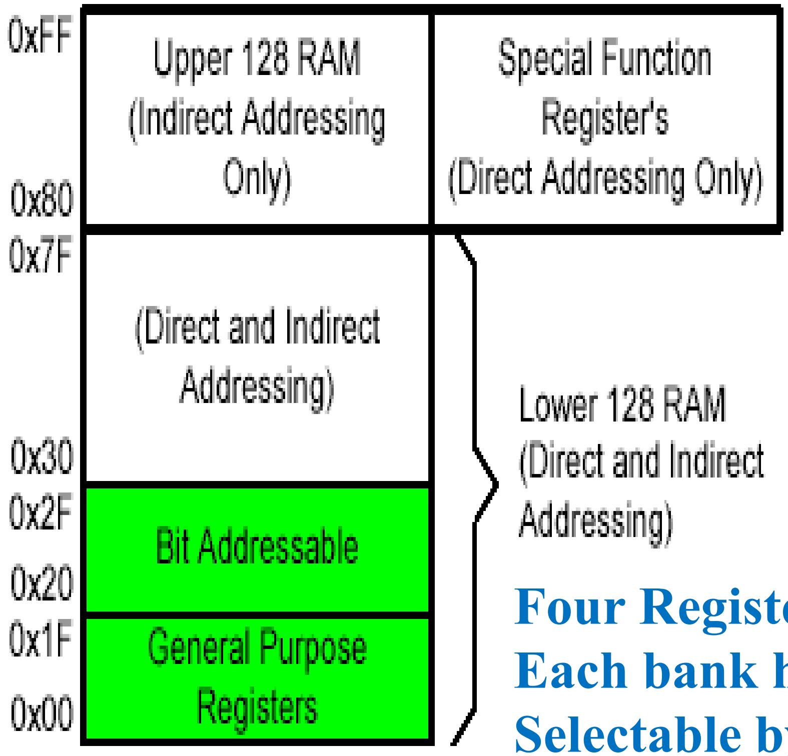


Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

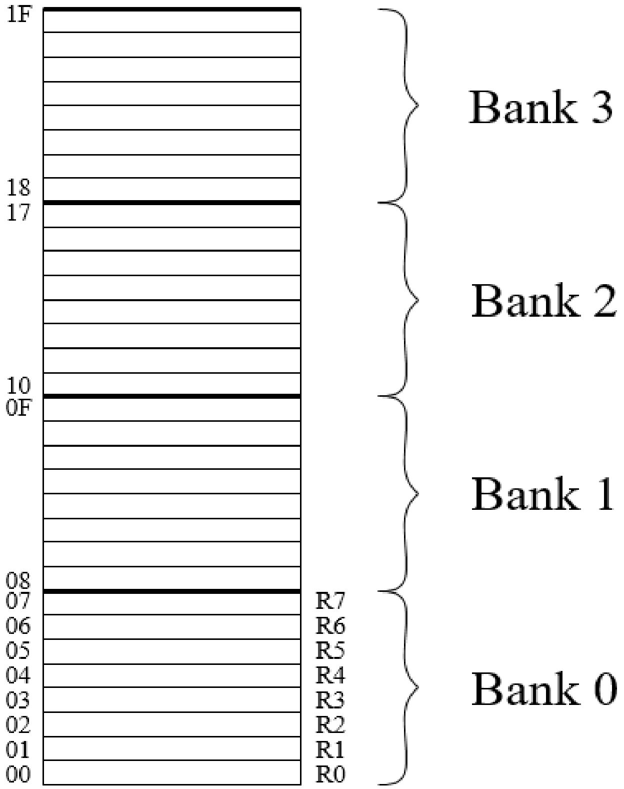
8051 Memory

Internal Memory

- A functioning computer must have memory for program code bytes, commonly in ROM, and RAM memory for variable data that can be altered as the program runs
- 8051 has internal RAM (128 bytes) and ROM (4Kbytes)
- 8051 uses the same address but in different memories for code and data
- Internal circuitry access the correct memory based on the nature of the operation in progress
- Can add memory externally if needed (up to 64Kbytes)


0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

Four Register Banks. Each bank has R0-R7 Selectable by psw.3,4



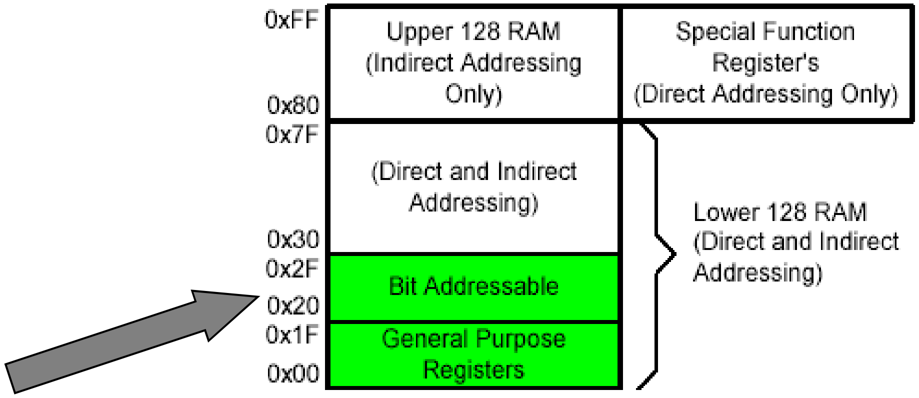
Bit Addressable Memory

2F	7F							78
2E								
2D								
2C								
2B								
2A								
29								
28								
27								
26								
25								
24								
23						1A		
22								10
21	0F							08
20	07	06	05	04	03	02	01	00

20h – 2Fh (16 locations X 8-bits = 128 bits)

Bit addressing:

mov C, 1Ah
or
mov C, 23h.2



0xFF

Upper 128 RAM (Indirect Addressing Only)

Special Function Register's (Direct Addressing Only)

0x80

0x7F

(Direct and Indirect Addressing)

0x30

0x2F

Bit Addressable

0x20

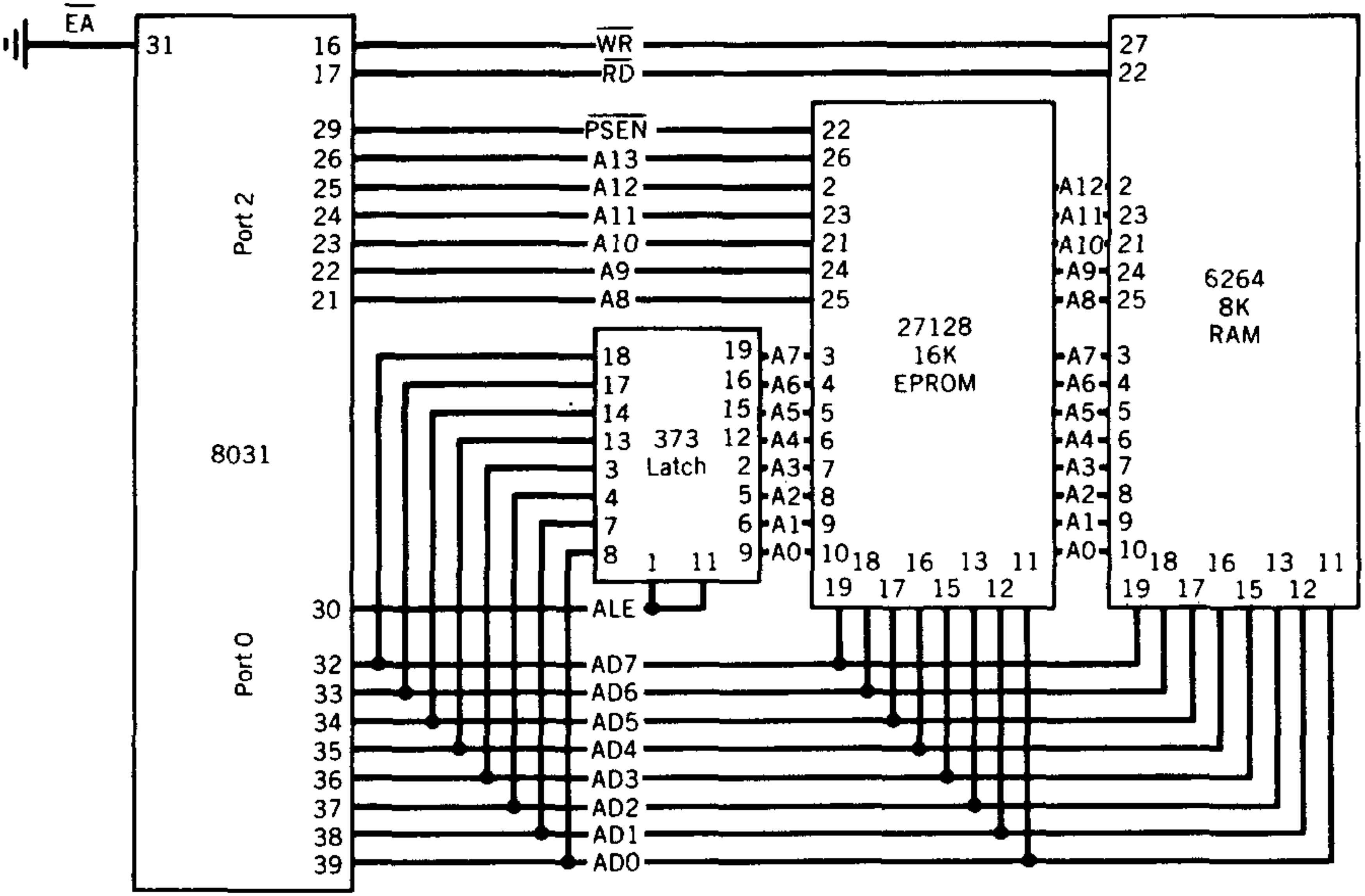
0x1F

General Purpose Registers

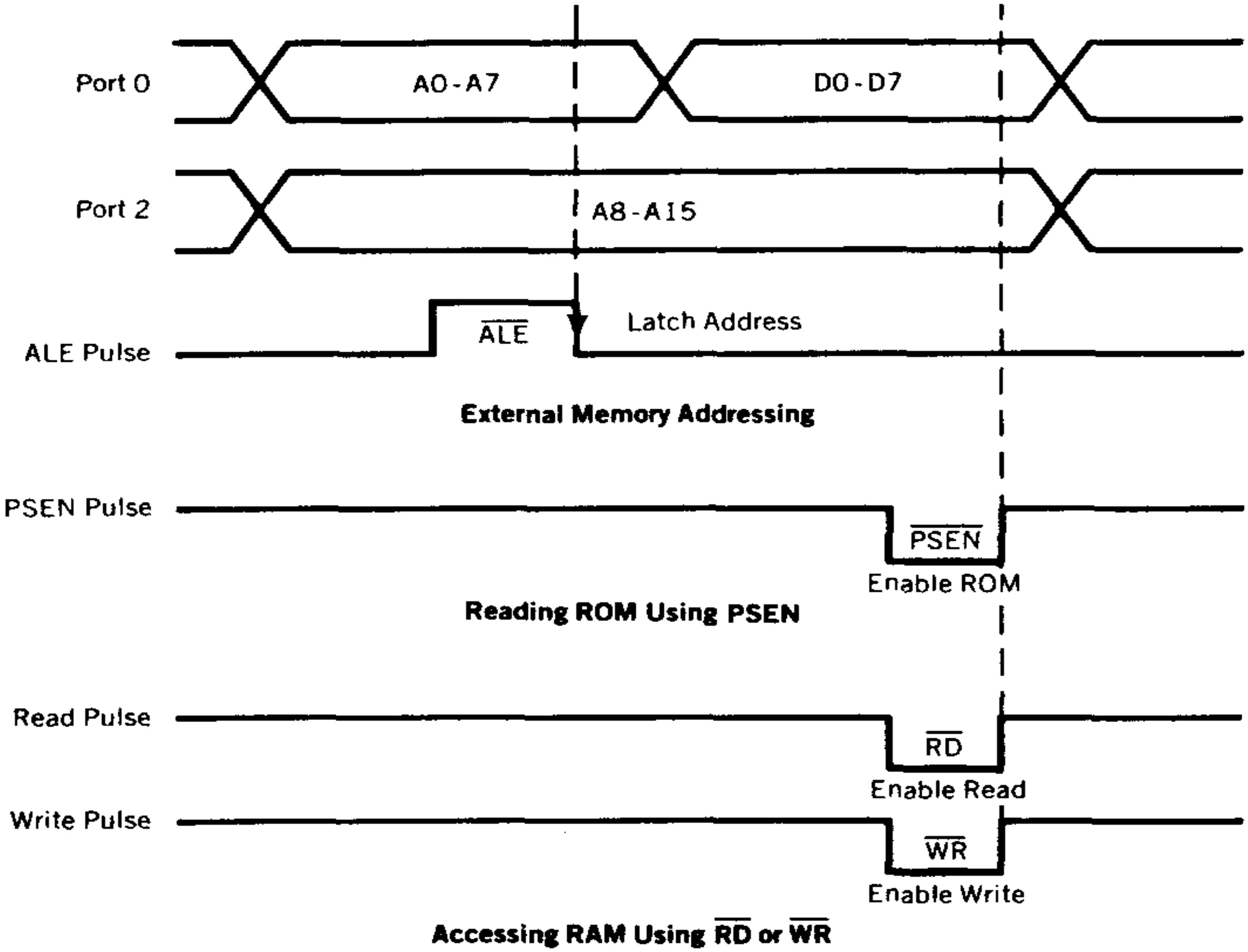
0x00

Lower 128 RAM (Direct and Indirect Addressing)

