EMBEDDED SYSTEMS

CHAPTER – 9

Microcontrollers in Embedded Systems

9. Microcontrollers in Embedded Systems [3 Hrs.]

9.1 Intel 8051 microcontroller family, its architecture and instruction sets

9.2 Programming in Assembly Language

9.3 A simple interfacing example with 7 segment display

Microcontroller is a Highly integrated chip that contains a CPU, scratchpad RAM, special and general purpose register arrays and integrated peripherals.

Among 8 bit microcontroller 8051 family is most popular: cost effective & versatile device offering extensive support in the embedded application domain.

1st popular model: 8031AH built by Intel in 1977

Microprocessor

- A Si chip representing a CPU, which is capable of performing arithmetic as well as logical operations according to a pre defined set of instructions.
- CPU is stand-alone, RAM, ROM, I/O, timer are separate

Microcontroller

- Is a highly integrated chip that contains a CPU, scratchpad RAM, special and general purpose register arrays, on chip ROM/FLASH memory for program storage, timer & interrupt control units and dedicated I/O ports.
- CPU, RAM, ROM, I/O and timer are all on a single chip

Microprocessor	Microcontroller
• is a dependent unit. It requires the combination of other chip like timers, program and data memory chips, interrupt controllers, etc. for functioning	• Is a self-contained unit & it doesn't require external interrupt controller, timer, UART (Universal Asynchronous Receiver Transmitter), etc. for its functioning
• Most of the time general purpose in design and operation	• Mostly application – oriented or domain-specific
• Targeted for high end market where performance is important	• Targeted for embedded market where performance is not so critical (At present this demarcation is invalid)

Microprocessor	Microcontroller
• doesn't contain a built in I/O port. The I/O port functionality needs to be implemented with the help of external programmable peripheral interface chips like 8255	• Most of the processors contain multiple built-in I/O ports which can be operated as a single 8 or 16 or 32 bit port or as individual port pins
• Limited power saving options compared to microcontrollers	• Includes lots of power saving features

Factors to be considered in selecting a controller

Feature set: interface, port requirement by the application, nos. of timers & counters, built-in ADC/DAC hardware, required performance

Speed of Operation: nos. of clocks required per instruction cycle and the max operating clock freq. supported by the processor greatly affected the speed of operation of the controller.

Code Memory Space: if the target processor/ controller application is written in C or any other high level language, does the controller support sufficient code memory space to hold the compiled hex code?

Factors to be considered in selecting a controller

Data Memory Space: does the controller support sufficient internal data memory (on chip RAM) to hold run time variables and data structures?

Development Support: Does the controller manufacture provide cost-effective development tools, sample product, development pains, support third party development tools, and technical support?

Availability: referred to as lead time. Lead time is the time elapsed between the purchase order approval and supply of the product.

Factors to be considered in selecting a controller

Power Consumption: of the controller should be minimal. It is a crucial factor since higher power requirement leads to bulky power supply designs. The high power dissipation also demands for cooling fans and it will make the overall system messy and expensive. Controllers should support idle and power down modes of operation to reduce consumption.

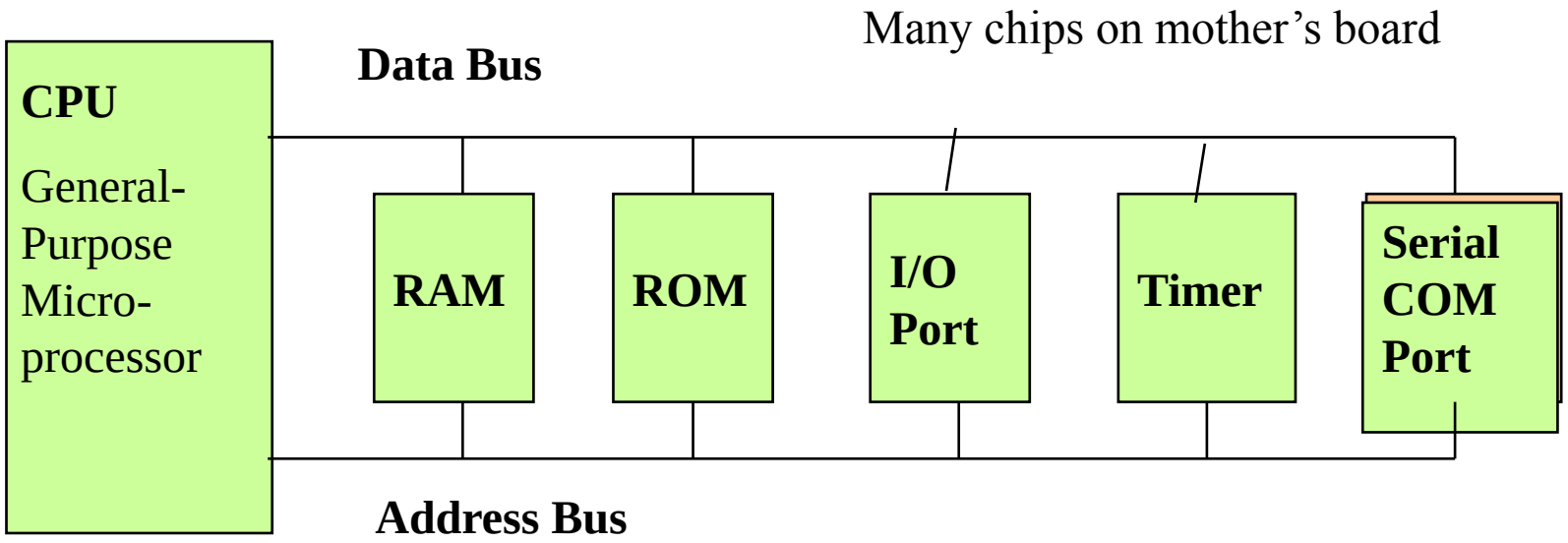
Cost: Last but not least, the cost should be within the reachable limit of the end user and the target user should not be high tech. Remember the ultimate aim of a product is to gain marginal benefit.

Why 8051 Microcontroller?

- Is very versatile microcontroller featuring powerful Boolean processor which supports bit manipulation instructions for real time industrial control applications.
- architecture supports 6 interrupts (2 external interrupts, 2 timer interrupts & 2 serial interrupts)
- 2 16bit timers/counters
- 32 I/O lines & a programmable full duplex serial interface.
- is the way it handles interrupts.
- low cost
- flash microcontroller (AT89C51) from Atmel is available in the market

Microprocessors:
General-purpose microprocessor

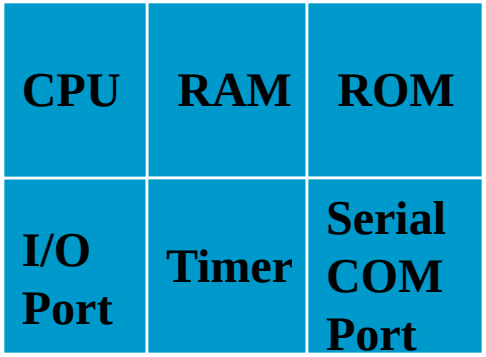
- CPU for Computers
- No RAM, ROM, I/O on CPU chip itself
- Example：Intel’s x86, Motorola’s 680x0



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.



A single chip

Microcontroller

Companies Producing 8051

Some Companies Producing a Member of the 8051 Family

Company	Web Site
Intel	www.intel.com/design/mcs51
Atmel	www.atmel.com
Philips/Signetics	www.semiconductors.philips.com
Siemens	www.sci.siemens.com
Dallas Semiconductor	www.dalsemi.com

Common Microcontrollers

- Atmel
- ARM
- Intel
  - 8-bit
    - 8XC42
    - MCS48
    - **MCS51** 
    - 8xC251
  - 16-bit
    - MCS96
    - MXS296
- National Semiconductor
  - COP8
- Microchip
  - 12-bit instruction PIC
  - 14-bit instruction PIC
    - PIC16F84
  - 16-bit instruction PIC
- NEC

- Motorola
  - 8-bit
    - 68HC05
    - 68HC08
    - 68HC11
  - 16-bit
    - 68HC12
    - 68HC16
  - 32-bit
    - 683xx
- Texas Instruments
  - TMS370
  - MSP430
- Zilog
  - Z8
  - Z86E02

8051 Family

Comparison of 8051 Family Members

Feature	8051	8052	8031
ROM (on chip 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

Intel 8051

- 8051 introduced by Intel in late 1970s
- Now produced by many companies in many variations
- The most popular microcontroller – about 40% of market share
- 8-bit microcontroller