External Memory

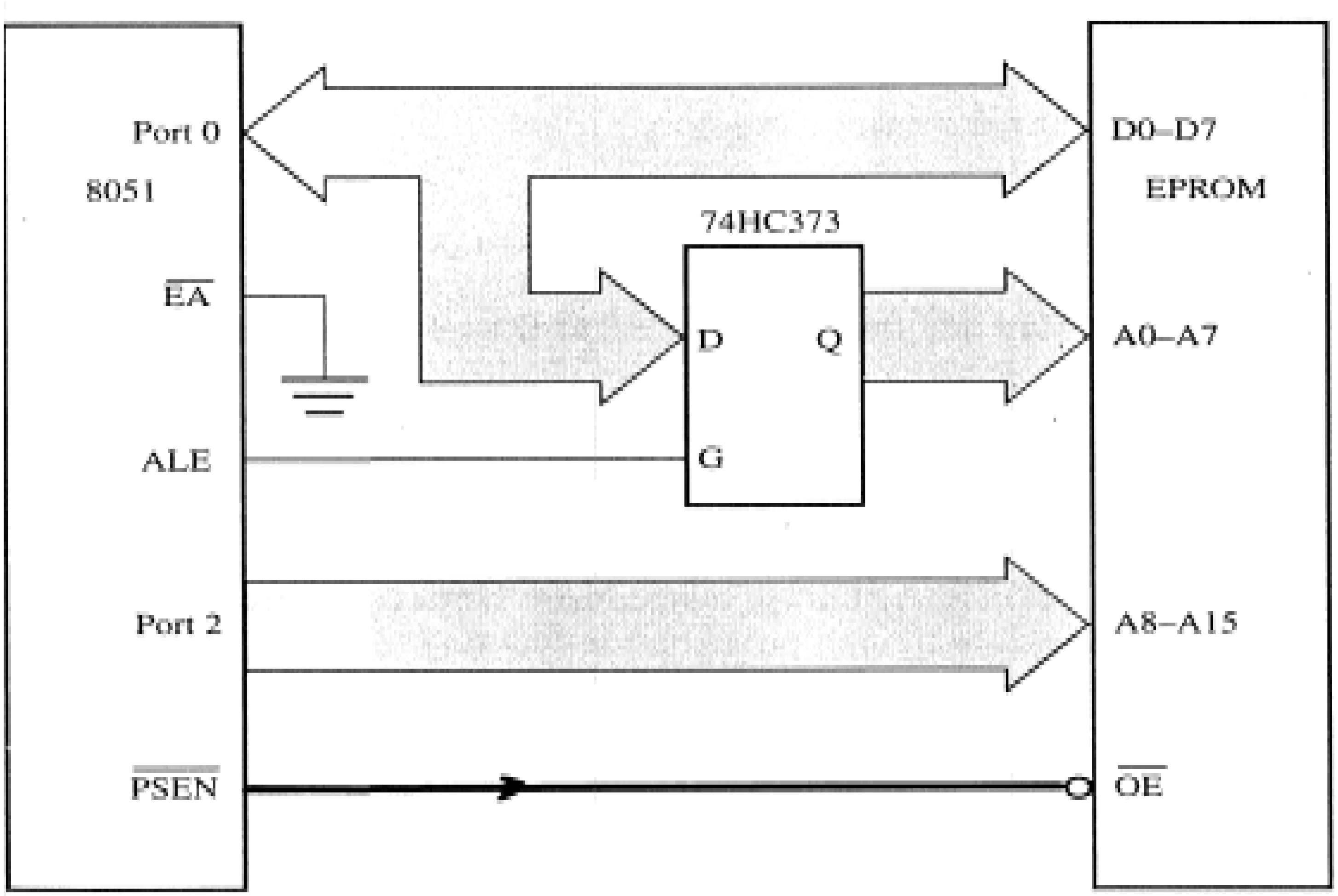


External Memory Timing



External Program Memory

Accessing external code memory



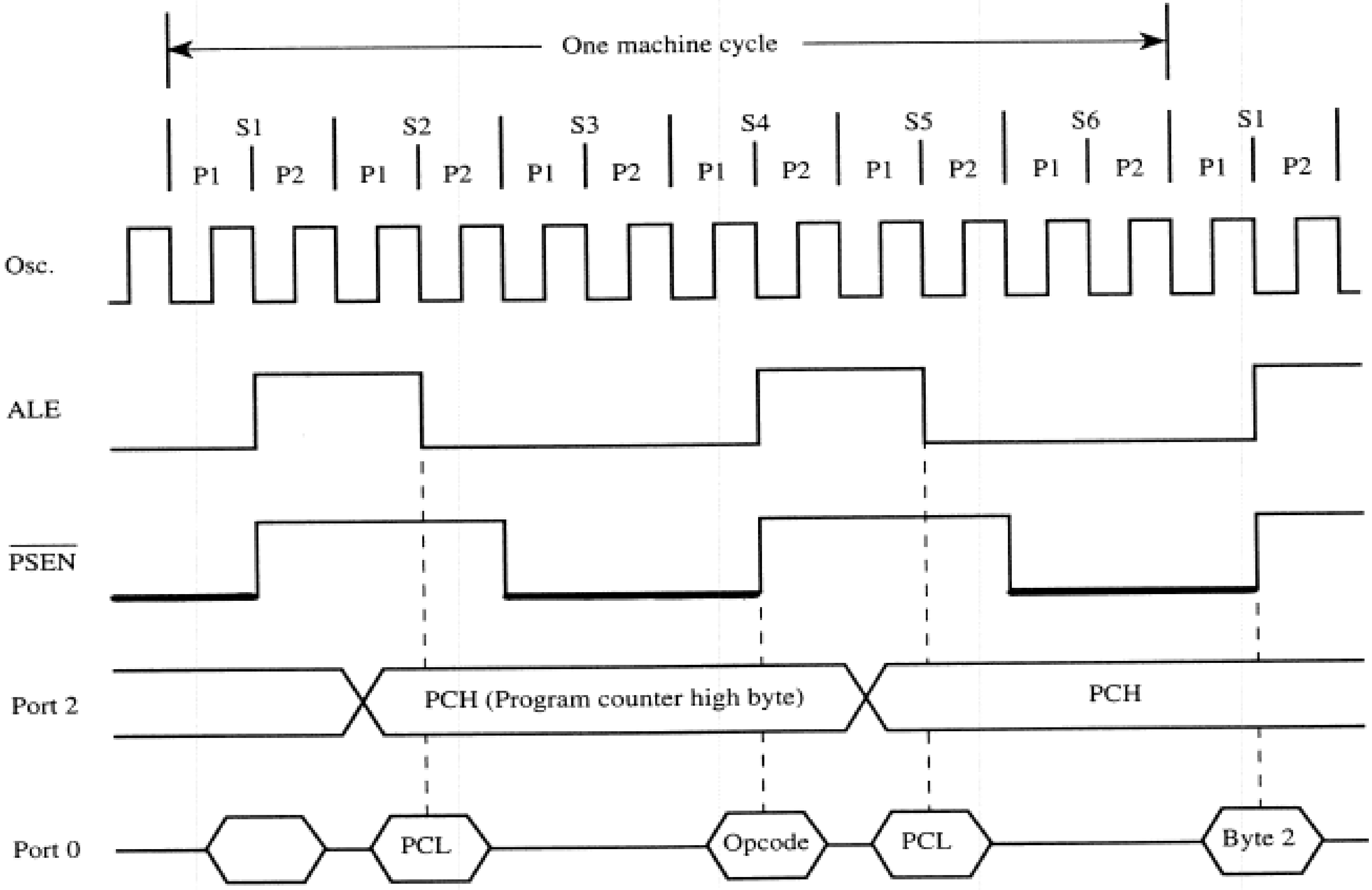
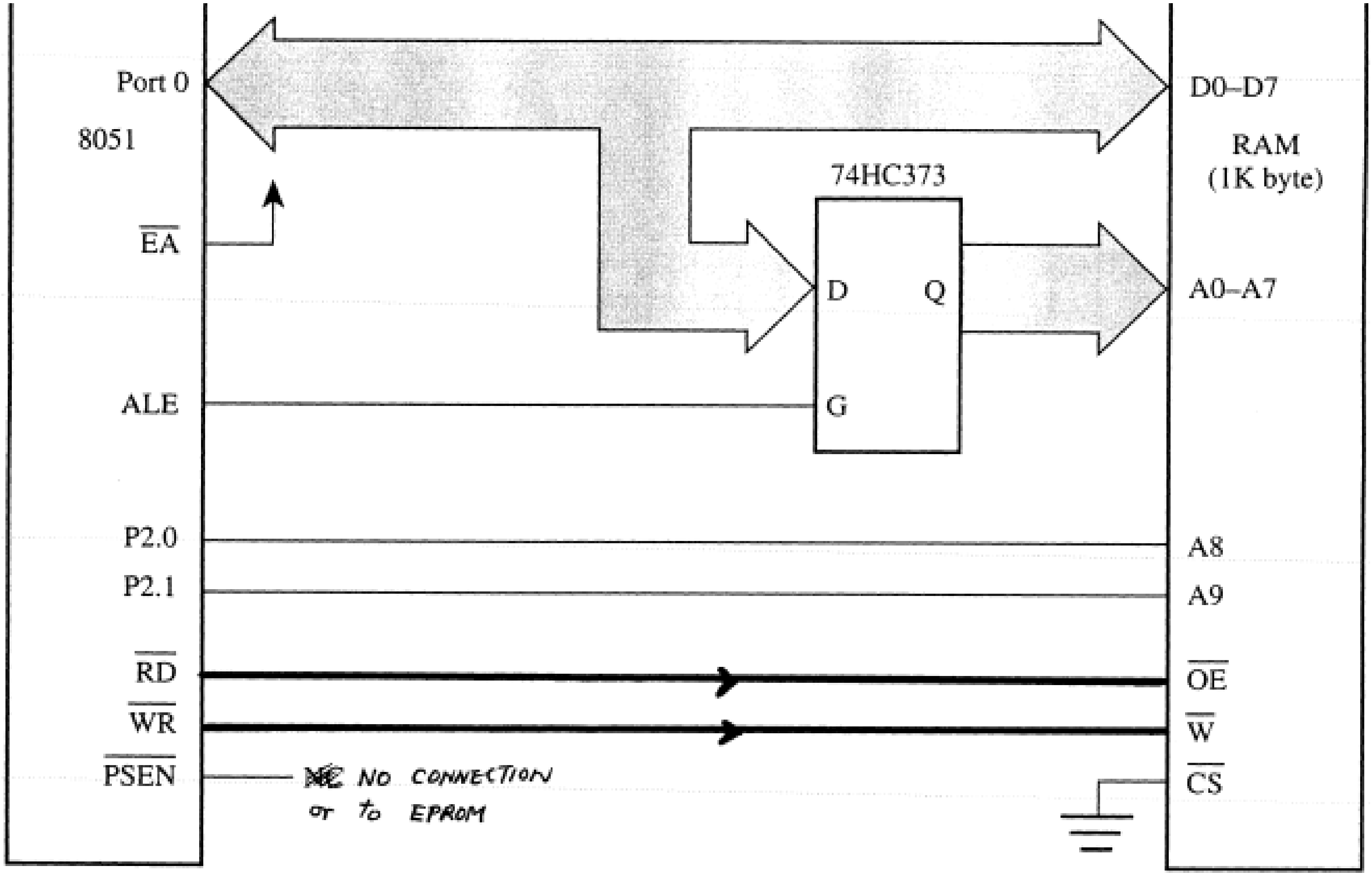


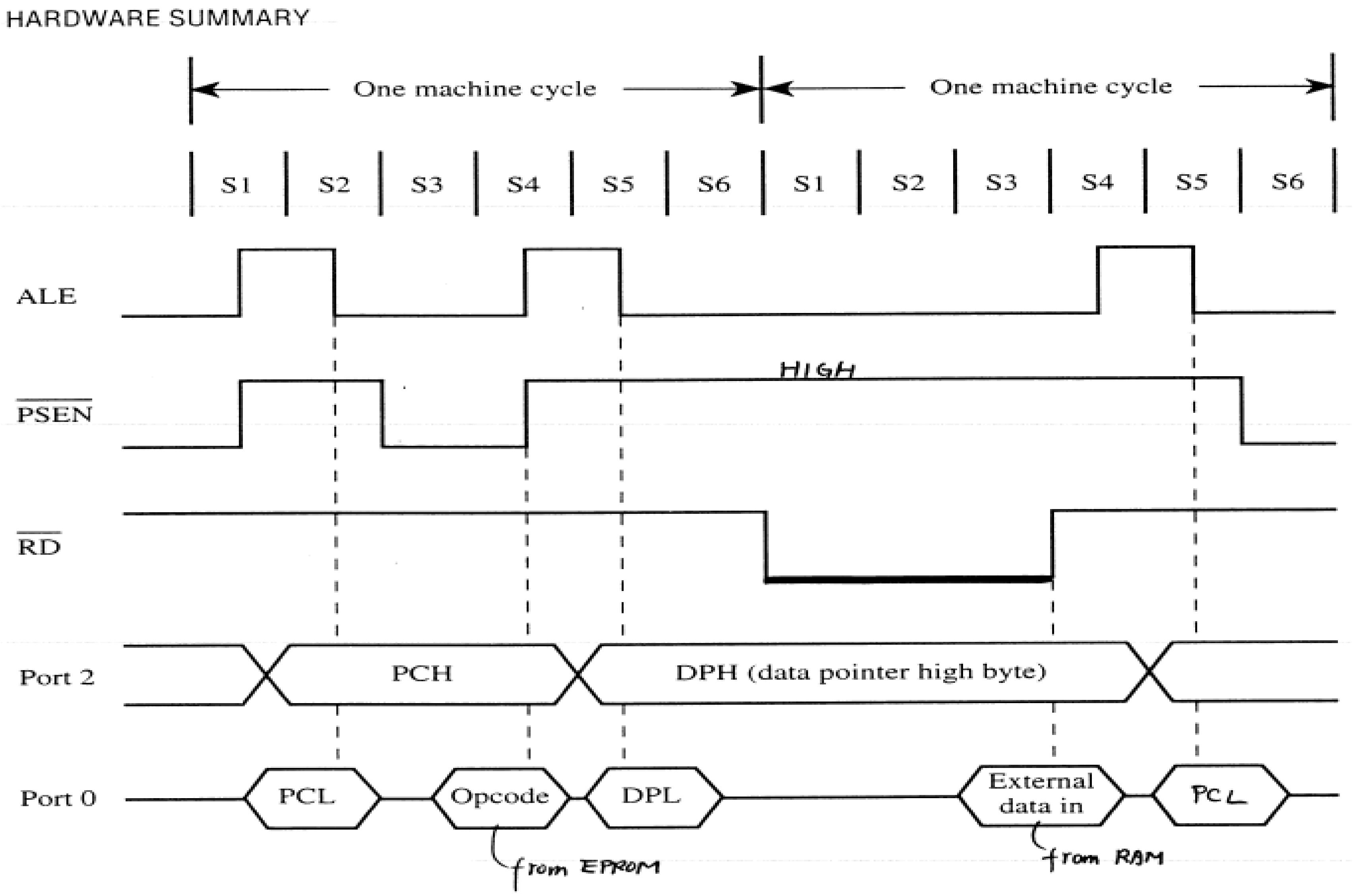
FIGURE 2–9
Read timing for external code memory

External Data Memory

Interface to 1K RAM



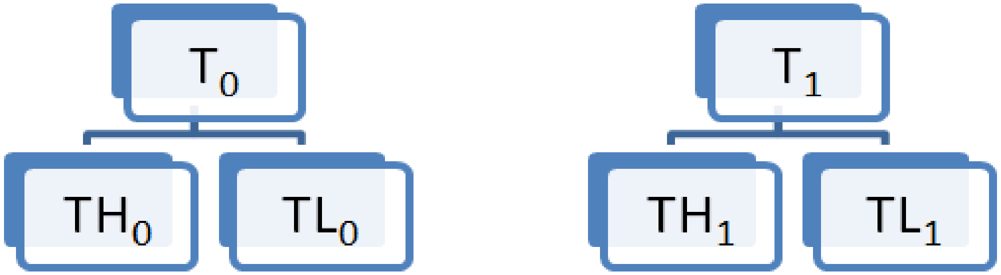
Timing for MOVX instruction



8051 timer

Timer/Counter

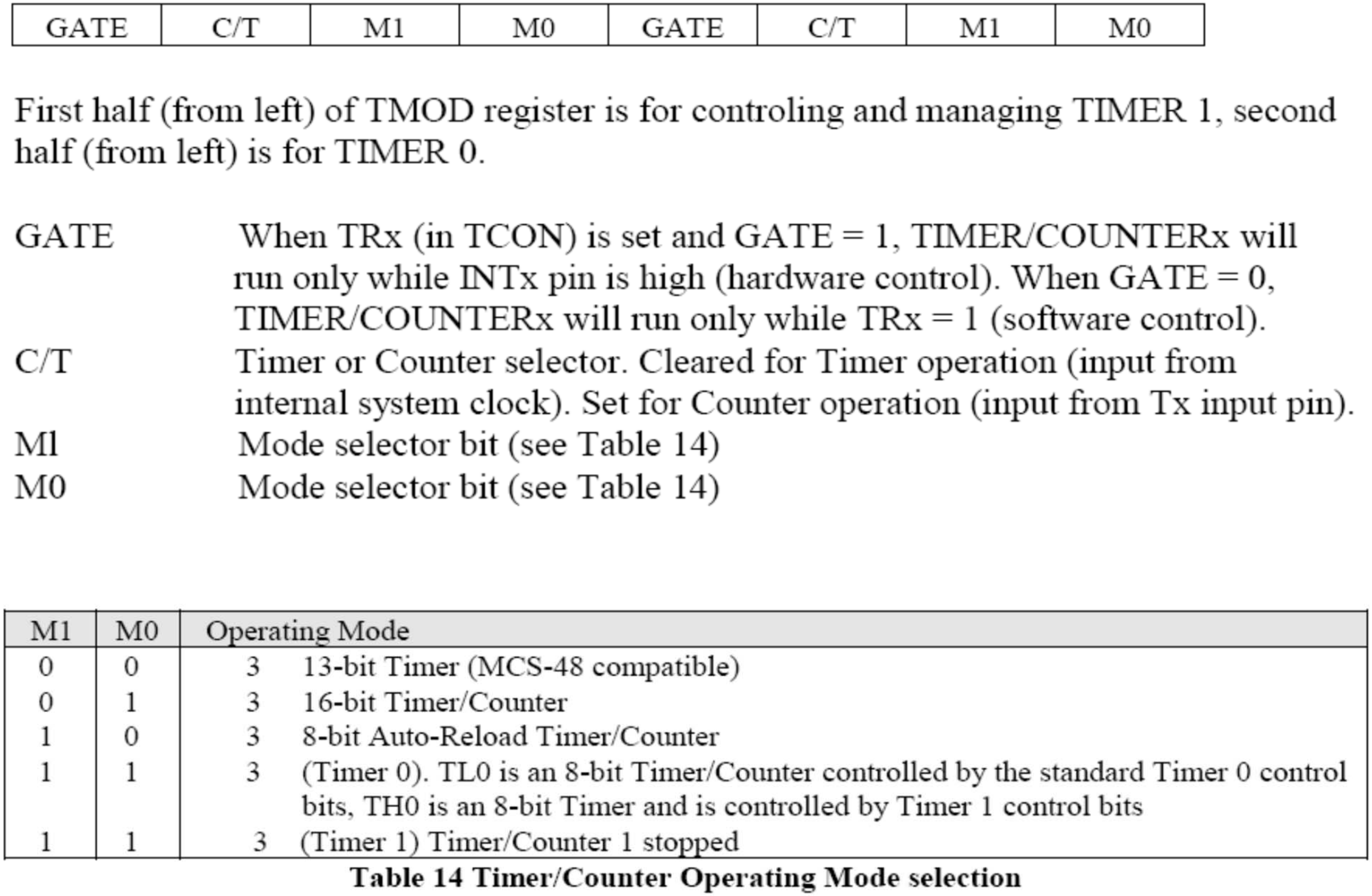
- Timers → It counts the clock pulses and to generate the time delay.
- Counters → To count the events or external pulses.
- 8051 has two (16) bit up counters/timers (T0 & T1).



- Types of Timer Registers:
  - TMOD
  - TCON
  - TL0, TH0, TL1, TH1

- Timer Interrupts:
  - TF0 → Timer overflows for Timer0.
  - TF1 → Timer overflows for Timer1.

TMOD Register



GATE	C/T	M1	M0	GATE	C/T	M1	M0

First half (from left) of TMOD register is for controling and managing TIMER 1, second half (from left) is for TIMER 0.

GATE When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).

C/T Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).

Ml Mode selector bit (see Table 14)

M0 Mode selector bit (see Table 14)

M1	M0	Operating Mode	
0	0	3	13-bit Timer (MCS-48 compatible)
0	1	3	16-bit Timer/Counter
1	0	3	8-bit Auto-Reload Timer/Counter
1	1	3	(Timer 0). TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8-bit Timer and is controlled by Timer 1 control bits
1	1	3	(Timer 1) Timer/Counter 1 stopped

Table 14 Timer/Counter Operating Mode selection

TCON Register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TFl TCON.7 Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.

TR1 TCON.6 Timer 1 run control bit. Set/clared by software to turn Timer/Counter 1 ON/OFF.

TF0 TCON.5 Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.

TR0 TCON.4 Timer 0 run control bit. Set/cleared by software to turn Timer/Counter ON/OFF.