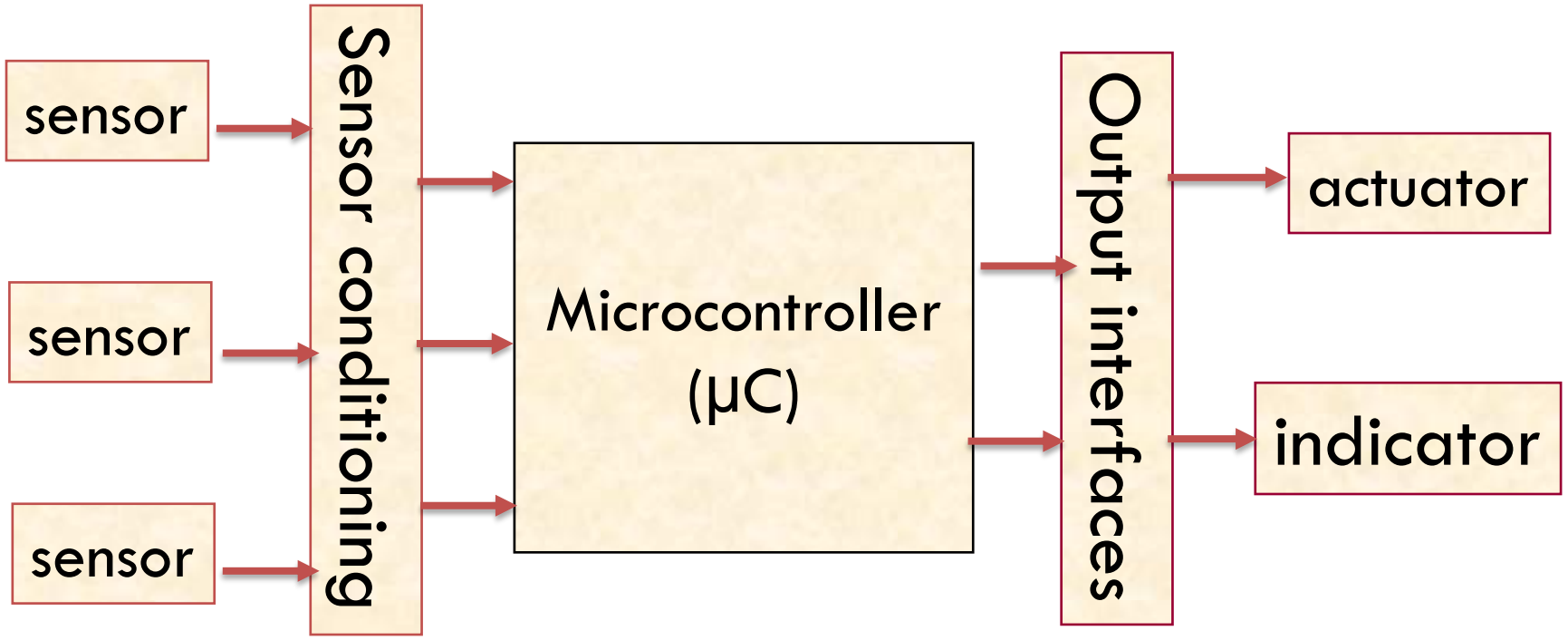
Important features of 8051

- 8-bit ALU, Accumulator and Registers; hence it is an 8-bit microcontroller.
- 8-bit data bus - It can access 8 bits of data in one operation.
- 16-bit address bus - It can access 2^{16} memory locations - 64 kb (65536 locations) each of RAM and ROM.
- On-chip RAM - 128 bytes (data memory)

Important features of 8051

- On-chip ROM- 4kb (program memory)
- Bi-directional input/output ports
- UART (serial port)
- Two 16-bit Counter/timers
- Two-level interrupt

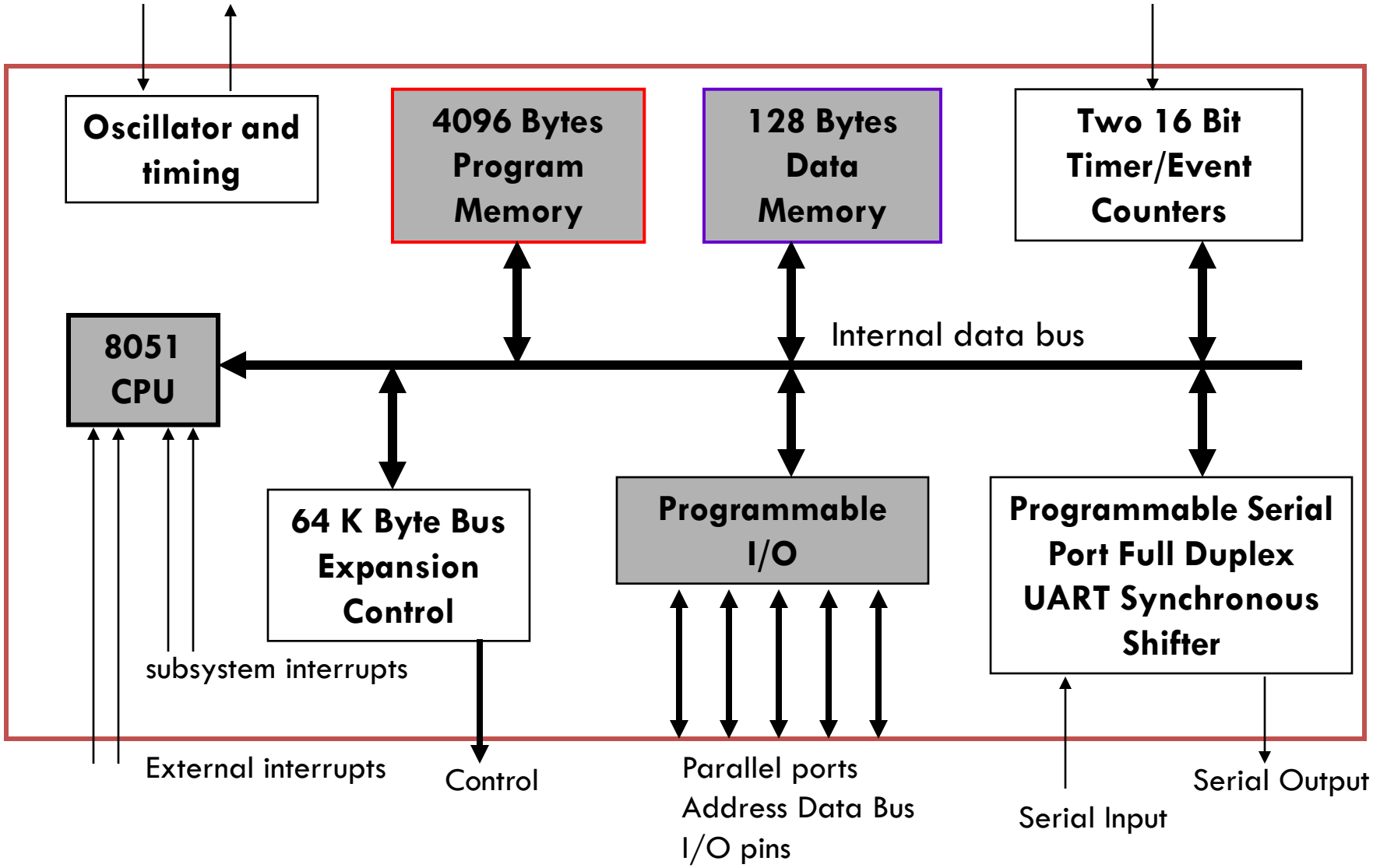
Embedded System General Block Diagram



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.

8051 Architecture

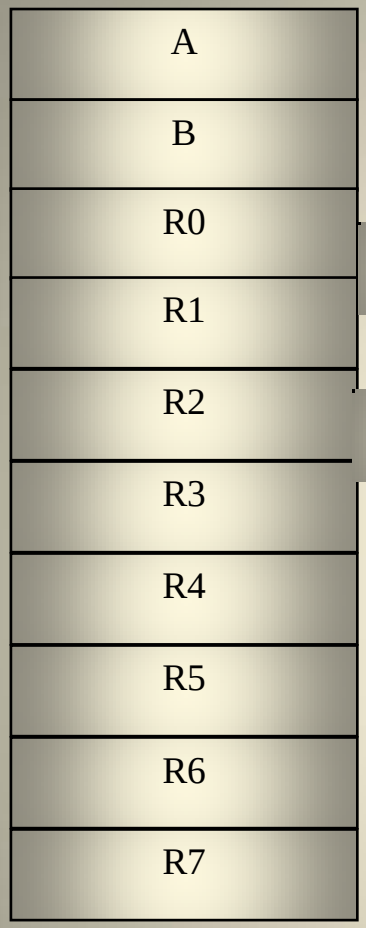


8051 Memory Architecture

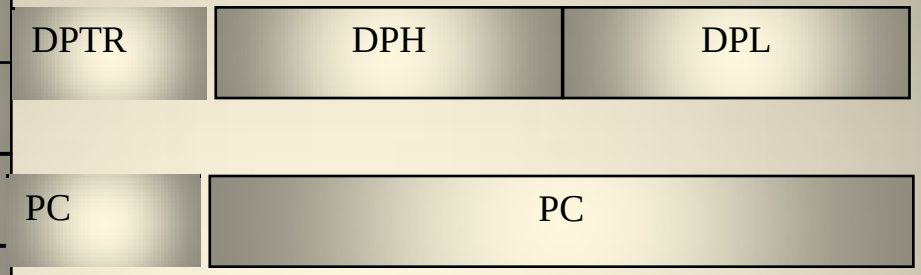
Memory Model

- **Program Memory**
  - Internal ROM (4k)
  - External EPROM
- **Data Memory**
  - Internal RAM (128 bytes)
    - General Purpose Registers
    - Special Function Registers
  - External SRAM