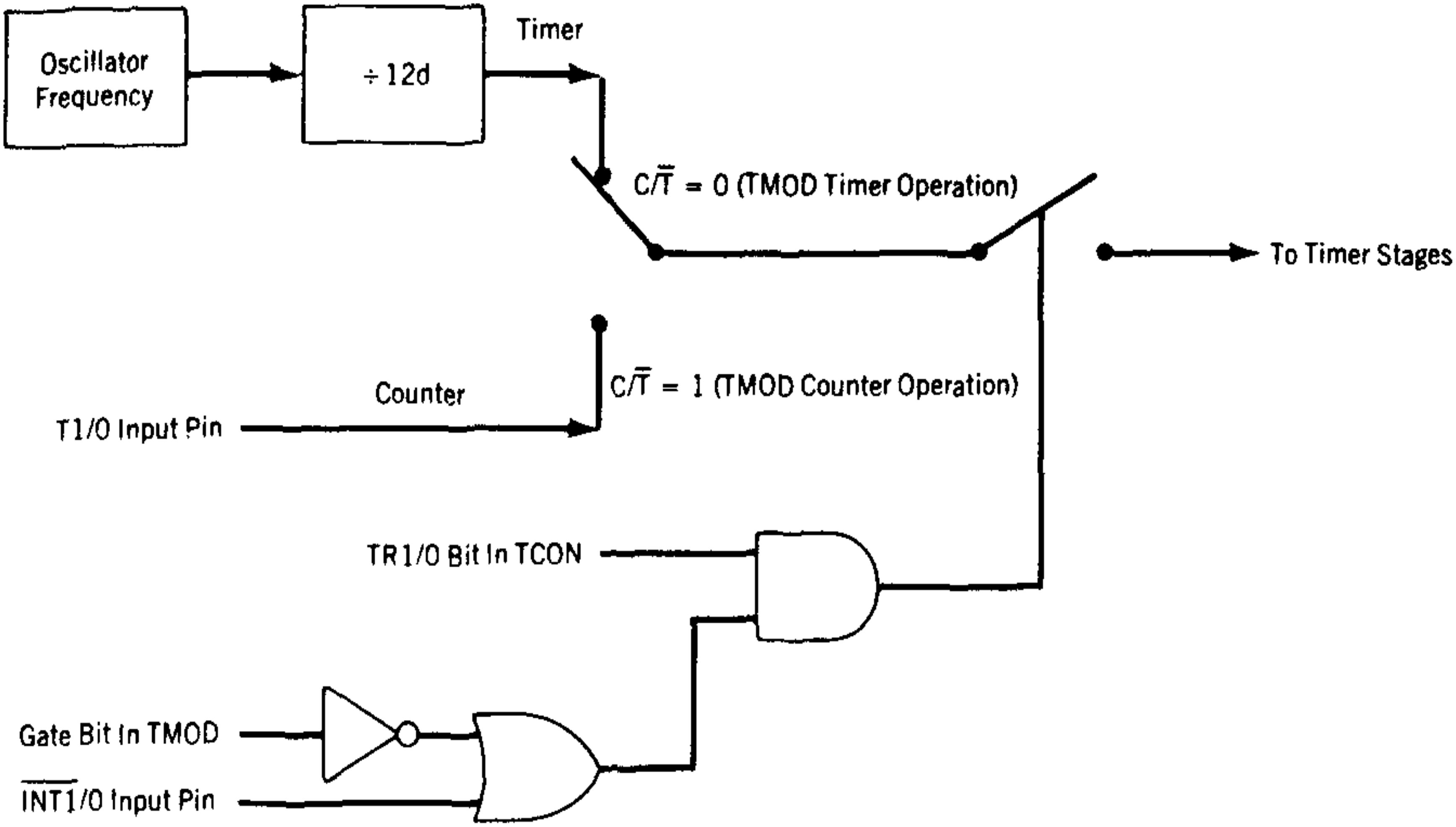
IE1 TCON.3 External Interrupt 1 edge flag. Set by hardware when External Interrupt edge is detected. Cleared by hardware when interrupt is processed.

IT1 TCON.2 Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

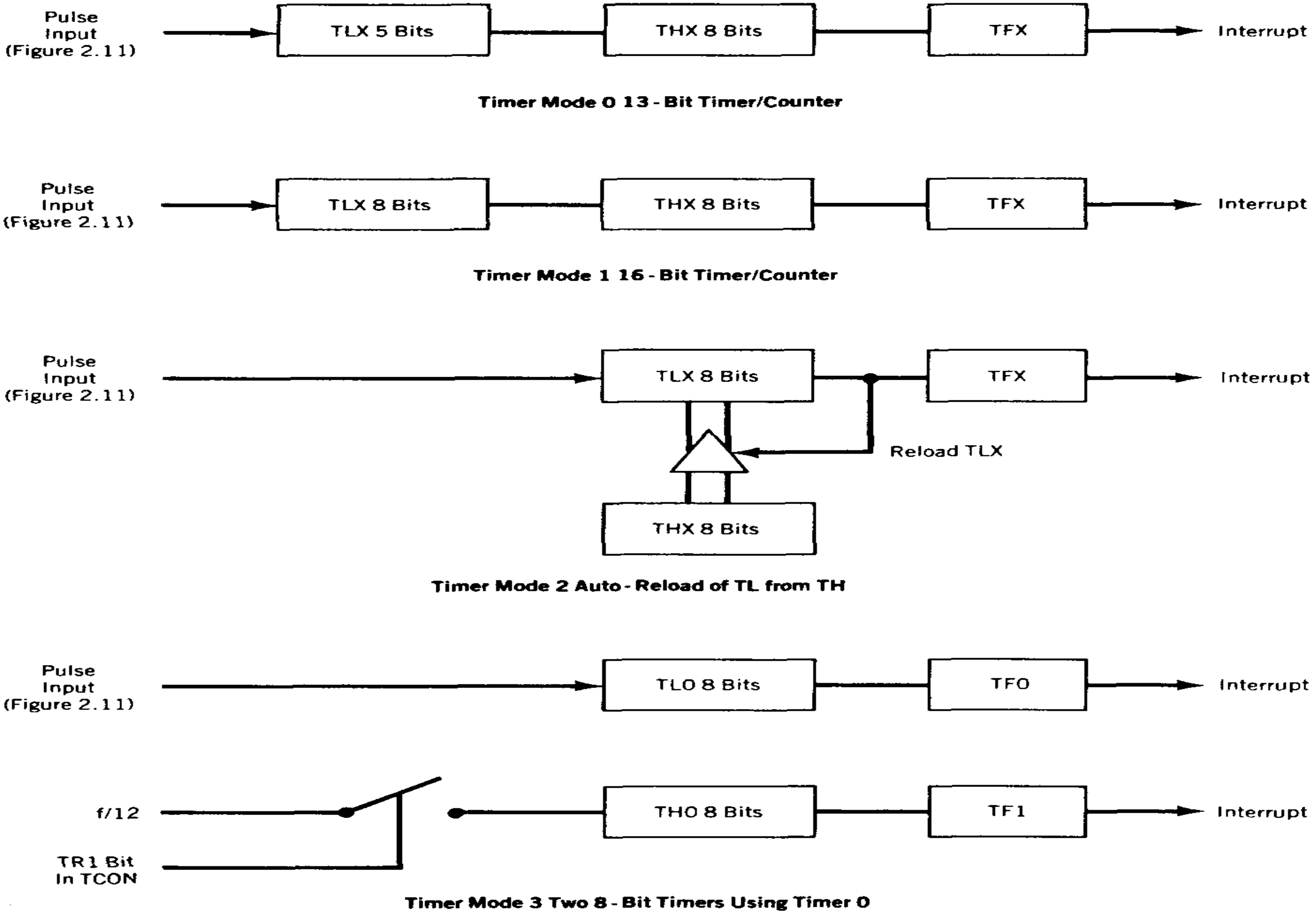
IE0 TCON.1 External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.

IT0 TCON.0 Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

Timer/Counter Control Logic


Oscillator Frequency
÷12d
Timer
C/$\overline{T}$ = 0 (TMOD Timer Operation)
To Timer Stages
Counter
T1/0 Input Pin
C/$\overline{T}$ = 1 (TMOD Counter Operation)
TR1/0 Bit In TCON
Gate Bit In TMOD
$\overline{INT1}$/0 Input Pin

Modes of Timer/Counter



Timer Mode 0 13 - Bit Timer/Counter

Timer Mode 1 16 - Bit Timer/Counter

Timer Mode 2 Auto - Reload of TL from TH

Timer Mode 3 Two 8 - Bit Timers Using Timer 0

Time Delay Calculations:

The 8051 microcontroller works with 11.0592 MHz frequency.
Frequency 11.0592MHz=12 pules
1 clock pulse =11.0592MHz/12
F =0.921 MHz

Time delay=1/F
T=1/0.92MHz
T=1.080506 us (for ‘1’ cycle)

1000us=1MS
1000ms=1sec

Generate 500us time delay

500us/1.080806us
461pulses
P=65535-461
P=65074
65074 conveted by hexa decimal =FE32

TH1=0xFE;
TL1=0x32;

Programming Steps:

- Programming steps for delay function
- Load Tmod register value i.e. TMOD = 0x00 for Timer0/1 mode0 (13-bit timer mode).
- Load calculated THx value i.e. here TH0 = 0xE3.
- Load calculated TLx value i.e. here TL0 = 0x14.
- Start timer by setting TRx bit. i.e. here TR0 = 1.
- Poll TFx flag till it does not get set.
- Stop timer by clearing TRx bit. i.e. here TR0 = 0.
- Clear timer flag TFx bit i.e. here TF0 = 0.
- Repeat from step 1 to 7 for delay again.

Programming :

```
/*  8051_Timer_Mode0 */
#include <reg51.h> /* Include x51 header file */
sbit test = P1^0; /* set test pin 0 of port1  */
void timer_delay() /* Timer0 delay function */
{ TH0 = 0xE3; /* Load 8-bit in TH0 (here Timer0 used) */ TL0 = 0x14;
/* Load 5-bit in TL0 */
TR0 = 1; /* Start timer0 */
while(TF0 == 0); /* Wait until timer0 flag set */
TR0 = 0; /* Stop timer0 */
TF0 = 0; /* Clear timer0 flag */
}
void main()
{
TMOD = 0x00; /* Timer0/1 mode0 (13-bit timer mode) */
while(1)
{ test = ~test; /* Toggle test pin */
timer_delay(); /* Call timer0 delay */
}
}
```

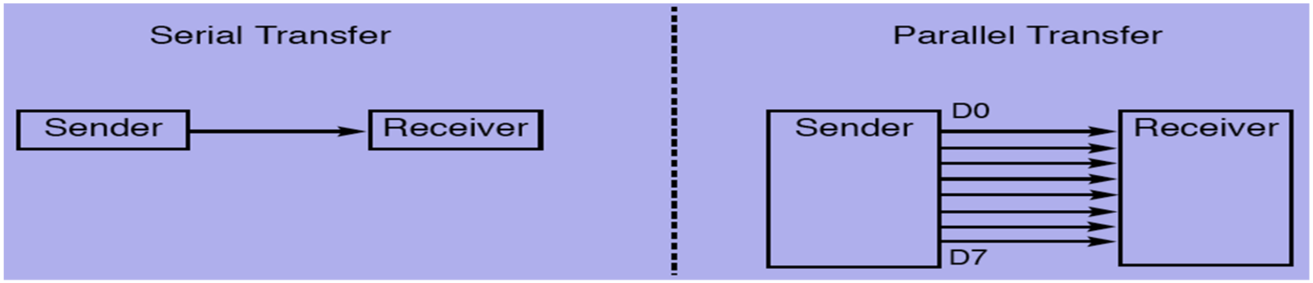
Write a program using Timer 0 to create a 10 kHz square wave on P1.0.

```
      ORG   0000H
      MOV TMOD, #02H       ;auto-reload mode
      MOV TH0, # -50       ;T= 1/10kHz=100µs
      SETB  TR0            ;start timer
LOOP: JNB  TF0, LOOP       ;wait for overflow
      CLR    TF0           ;clear overflow flag
      CPL   P1.0           ;toggle port bit
      SJMP LOOP            ;repeat
      END
```