8051 General Purpose Registers



Some 8-bit Registers of the 8051



Some 8051 16-bit Register

Note:
A= accumulator
PC=program counter
DPTR=data pointer

8051 Special Function Registers(SFRs)

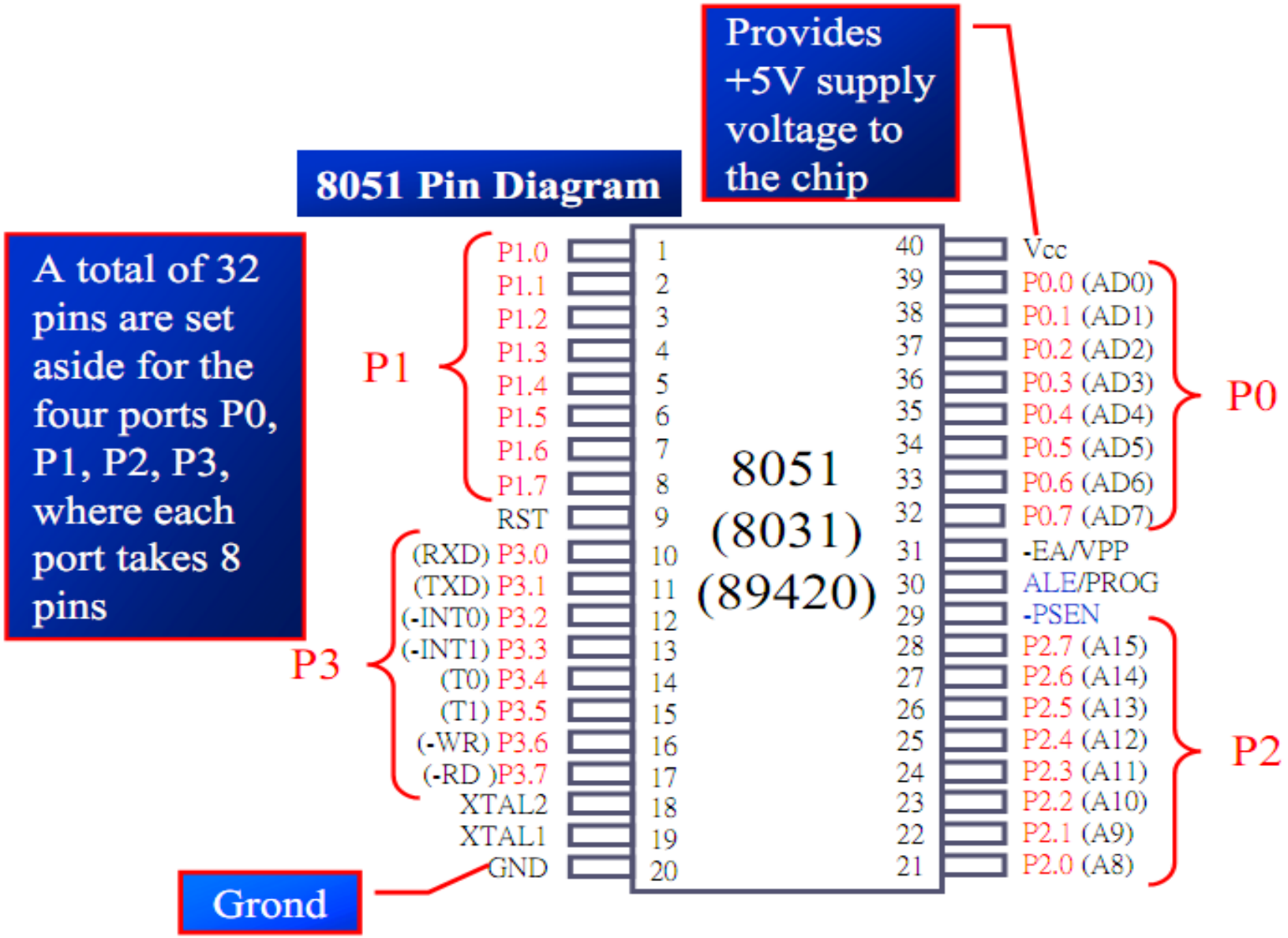
Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B register	0F0H
PSW*	Program status word	0D0H
SP	Stack pointer	81H
DPTR	Data pointer 2 bytes	
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H
P3*	Port 3	0B0H
IP*	Interrupt priority control	0B8H
IE*	Interrupt enable control	0A8H
…	…	…

Contd...

Symbol	Name	Address
TMOD	Timer/counter mode control	89H
TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
T2MOD	Timer/counter mode control	OC9H
TH0	Timer/counter 0 high byte	8CH
TL0	Timer/counter 0 low byte	8AH
TH1	Timer/counter 1 high byte	8DH
TL1	Timer/counter 1 low byte	8BH
TH2	Timer/counter 2 high byte	0CDH
TL2	Timer/counter 2 low byte	0CCH
RCAP2H	T/C 2 capture register high byte	0CBH
RCAP2L	T/C 2 capture register low byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON	Power ontrol	87H

*** Bit addressable**

Pin Description of the 8051



Pins of 8051

- V_{cc}（pin 40）：
  - V_{cc} provides supply voltage to the chip.
  - The voltage source is +5V.
- GND（pin 20）：ground
- XTAL1 and XTAL2（pins 19,18）

Pins of 8051

- 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.

Pins of 8051

- /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.
  - /PSEN & ALE are used for external ROM.
  - For 8051, /EA pin is connected to V_{cc}.
  - “/” 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

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

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
    - Example：P0.0 is the bit 0（LSB）of P0
    - Example：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).

Port 0

- It can be used for both input or output.
- Each pin must be connected externally to a 10K-ohm pull-up resistors to use the pins of port 0 as both input and output ports (open drain).
- Dual role of PORT 0
  - It is used for both address and data(for 8031).
  - It is also designated as AD0-AD7 for address to be connected to an external memory.
- It must be assigned 1for input and 0 for output.

Port 1

- It can be used for both input or output.
- Does not need any pull up registers.

Port 2

- Can be used for both input or output.
- Does not need any pull up registers.
- Dual role of PORT 2
  - It is used along with P0 to provide 16-bit address for external memory(for 8031).
  - It is also designated as A8-A15 for address to be connected to an external memory.

Port 3

- Can be used for both input or output.
- Does not need any pull up registers.
- **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

Serial communications

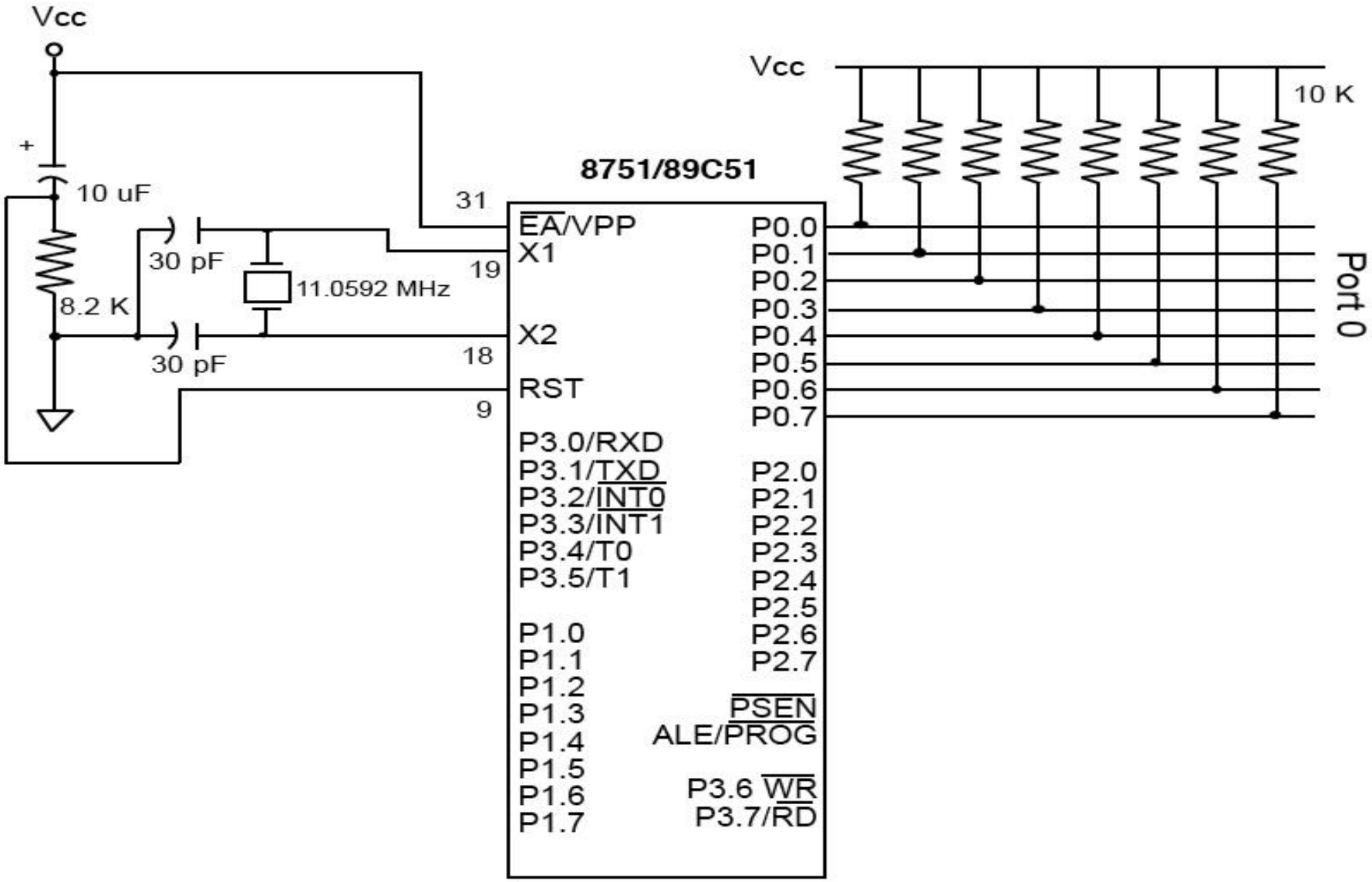
External interrupts

Timers

Read/Write signals of external memories

In systems based on 8751, 89C51 or DS89C4x0, pins 3.6 and 3.7 are used for I/O while the rest of the pins in port 3 are normally used in the alternate function role

Minimum hardware connection for 8051 based system



8051 Instructions Sets

- The 8051 instruction sets are divided into five functional groups:
  - Arithmetic instructions
  - Logical instructions
  - Data transfer instructions
  - Boolean variable instructions
  - Program branching instructions

Arithmetic Instructions

- With arithmetic instructions, 8051 performs all the arithmetic operations.
- Arithmetic operations effect the flags, such as Carry Flag (CY), Overflow Flag (OV) etc, in the PSW register.

Mnemonic	Description
ADD A, Rn	A = A + [Rn]
ADD A, direct	A = A + [direct memory]
ADD A,@Ri	A = A + [memory pointed to by Ri]
ADD A,#data	A = A + immediate data
ADDC A,Rn	A = A + [Rn] + CY
ADDC A, direct	A = A + [direct memory] + CY
ADDC A,@Ri	A = A + [memory pointed to by Ri] + CY
ADDC A,#data	A = A + immediate data + CY
SUBB A,Rn	A = A - [Rn] - CY
SUBB A, direct	A = A - [direct memory] - CY
SUBB A,@Ri	A = A - [@Ri] - CY
SUBB A,#data	A = A - immediate data - CY
INC A	A = A + 1
INC Rn	[Rn] = [Rn] + 1
INC direct	[direct] = [direct] + 1
INC @Ri	[@Ri] = [@Ri] + 1
DEC A	A = A - 1
DEC Rn	[Rn] = [Rn] - 1
DEC direct	[direct] = [direct] - 1
DEC @Ri	[@Ri] = [@Ri] - 1
MUL AB	Multiply A & B
DIV AB	Divide A by B
DA A	Decimal adjust A

Contd…

Note:

- [@Ri] implies contents of memory location pointed to by R0 or R1

- Rn refers to registers R0-R7 of the currently selected register bank

Mnemonic	Description
ANL A, Rn	A = A & [Rn]
ANL A, direct	A = A & [direct memory]
ANL A,@Ri	A = A & [memory pointed to by Ri]
ANL A,#data	A= A & immediate data
ANL direct,A	[direct] = [direct] & A
ANL direct,#data	[direct] = [direct] & immediate data
ORL A, Rn	A = A OR [Rn]
ORL A, direct	A = A OR [direct]
ORL A,@Ri	A = A OR [@RI]
ORL A,#data	A = A OR immediate data
ORL direct,A	[direct] = [direct] OR A
ORL direct,#data	[direct] = [direct] OR immediate data
XRL A, Rn	A = A XOR [Rn]
XRL A, direct	A = A XOR [direct memory]
XRL A,@Ri	A = A XOR [@Ri]
XRL A,#data	A = A XOR immediate data
XRL direct,A	[direct] = [direct] XOR A
XRL direct,#data	[direct] = [direct] XOR immediate data
CLR A	Clear A
CPL A	Complement A
RL A	Rotate A left
RLC A	Rotate A left (through C)
RR A	Rotate A right
RRC A	Rotate A right (through C)
SWAP A	Swap nibbles

Logical Instructions

- Logical instructions perform Boolean operations (AND, OR, XOR, and NOT) on data bytes on a bit-by-bit basis.

Data Transfer Instructions

- Data transfer instructions can be used to transfer data between an internal RAM location and an SFR location without going through the accumulator.
- It is also possible to transfer data between the internal and external RAM by using indirect addressing.

Mnemonic	Description
MOV @Ri, direct	[@Ri] = [direct]
MOV @Ri, #data	[@Ri] = immediate data
MOV DPTR, #data 16	[DPTR] = immediate data
MOVC A,@A+DPTR	A = Code byte from [@A+DPTR]
MOVC A,@A+PC	A = Code byte from [@A+PC]
MOVX A,@Ri	A = Data byte from external ram [@Ri]
MOVX A,@DPTR	A = Data byte from external ram [@DPTR]
MOVX @Ri, A	External[@Ri] = A
MOVX @DPTR,A	External[@DPTR] = A
PUSH direct	Push into stack
POP direct	Pop from stack
XCH A,Rn	A = [Rn], [Rn] = A
XCH A, direct	A = [direct], [direct] = A
XCH A, @Ri	A = [@Rn], [@Rn] = A
XCHD A,@Ri	Exchange low order digits

Contd…

Boolean Variable Instructions

- The 8051 processor can perform single bit operations.
- The operations include set, clear, and, or and complement instructions.
- Also included are bit–level moves or conditional jump instructions.

Mnemonic	Description
CLR C	Clear C
CLR bit	Clear direct bit
SETB C	Set C
SETB bit	Set direct bit
CPL C	Complement c
CPL bit	Complement direct bit
ANL C,bit	AND bit with C
ANL C,/bit	AND NOT bit with C
ORL C,bit	OR bit with C
ORL C,/bit	OR NOT bit with C
MOV C,bit	MOV bit to C
MOV bit,C	MOV C to bit
JC rel	Jump if C set
JNC rel	Jump if C not set
JB bit,rel	Jump if specified bit set
JNB bit,rel	Jump if specified bit not set
JBC bit,rel	if specified bit set then clear it and jump

Contd…

Program Branching Instructions

- Program branching instructions are used to control the flow of program execution
- Some instructions provide decision making capabilities before transferring control to other parts of the program (conditional branches).

Mnemonic	Description
ACALL addr11	Absolute subroutine call
LCALL addr16	Long subroutine call
RET	Return from subroutine
RETI	Return from interrupt
AJMP addr11	Absolute jump
LJMP addr16	Long jump
SJMP rel	Short jump
JMP @A+DPTR	Jump indirect
JZ rel	Jump if A=0
JNZ rel	Jump if A NOT=0
CJNE A,direct,rel	Compare and Jump if Not Equal
CJNE A,#data,rel	
CJNE Rn,#data,rel	
CJNE @Ri,#data,rel	
DJNZ Rn,rel	Decrement and Jump if Not Zero
DJNZ direct,rel	
NOP	No Operation

Contd…

8051 Programming with C and Assembly

Write a assembly program that continuously toggles the value of port 0.