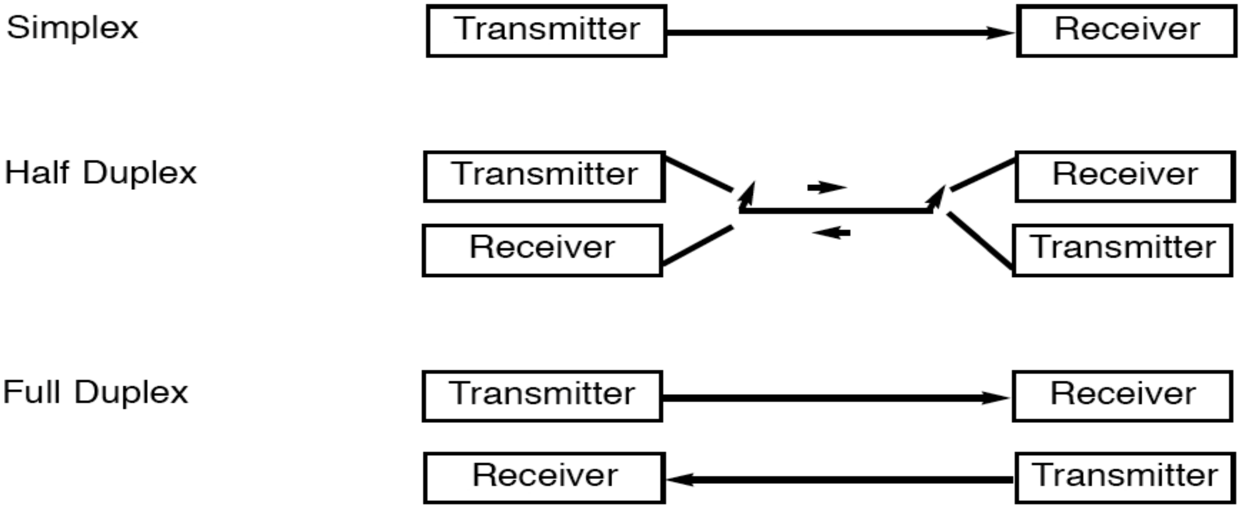
8051 Serial

Serial Basics

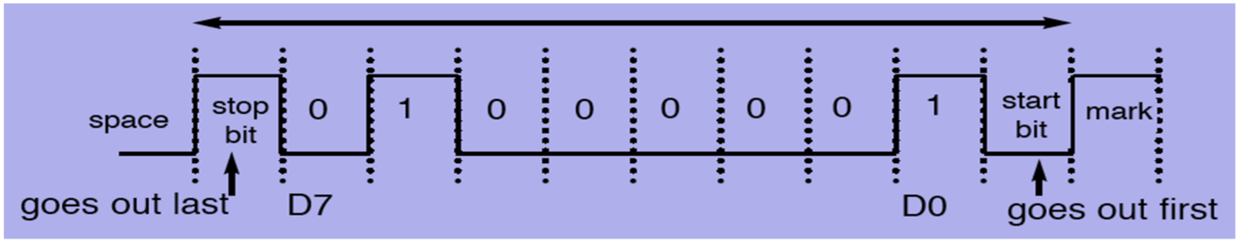
Serial Vs Parallel



Simplex Vs Duplex



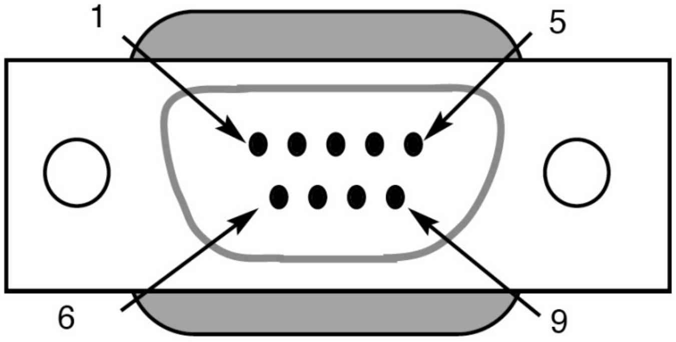
Sync Vs Async

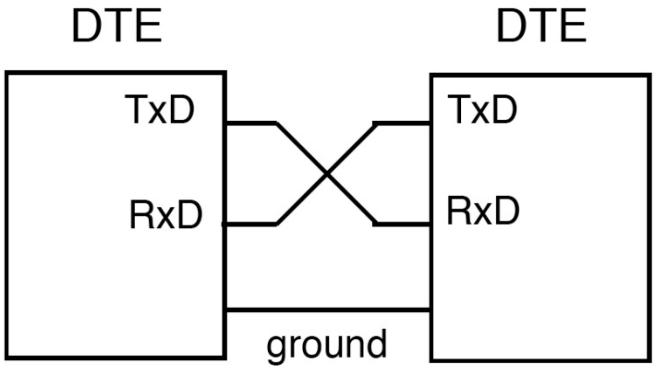


Serial Basics

RS232 Connector DB-9

Pin	Description
1	Data carrier detect ($\overline{\text{DCD}}$)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready ($\overline{\text{DSR}}$)
7	Request to send ($\overline{\text{RTS}}$)
8	Clear to send ($\overline{\text{CTS}}$)
9	Ring indicator (RI)





Null Modem Connection

In RS232 → 1 bit is represented by -3 to -25 V.
0 bit is +3 to +25 V.

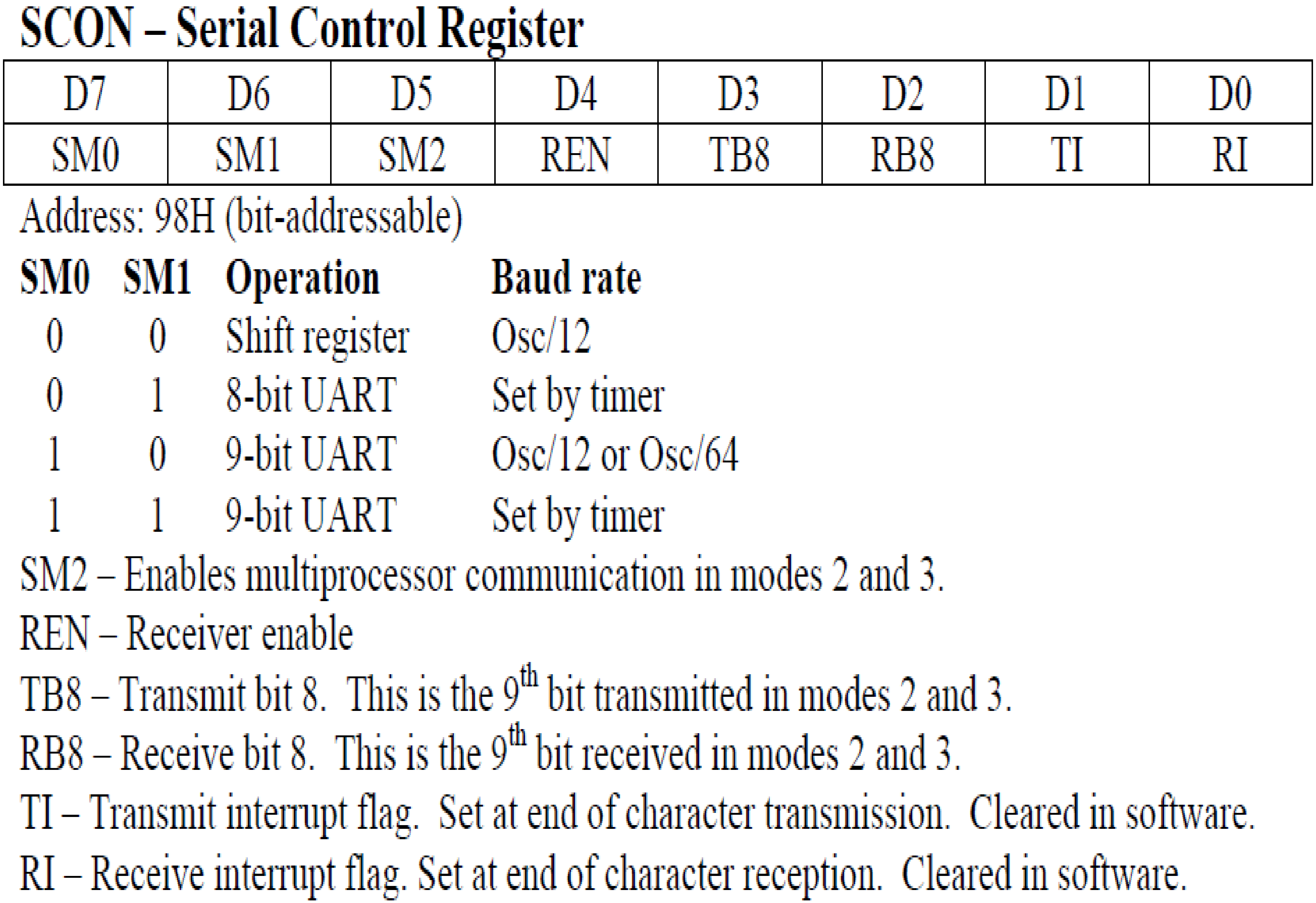
MAX232 → converts from RS232 voltage levels to TTL voltage levels.

SBUF - Serial Buffer

- **SBUF Register**
  - A byte of data to be transferred via the TxD line must be placed in the SBUF register.
  - SBUF holds the byte of data when it is received by the RxD line.
- Can be accessed like any other register

```
MOV SBUF,#'D'      ;load SBUF=44H, ASCII for 'D'
MOV SBUF,A         ;copy accumulator into SBUF
MOV A,SBUF         ;copy SBUF into accumulator
```

- When a byte is written, it is framed with the start and stop bits and transferred serially via the TxD pin.
- When the bits are received serially via RxD, it is deframe by eliminating the stop and start bits, making a byte out of the data received, and then placing it in the SBUF.



SCON – Serial Control Register

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Address: 98H (bit-addressable)

SM0	**SM1**	**Operation**	**Baud rate**
0	0	Shift register	Osc/12
0	1	8-bit UART	Set by timer
1	0	9-bit UART	Osc/12 or Osc/64
1	1	9-bit UART	Set by timer

SM2 – Enables multiprocessor communication in modes 2 and 3.

REN – Receiver enable

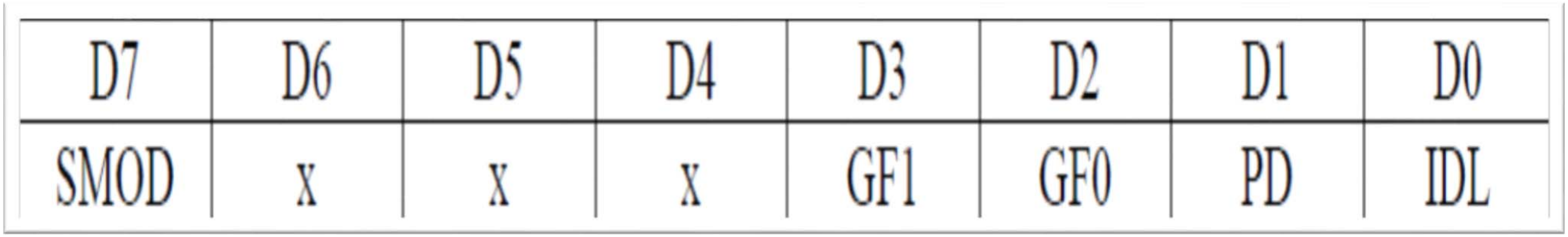
TB8 – Transmit bit 8. This is the 9th bit transmitted in modes 2 and 3.

RB8 – Receive bit 8. This is the 9th bit received in modes 2 and 3.

TI – Transmit interrupt flag. Set at end of character transmission. Cleared in software.

RI – Receive interrupt flag. Set at end of character reception. Cleared in software.

PCON – Power Control Register



D7	D6	D5	D4	D3	D2	D1	D0
SMOD	X	X	X	GF1	GF0	PD	IDL

- ✓ Address: 87H (not bit addressable)
- ✓ **SMOD – Serial mode bit** used to determine the baud rate with Timer 1. (1=double the baud rate)
- ✓ **GF1 and GF0 are General purpose flags** not implemented on the standard device
- ✓ **PD is the power down bit.** Not implemented on the standard device
- ✓ **IDL activate the idle mode to save power**. Not implemented on the standard device

TI / RI bits

- TI (transmit interrupt)
  - when 8051 finishes the transfer of the 8-bit character, it raises the TI flag to indicate that it is ready to transfer another byte.

- RI (receive interrupt)
  - when the 8051 receives data serially via RxD, it places the byte in the SBUF register.
  - then raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost.

Baud Rate

- **Baud rate in the 8051**
  - baud rate in the 8051 is programmable
  - done with the help of Timer 1
  - relationship between the crystal frequency and the baud rate in the 8051
  - 8051 divides the crystal frequency by 12 to get the machine cycle frequency
  - XTAL = 11.0592 MHz, the machine cycle frequency is 921.6 kHz
  - 8051's UART divides the machine cycle frequency of 921.6 kHz by 32 once more before it is used by Timer 1 to set the baud rate
  - Timer 1 must be programmed in mode 2, that is 8-bit, auto-reload

- Machine cycle frequency

  = 11.0592 MHz / 12 = 921.6 kHz

- Timer 1 frequency provided by 8051 UART

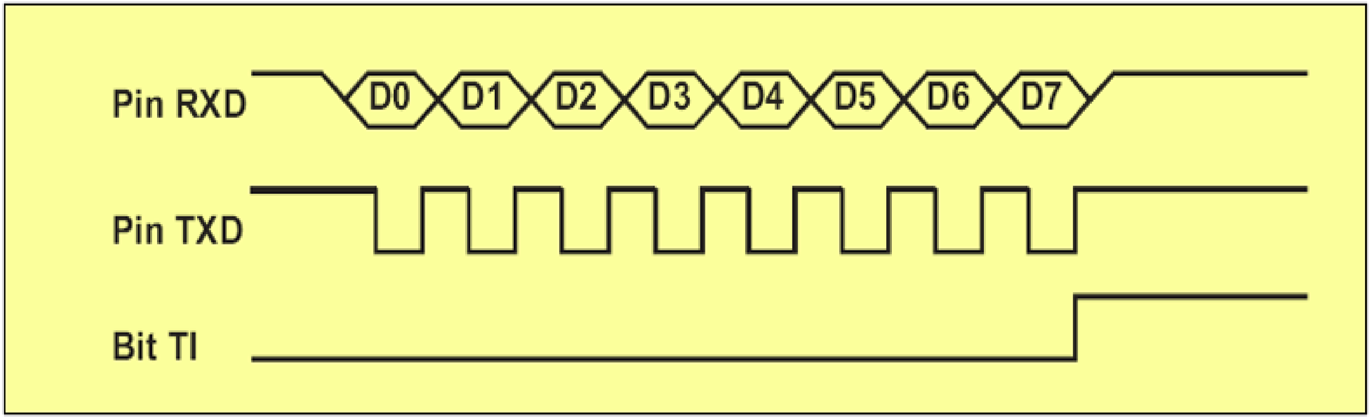
  = 921.6 kHz / 32 = 28,800 Hz

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

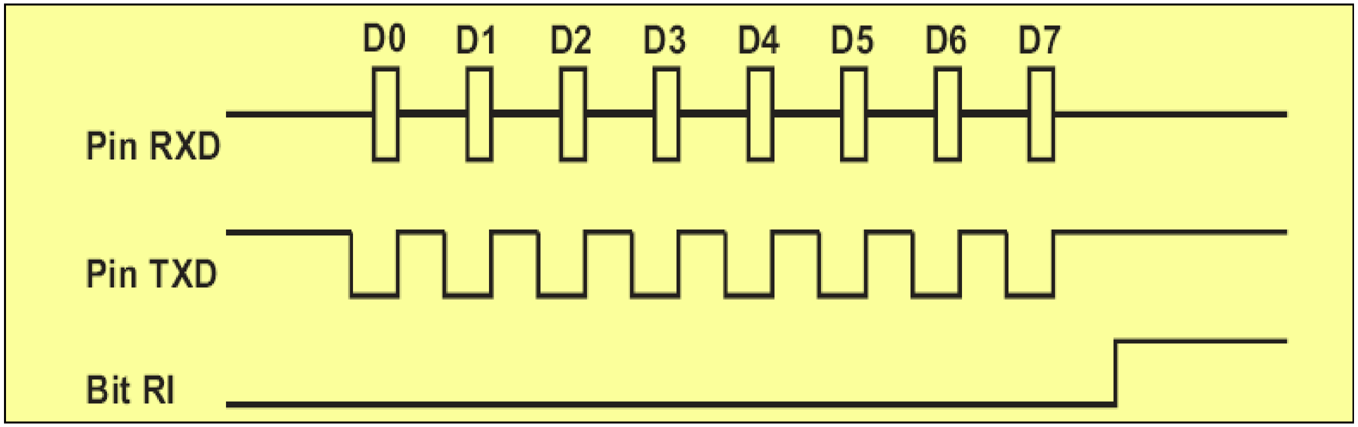
(b) 28,800 / 12 = 2400 where -12 = F4 (hex)

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

MODE:0 SHIFT REGISTER MODE

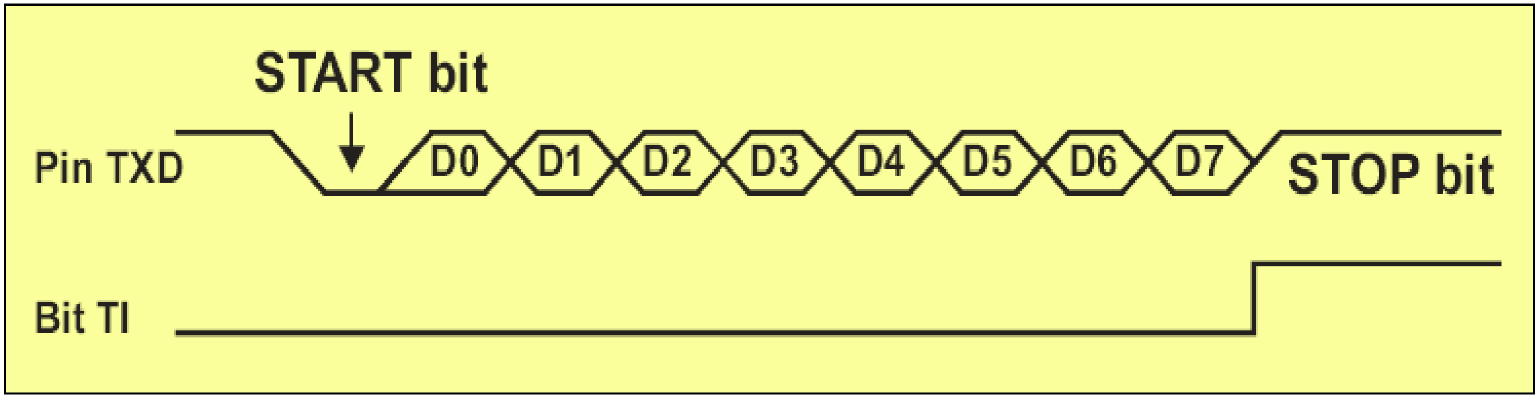


TRANSMISSION

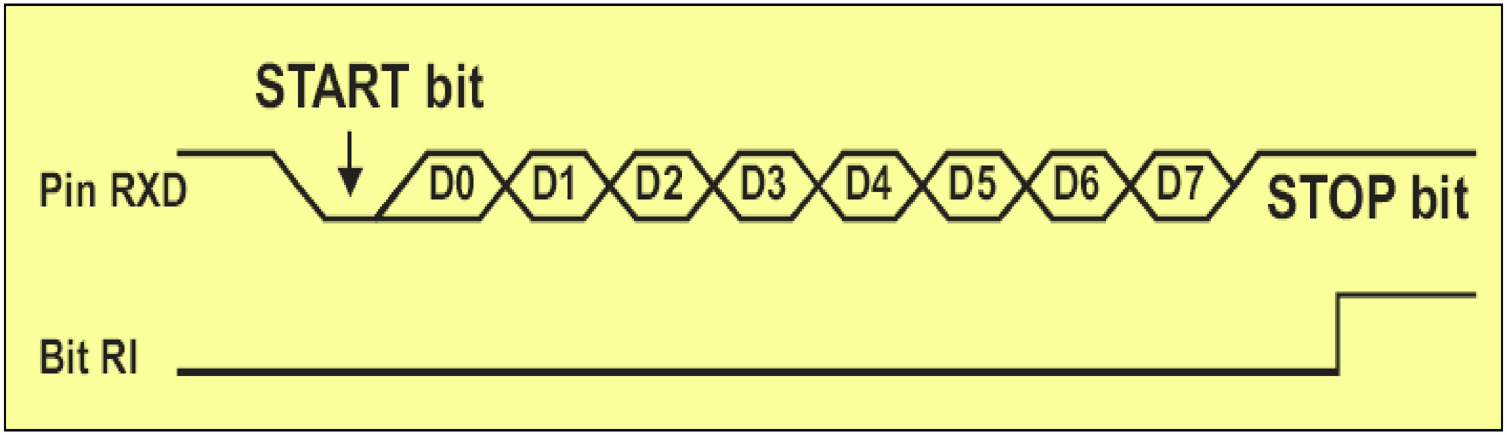


RECEPTION

MODE 1: 8 BIT UART MODE

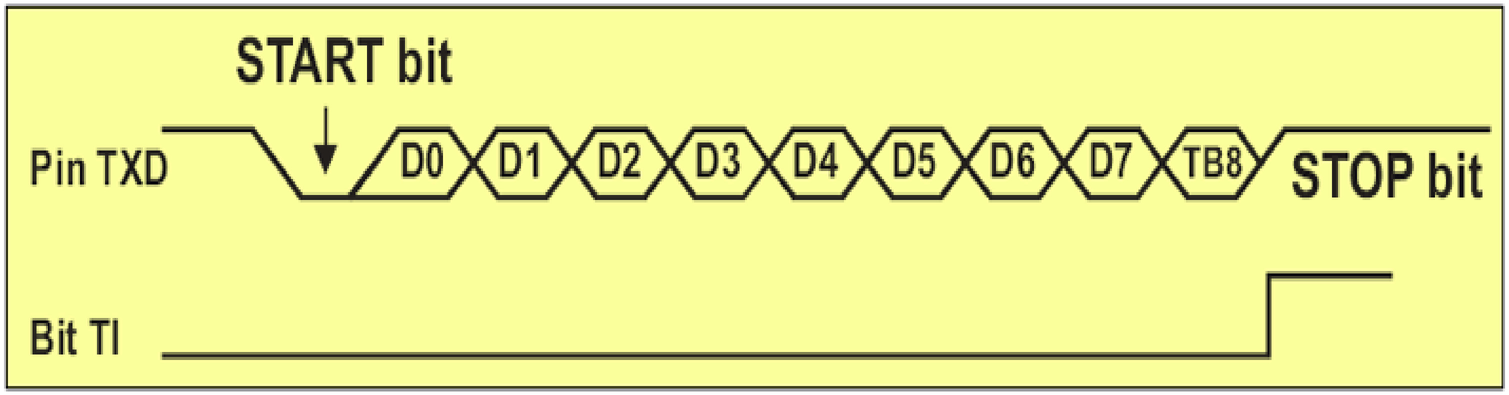


TRANSMIT

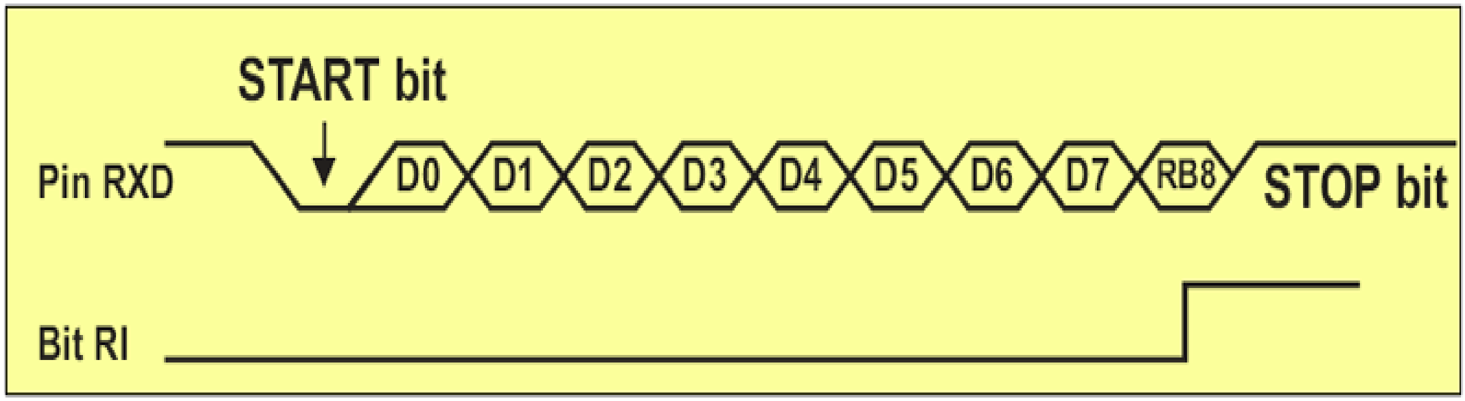


RECIEVE

MODE 2: 9 BIT UART MODE



TRANSMIT



RECIEVE

Dr. V.Vijayaraghavan, Assistant Professor-ECE, VFSTR, Guntur

Serial Programming - Steps

- **Program to transfer data serially**

1. TMOD register is loaded with the value 20H
2. TH1 is loaded with value to set the baud rate
3. SCON register is loaded with the value 50H
4. TR1 is set to 1 to start Timer1
5. TI is cleared by the "CLR TI" instruction
6. transmit character byte is written into the SBUF register
7. TI flag bit is monitored to see if the character has been transferred completely
8. To transfer the next character, go to Step 5.