The following code will continuously send out to port 0 the alternating value 55H and AAH

```
BACK:  MOV    A,#55H
       MOV    P0,A
       ACALL  DELAY        // Absolute subroutine call
       MOV    A,#0AAH
       MOV    P0,A
       ACALL  DELAY
       SJMP   BACK         // Short Jump
```

Contd…

Corresponding C program:

```
#include <regx51.h>

Void Delay(unsigned int);

Void main(void)
{
    P0=0x00;        //make P0 an output
                    //port
    while(1)
    {
        P0=0x55;
        Delay(200);
        P0=0xAA;
        Delay(200);
    }
}
```

```
Void Delay(unsigned int n)
{
    unsigned int i, j;
    for(i=0; i<n; i++)
        for(j=0; j<1275; j++);
}
```

Contd…

Write assembly program to get data from P0 and send it to P1.

Port 0 is configured first as an input port by writing 1s to it, and then data is received from that port and sent to P1

```
        MOV     A,#0FFH         ;A=FF hex
        MOV     P0,A            ;make P0 an i/p port
        MOV     A, #00H         ;by writing it all 1s
        MOV     P1, A
BACK:   MOV     A,P0            ;get data from P0
        MOV     P1,A            ;send it to port 1
        SJMP    BACK            ;keep doing it
```

Corresponding C program:

Contd…

```
#include <regx51.h>

Void main(void)
{
    P0=0xFF;      //make P0 an input port
    P1=0x00;      //make P1 an output port
    while(1)
    {
        P1=P0;
    }
}
```

Contd…

Write the following programs.
Create a square wave of 50% duty cycle on bit 0 of port 1.

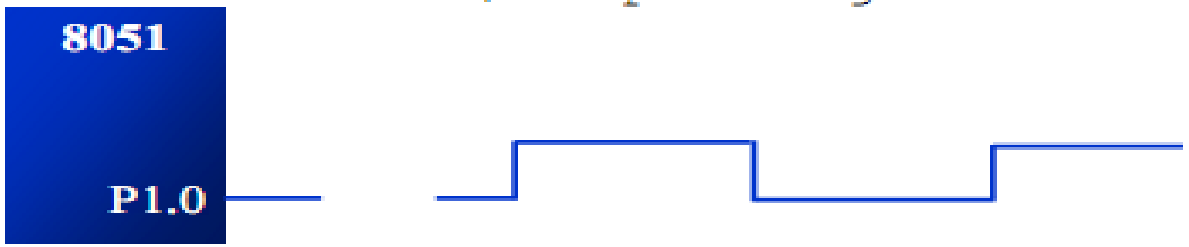
Solution:

The 50% duty cycle means that the "on" and "off" state (or the high and low portion of the pulse) have the same length. Therefore, we toggle P1.0 with a time delay in between each state.

```
HERE:   SETB    P1.0    ;set to high bit 0 of port 1
        LCALL   DELAY   ;call the delay subroutine
        CLR     P1.0    ;P1.0=0
        LCALL   DELAY
        SJMP    HERE    ;keep doing it
```

Another way to write the above program is:

```
HERE:   CPL     P1.0    ;set to high bit 0 of port 1
        LCALL   DELAY   ;call the delay subroutine
        SJMP    HERE    ;keep doing it
```



Contd…

Corresponding C program:

```
#include <reg51.h>
void T1M2Delay(void);
sbit mybit= P1^0
void main(v

   while (1) {
     mybit=~mybit;
     T1M2Delay();
   }
}
void T1M2Delay(void){
   TMOD=0x20;
   TH1=-184;
   TR1=1;
   while (TF1==0);
   TR1=0;
   TF1=0;
}
```

1/2500 Hz = 400 µs

400 µs /2 = 200 µs

200 µs / 1.085 µs = 184

Contd…

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

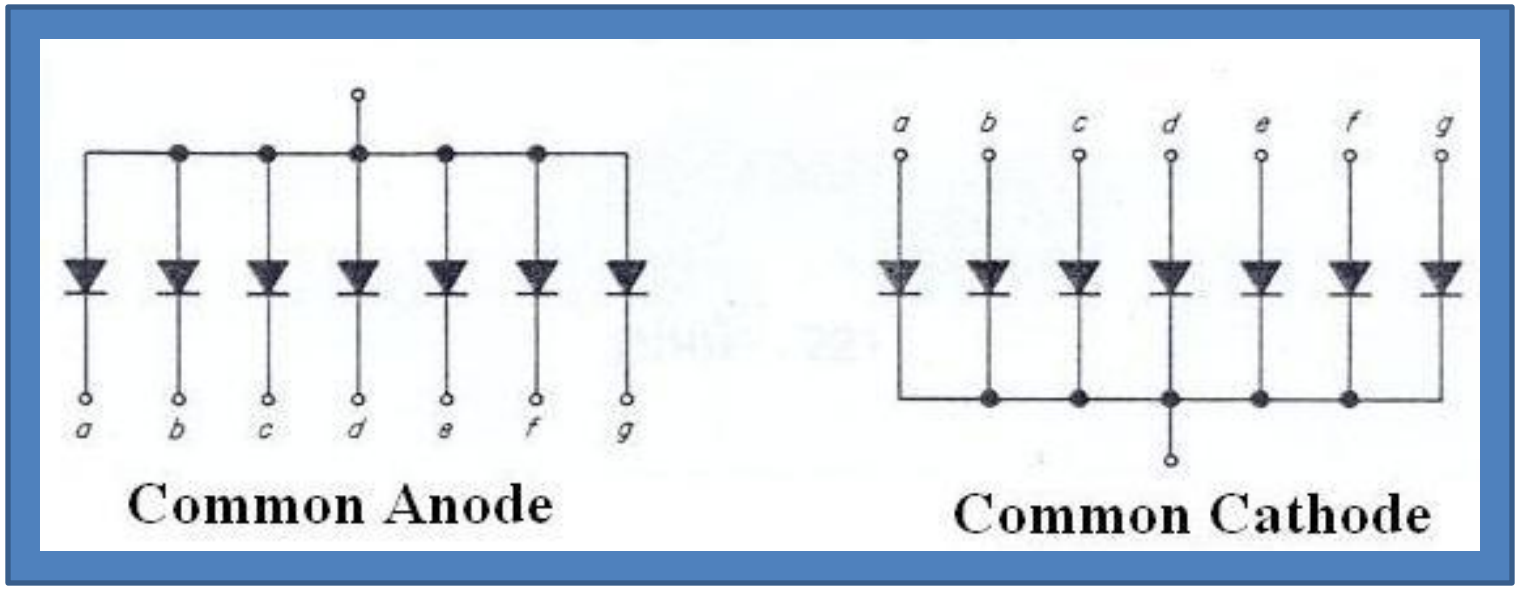
Solution:

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

The instruction 'MOV P2.7,P1.0' is wrong , since such an instruction does not exist

However 'MOV P2, P1' is a valid instruction

Interfacing 7 segment display with 8051

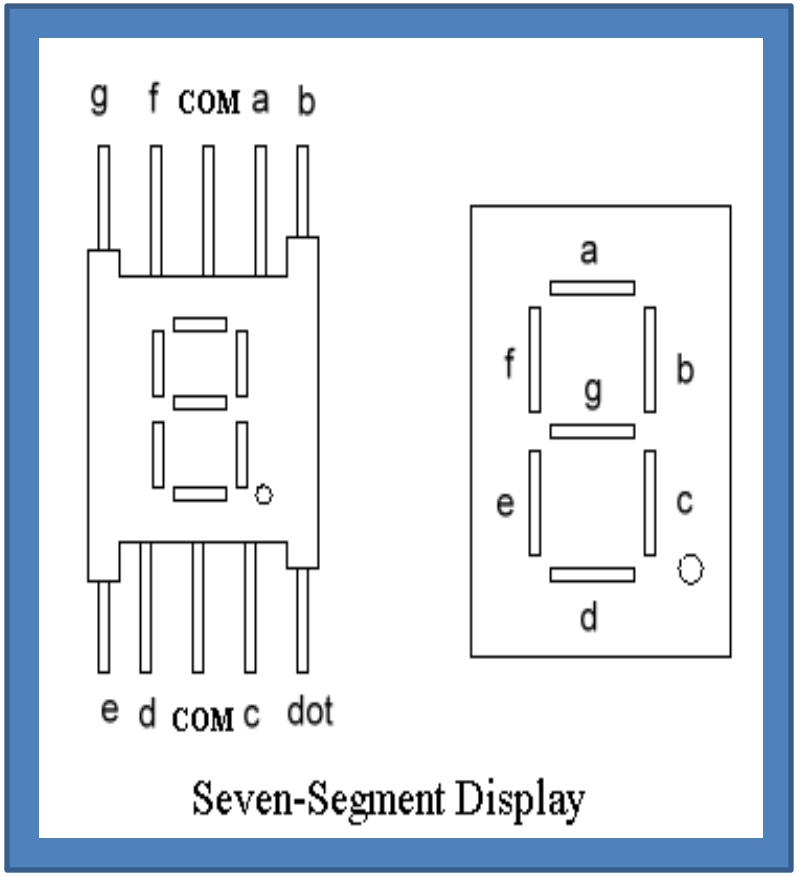


Basically there are two types of 7-Seg display's:

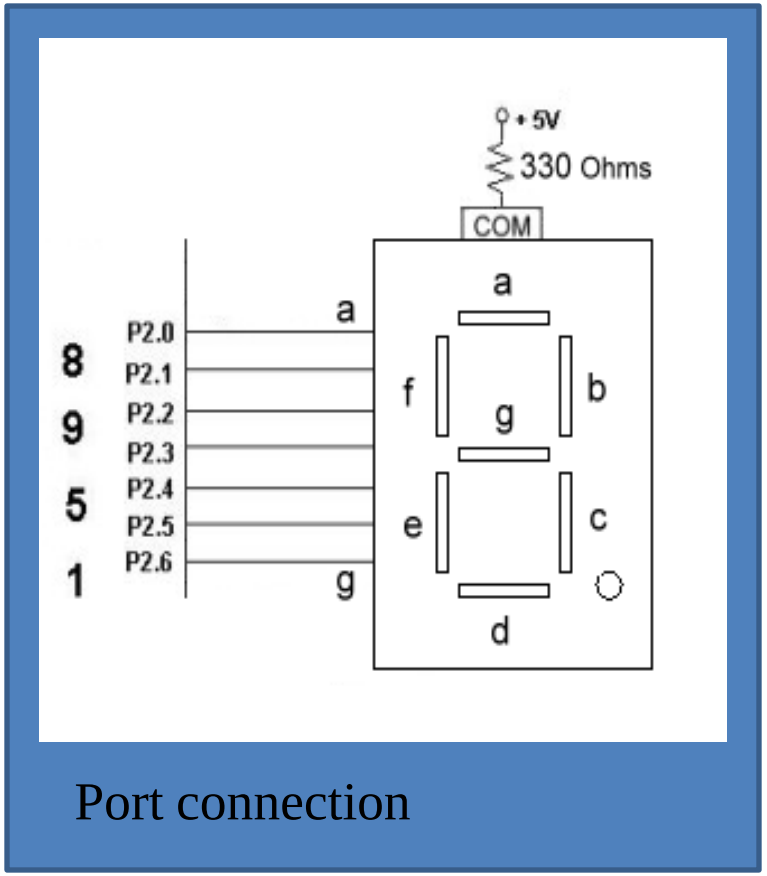
1. Common Cathode: where all the segments share the same Cathode.
2. Common Anode: where all Segments share the same Anode.

Here we will be only discussing the Common Anode type. In common Anode in order to turn ON a segment the corresponding pin must be set to 0. And to turn it OFF it is set to 1.

Contd…



Seven-Segment Display

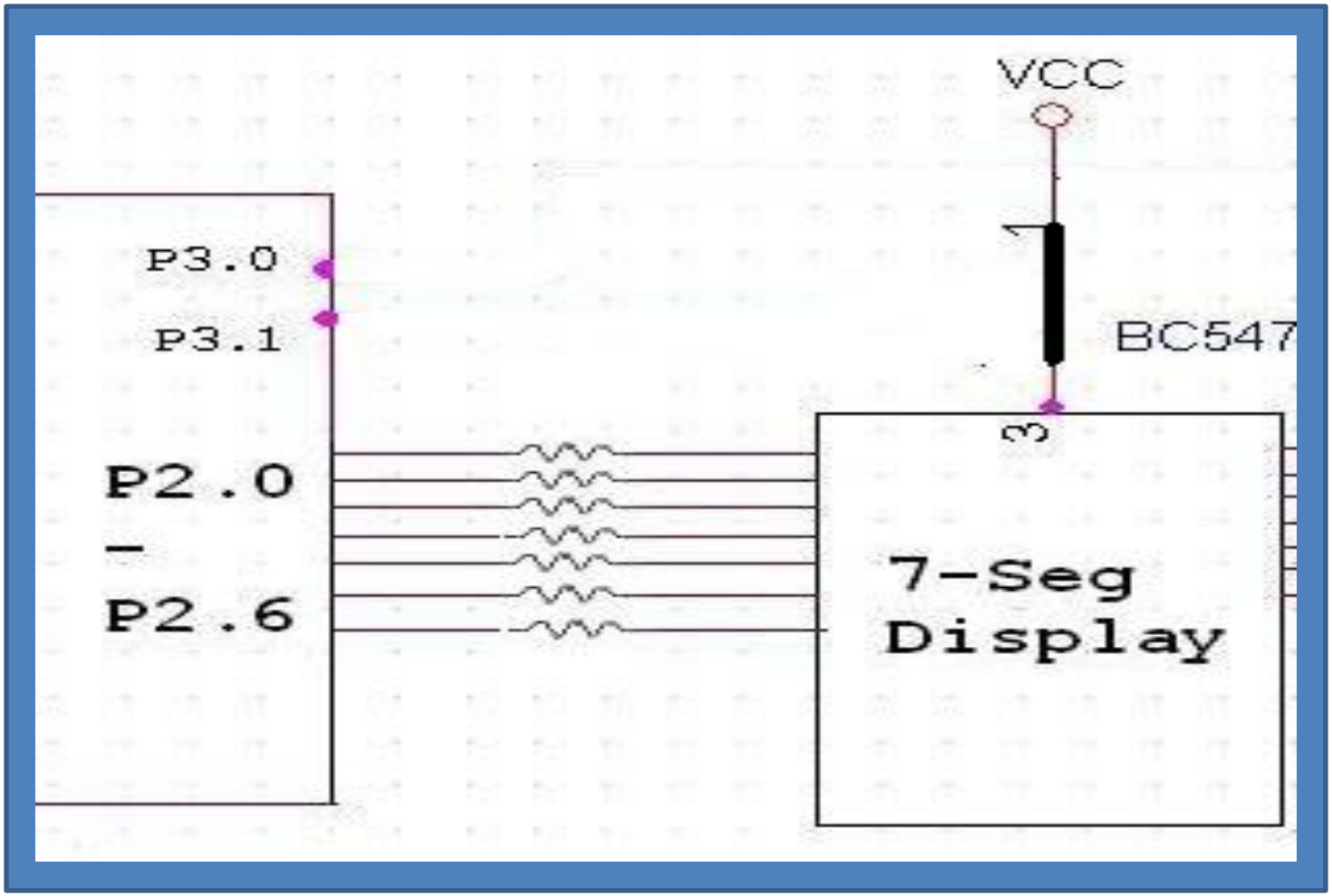


Port connection

Lookup Table for 7 Segment Decoding

Hex Number	Seven Segment conversion								Seven Segment equivalent
	dot	g	f	e	d	c	b	a	
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F8
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	1	0	0	0	98

Hardware connection of 7 segment with 8051



Assembly program to display 0 to 9 in 7 segment display

```
MOV   A, #00H
MOV   P2, A        // make P2 an output port

MOV   P2, #C0                  MOV   P2, #82
ACALL DELAY                    ACALL DELAY
MOV   P2, #F9                  MOV   P2, #F8
ACALL DELAY                    ACALL DELAY
MOV   P2, #A4                  MOV   P2, #80
ACALL DELAY                    ACALL DELAY
MOV   P2, #B0                  MOV   P2, #98
ACALL DELAY                    ACALL DELAY
MOV   P2, #99
ACALL DELAY
MOV   P2, #92
ACALL DELAY
```

C Program Contd…

```
#include <regx51.h>

Void Delay(unsigned int);

Void main(void)
{
    P2=0x00;    //make P0 an output port
    P2=0xC0;
    Delay(200);
    P2=0xF9;
    Delay(200);
    P2=0xA4;
    Delay(200);
    P2=0xB0;
    Delay(200);
```