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

```
        MOV   TMOD,#10H    ;timer 1, mode 1
        MOV   TH1,#-6      ;4800 baud rate
        MOV   SCON,#50H    ;8-bit, 1 stop, REN enabled
        SETB  TR1          ;start timer 1
AGN:    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  AGN          ;keep sending A
```

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

```
        MOV    TMOD,#10H   ;timer 1, mode 1
        MOV    TH1,#-3     ;9600 baud
        MOV    SCON,#50H   ;8-bit, 1 stop bit, REN enabled
        SETB   TR1         ;start timer 1
AGN:    MOV    A,#"Y"      ;transfer "Y"
        ACALL XMIT
        MOV    A,#"E"      ;transfer "E"
        ACALL  XMIT
        MOV    A,#"S"      ;transfer "S"
        ACALL  XMIT
        SJMP   AGN         ;keep doing it

;serial data transfer subroutine
XMIT:   MOV    SBUF,A;load SBUF
HERE:   JNB    TI,HERE     ;wait for last bit to transfer
        CLR    TI          ;get ready for next byte
        RET
```

Serial Programming - Steps

- **Program to receive data serially**

1. TMOD register is loaded with the value 20H
2. TH1 is loaded with value set the baud rate
3. SCON register is loaded with the value 50H
4. TR1 is set to 1 to start Timer 1
5. RI is cleared with the "CLR RI" instruction
6. RI flag bit is monitored to see if an entire character has been received yet
7. RI=1 SBUF has the byte, its contents are moved into a safe place
8. To receive the next character, go to Step 5

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.

```
        MOV   TMOD,#10H    ;timer 1, mode 1
        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
```

8051 Interrupts

Interrupt Sources

- 8051 has 5 sources of interrupts
  - External Interrupt 0 (INT0’)
  - External Interrupt 1 (INT1’)
  - Timer 0 overflow (TF0)
  - Timer 1 overflow (TF1)
  - Serial Port events (TI/RI)

Interrupt Process

If interrupt event occurs AND interrupt flag for that event is enabled, AND interrupts are enabled, then:

1. Current PC is pushed on stack.
2. Program execution continues at the interrupt vector address for that interrupt.
3. When a RETI instruction is encountered, the PC is popped from the stack and program execution resumes where it left off.

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 default priority order.
- Priority can also be set to “high” or “low”.

Interrupt Vectors

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

High

Priority

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

Low

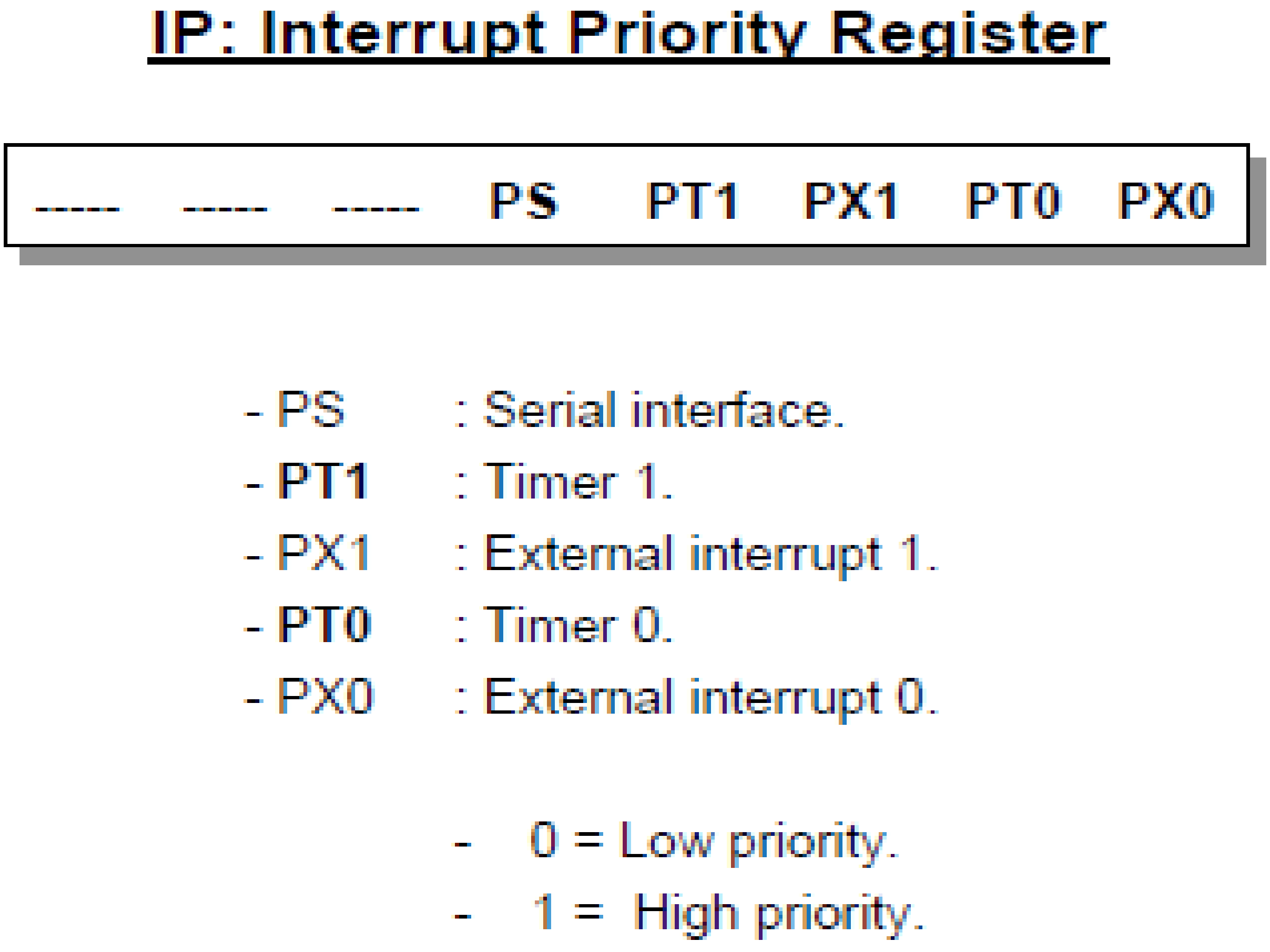
Note: that there are only 8 memory locations between vectors.

IE : Interrupt Enable Register

EA	----	----	ES	ET1	EX1	ET0	EX0

- EA : Global interrupt enable.
- ES : Serial interface.
- ET1 : Timer 1.
- EX1 : External interrupt 1.
- ET0 : Timer 0.
- EX0 : External interrupt 0.

- 0 = Disabled.
- 1 = Enabled.



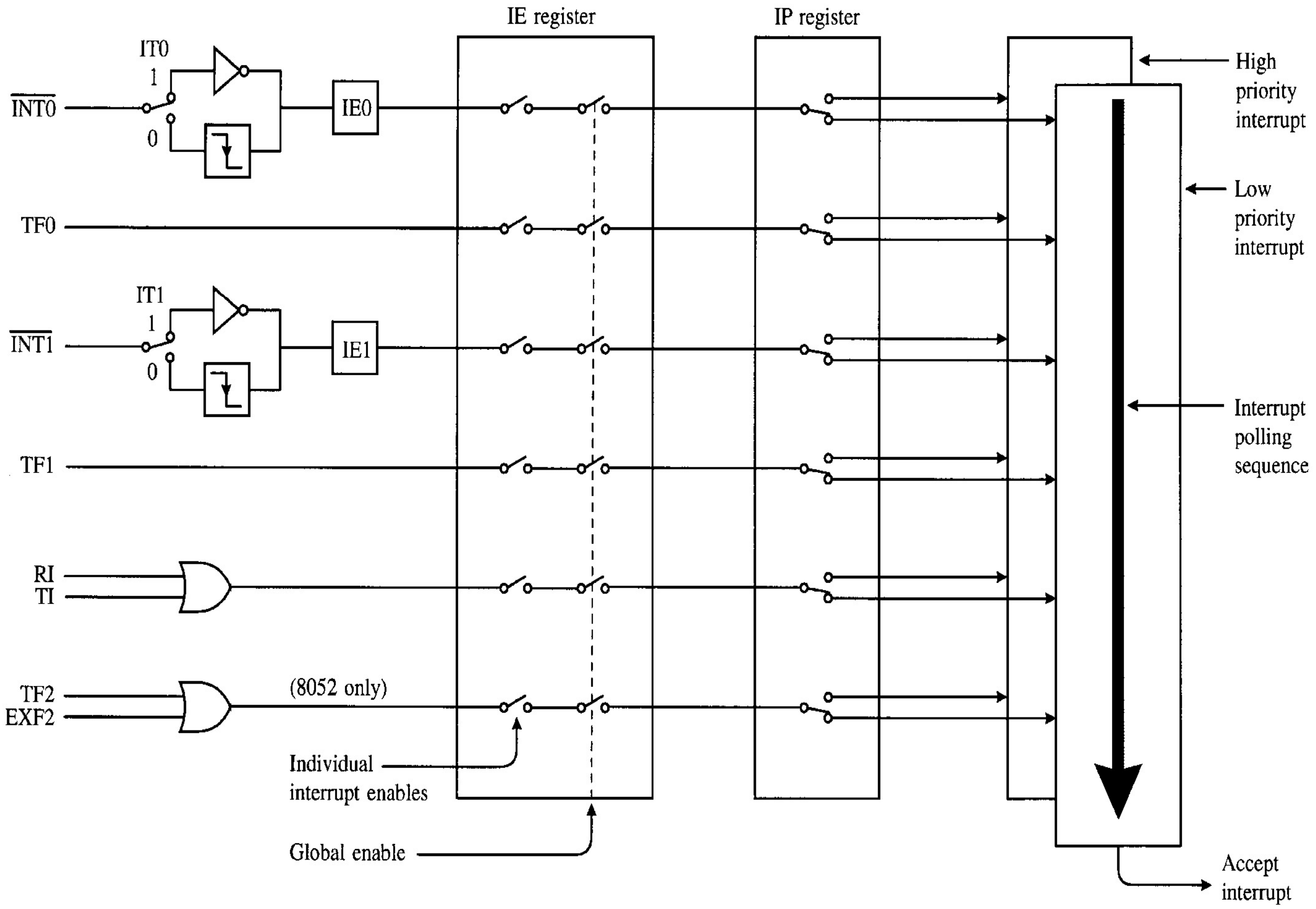
IP: Interrupt Priority Register

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

- PS : Serial interface.
- **PT1** : Timer 1.
- PX1 : External interrupt 1.
- **PT0** : Timer 0.
- PX0 : External interrupt 0.