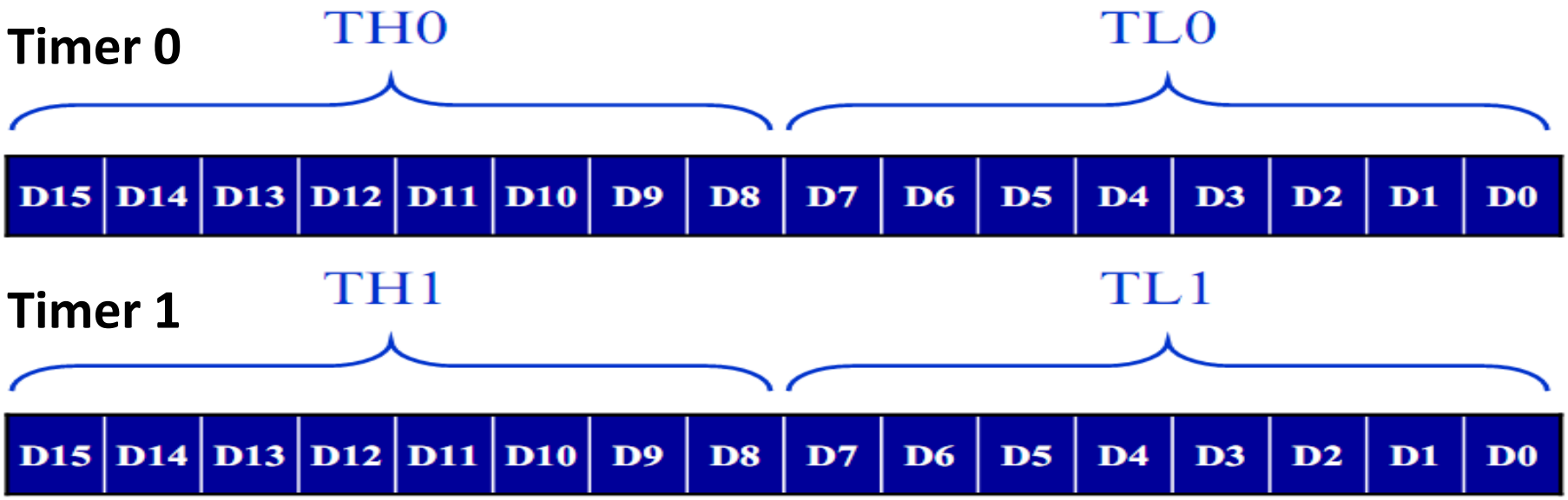
```
    P2=0x82;
    Delay(200);
    P2=0xF8;
    Delay(200);
    P2=0x80;
    Delay(200);
    P2=0x98;
 }


Void Delay(unsigned int n)
{
    unsigned int i, j;
    for(i=0; i<n; i++)
        for(j=0; j<1275; j++);
}
```

Contd…

Timers

- The 8051 has two timers/counters, they can be used either as
  - Timers to generate a time delay
  - or as Event counters to count events happening outside the microcontroller
- Both Timer 0 and Timer 1 are 16 bits wide.
- Since 8051 has an 8-bit architecture, each 16-bits timer is accessed as two separate registers of low byte and high byte.
  - The low byte register is called TL0/TL1 and
  - The high byte register is called TH0/TH1

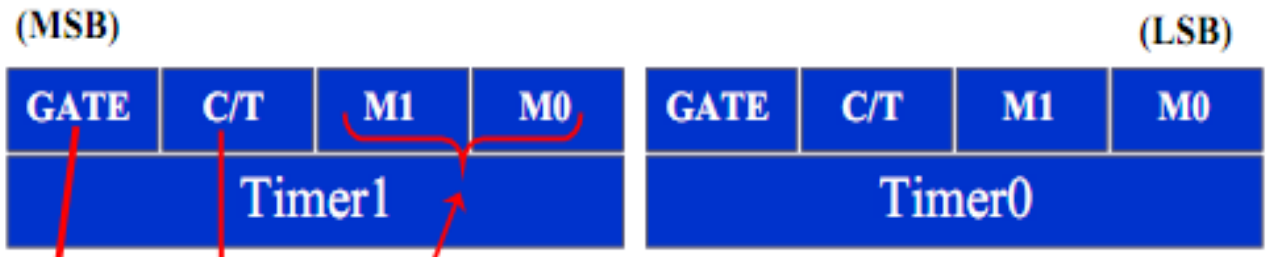
Contd…



- Both timers 0 and 1 use the same register, called TMOD (timer mode), to set the various timer operation modes.

PROGRAMMING TIMERS

TMOD Register (cont')

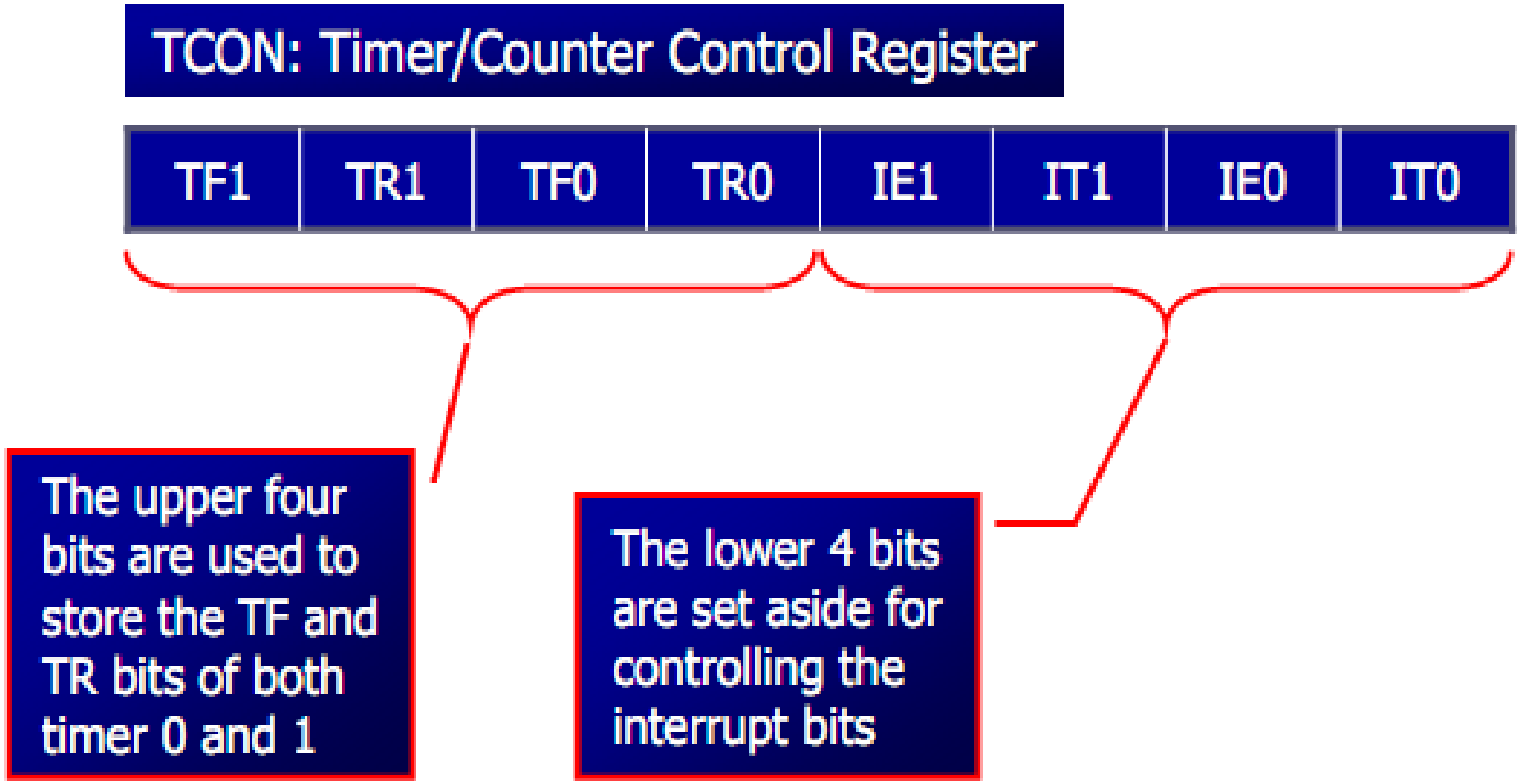


M1	M0	Mode	Operating Mode
0	0	0	**13-bit timer mode** 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	**16-bit timer mode** 16-bit timer/counter THx and TLx are cascaded; there is no prescaler
1	0	2	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded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

When GATE=0, Timer ON/OFF is controller using software.
When GATE=1, Timer ON/OFF is controller using hardware.

Timer or counter selected
Cleared for timer operation (input from internal system clock)
Set for counter operation (input from Tx input pin)

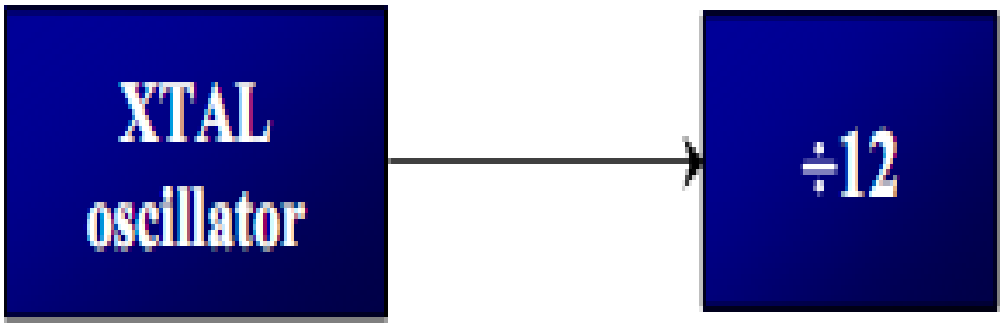
TCON (Timer control) Register



Contd…

Find the timer's clock frequency and its period for various 8051-based system, with the crystal frequency 11.0592 MHz when C/T bit of TMOD is 0.