- 0 = Low priority.
- 1 = High priority.

Interrupt Structure



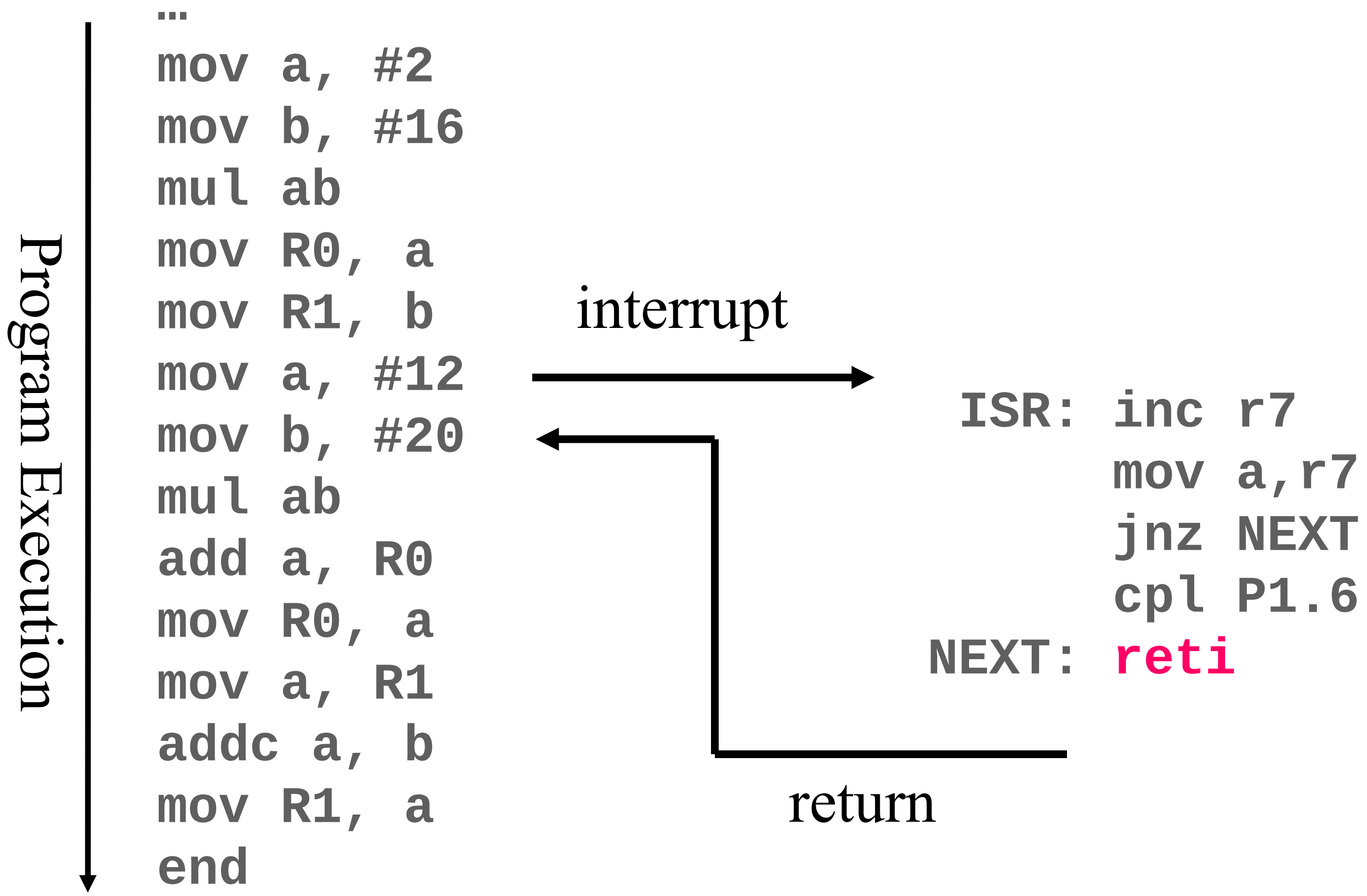
- Example of ISRs

```
ORG     0000H   ; Reset
LJMP    main    ; 3-byte instruction
…..
LJMP T0_ISR     ; Calls Timer0 ISR

ORG     0030H   ; Main Program
Main:
# initialized the timer, serial port , …..
   …….
HERE:   SJMP    HERE

ORG     000BH   ; TF0 Vector Address
T0_ISR:
….                          ; or LJMP T0_ISR
RETI
```

Interrupts



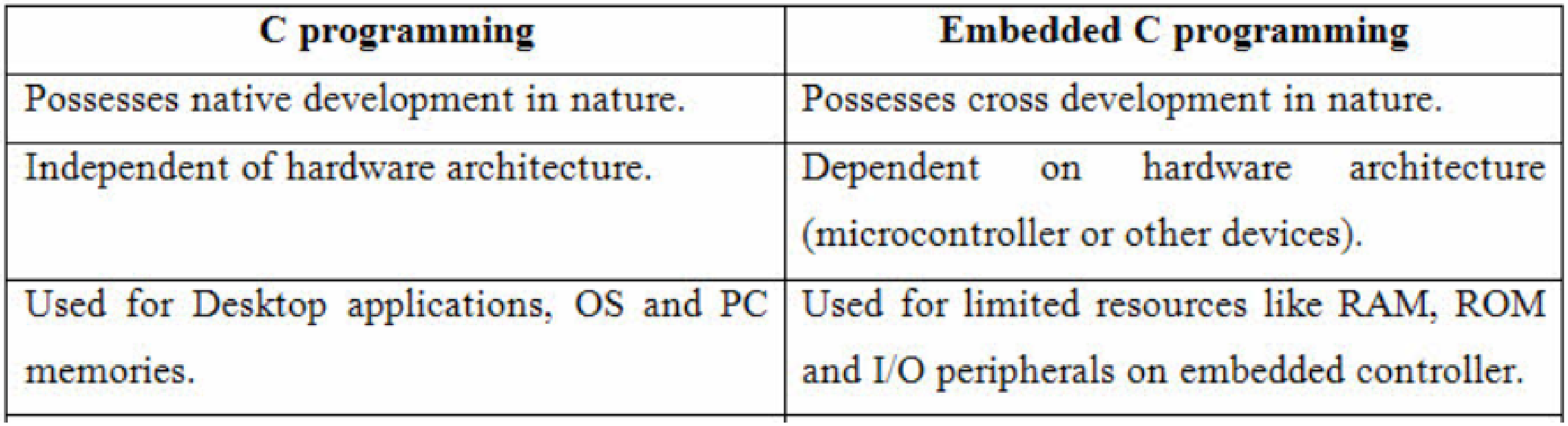
```
…
mov a, #2
mov b, #16
mul ab
mov R0, a
mov R1, b
mov a, #12
mov b, #20
mul ab
add a, R0
mov R0, a
mov a, R1
addc a, b
mov R1, a
end
```

```
  ISR: inc r7
       mov a,r7
       jnz NEXT
       cpl P1.6
 NEXT: reti
```

EMBEDDED C

- Embedded C is most popular programming language in software field for developing electronic gadgets.
- Embedded C programming plays a key role in performing specific function by the processor. In day-to-day life we used many electronic devices such as mobile phone, washing machine, digital camera, etc. These all device working is based on microcontroller that are programmed by embedded C.

Differences between C and Embedded C



C programming	Embedded C programming
Possesses native development in nature.	Possesses cross development in nature.
Independent of hardware architecture.	Dependent on hardware architecture (microcontroller or other devices).
Used for Desktop applications, OS and PC memories.	Used for limited resources like RAM, ROM and I/O peripherals on embedded controller.

In embedded system programming C code is preferred over other language. Due to the following reasons:

*Easy to understand *High Reliability *Portability *Scalability

Benefits of Assembly Language Program (ALP)

The following are the advantages of programming in assembly language:

- There is efficient use of the memory. Machine codes generated from the ALP are compact. The ALP codes need smaller memory than in machine codes for the same function in memory after the compilation of a high level language program. [C is a high level language.]
- Features of processor instruction set are kept in full view when program is done using ALP.
- Only few assembly instructions are needed for driving the devices. Driving a device means configuring of device by writing control or command words, and writing or reading the bytes at registers or ports of device.
- System operations are speeded by the use of ALP.
- There is full understanding of the process that is taking place at each instance of program execution. Time critical codes are therefore programmed in ALP.
- The code size and code execution speed of the program are known.

Use of High-level Language Like ‘C’

A high-level language like ‘C’ is used instead of assembly language programming (ALP). This is because of the following reasons:

- Enables development of a lengthy and complex program in a short time.
- Facilitates data-type declarations. It enables the use of variables and arrays simpler. Enables definitions of objects and data structures.
- Performs the data-type check during compilation. It results in less programming errors.
- Ease in design of program-flow control structures. For example, while do, if then else, for and case statements are used.
- Coding in C is not specific to a particular microcontroller family or version. The compiler must automatically take into account the instruction set of the microcontroller family or version after the appropriate preprocessor directives.
- Uses functions from the library. A library is a set of files and each file has the frequently required functions in the computations and programs.
- Implements modular programming concepts: A sources file containing a collection of functions, which are related in a program, is called a module. A module can be built by including functions from several source files.
- In-line assembly codes can be permitted. It means inserting the assembly language codes in- between. A direct hardware control is thus also feasible in C.
- Software re-usability is feasible.

Addressing Modes

How to represents the operands in a instruction is called "addressing mode".

- In summary, the addressing modes are as follows, with an example of each:

• **Register Addressing**	MOV A,R2
• **Immediate Addressing**	MOV A,#20h
• **Direct Addressing**	MOV A,30h
• **Indirect Addressing**	MOV A,@R0
• **External Indirect**	MOVX A,@DPTR
• **Indexed**	MOVC A,@A+DPTR
• **Implied**	RL A

Assembler Directives

- ORG: Used to indicate the beginning of the address.
- EQU: Used to define a constant without occupying a memory location.
  - e.g. Count EQU 25
- END: Indicates the end of the source file.

Instruction Sets

Based on the operations we divide the instructions in to some groups are called “Instruction set".

- In summary, the Instruction sets are as follows, with an example of each:

Data Transfer Instructions	MOV A,R2
Arithmetic Instructions	ADD A,#20h
Logical Instructions	ANL A,30h
Branch Instructions	SJMP <addr>
Bit Level Instructions	SETB <bit>

REFERENCE BOOKS

VIGNAN'S Foundation for Science, Technology & Research (Deemed to be UNIVERSITY) -Estd. u/s 3 of UGC Act 1956

- Raj Kamal, "Embedded Systems Architecture, Programming and Design", 2nd Edition, McGraw Hill, 2009.
- Lyla B. Das, "Embedded Systems: An Integrated Approach", 1st Edition, Pearson, 2012.
- Kenneth J. Ayala, "The 8051 microcontroller : Architecture, Programming", 3rd Edition, Thomson Learning, 2007.
- David E. Simon, "An Embedded Software Primer", 1st Edition, Pearson, 2008.
- Marilyn Wolf, "Computers as Components - Principles of Embedded Computing System Design", 3rd Edition, Morgan Kaufmann Publisher (Elsevier), 2012.
- Jean J. Labrosse, "Embedded System Building Blocks: Complete and Ready-to-Use Modules in C", 2nd Edition, CRC Press, 1999.
- Frank Vahid and Tony Givargis, "Embedded System Design: A Unified Hardware/Software Introduction", 3rd Edition, John Wiley & Sons, 2006.
- K.V.K.K. Prasad, "Embedded Real-Time Systems: Concepts, Design & Programming", Dream Tech Press, 2005.

THANK YOU…