Solution:



1/12 × 11.0529 MHz = 921.6 MHz;

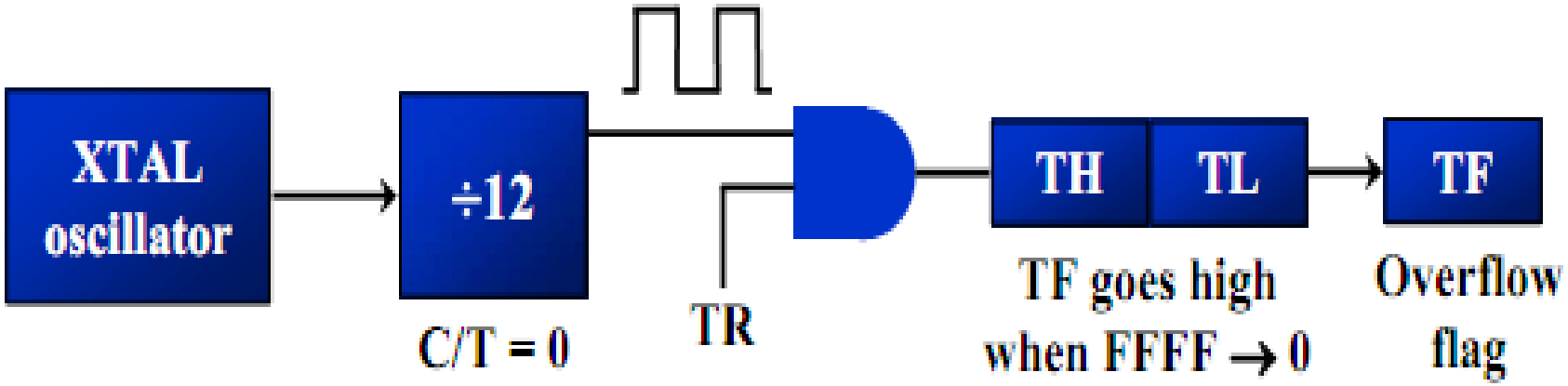
T = 1/921.6 kHz = 1.085 us

Timer mode 1 programming

- **The following are the characteristics and operations of mode1:**
  - **It is a 16-bit timer; therefore, it allows value of 0000 to FFFFH to be loaded into the timer's register TL and TH**
  - **After TH and TL are loaded with a 16-bit initial value, the timer must be started**
  - **This is done by setting high TR0 for timer0 and TR1 for timer1**
  - **After the timer is started, it starts to count up**
  - **It counts up until it reaches its limit of FFFFH**
  - **When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag).Each timer has its own timer flag: TF0 for timer 0, and TF1 for timer 1.**

Contd…

- When this timer flag is raised, one option would be to stop the timer.
- After the timer reaches its limit and rolls over, in order to repeat the process TH and TL must be reloaded with the original value, and TF must be reloaded to 0.



Delay generation process

1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used and which timer mode (0 or 1) is selected.
2. Load registers TL and TH with initial count value.
3. Start the timer.
4. Keep monitoring the timer flag (TF). if it is raised,
5. Stop the timer.
6. Clear the TF flag for the next round
7. Go back to Step 2 to load TH and TL again

How to calculate values to be loaded into TH and TL

Assume XTAL = 11.0592 MHz, we can use the following steps for finding the TH and TL registers' values,

1. **Divide the desired time delay by 1.085 us.**
2. **Calculate 65536 – n, where n is the decimal value we got in Step1.**
3. **Convert the result of Step2 to hex, where yyxx is the initial hex value to be loaded into the timer's register**
4. **Set TL = xx and TH = yy.**

Program that generates 56 ms delay

```
void T0Delay(){
   TMOD=0x01;
   TL0=0x00;
   TH0=0x35;
   TR0=1;
   while (TF0==0);
   TR0=0;
   TF0=0;
}
```

FFFFH – 3500H = CAFFH

= 51967 + 1 = 51968

51968 x 1.085 µs = 56.384 ms is the approximate delay

Interrupt

- **Concept behind Interrupt**
  - **Interrupt vs Polling**
  - **What is the advantage of having interrupt based system over polling system**
- **Interrupt Process:**

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

1. **It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack**
2. **It also saves the current status of all the interrupts internally (i.e: not on the stack)**

Contd…

3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR.
4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it.
   - It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted and starts executing form that address.

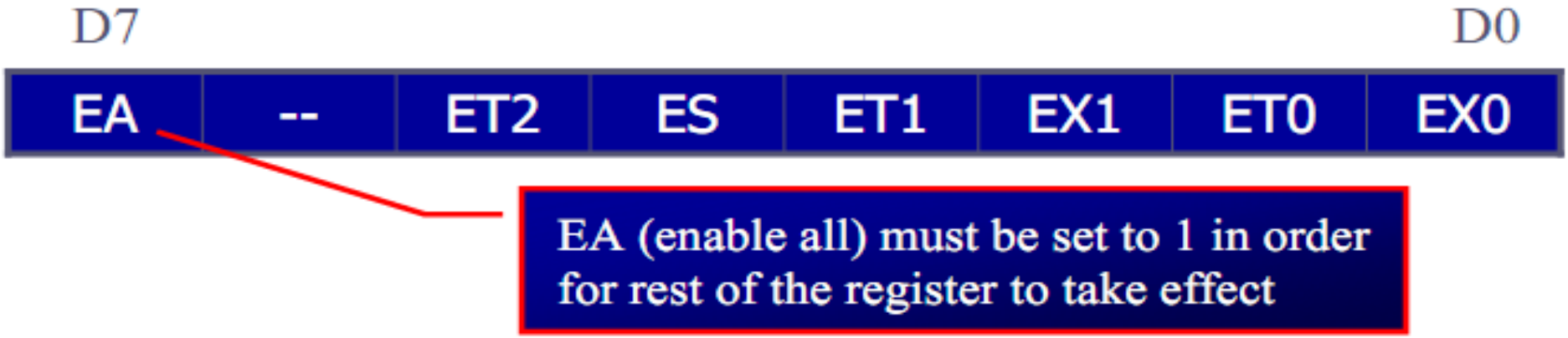
Interrupt sources of 8051

Interrupt	ROM Location (hex)	Pin
Reset	0000	9
External HW (INT0)	0003	P3.2 (12)
Timer 0 (TF0)	000B	
External HW (INT1)	0013	P3.3 (13)
Timer 1 (TF1)	001B	
Serial COM (RI and TI)	0023	

Level triggered (normally HIGH) and edge trigged (falling edge)

Interrupt Programming

IE (Interrupt Enable) Register



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

TCON (Timer/Counter) Register (Bit-addressable)

D7 D0

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the interrupt service routine
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off

TCON (Timer/Counter) Register (Bit-addressable) (cont')

IE1	TCON.3	External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the 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 CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt

Interrupt Priority Upon Reset

Highest To Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)

- We can alter the sequence of interrupt priority by assigning a higher priority to any one of the interrupts by programming a register called IP (interrupt priority)

Interrupt Priority Register (Bit-addressable)

D7							D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

--	IP.7	Reserved
--	IP.6	Reserved
PT2	IP.5	Timer 2 interrupt priority bit (8052 only)
PS	IP.4	Serial port interrupt priority bit
PT1	IP.3	Timer 1 interrupt priority bit
PX1	IP.2	External interrupt 1 priority bit
PT0	IP.1	Timer 0 interrupt priority bit
PX0	IP.0	External interrupt 0 priority bit

Priority bit=1 assigns high priority

Priority bit=0 assigns low priority

Assume that after reset, the interrupt priority is set the instruction `MOV IP,#00001100B`. Discuss the sequence in which the interrupts are serviced.

Solution:

The instruction "MOV IP #00001100B" (B is for binary) and timer 1 (TF1)to a higher priority level compared with the reset of the interrupts. However, since they are polled according to Table, they will have the following priority.

Highest Priority	External Interrupt 1	(INT1)
	Timer Interrupt 1	(TF1)
	External Interrupt 0	(INT0)
	Timer Interrupt 0	(TF0)
Lowest Priority	Serial Communication	(RI+TI)

PROGRAMMING IN C (cont')

Example 11-14

Write a C program that continuously gets a single bit of data from P1.7 and sends it to P1.0, while simultaneously creating a square wave of 200 µs period on pin P2.5. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

Solution:

We will use timer 0 mode 2 (auto-reload). One half of the period is 100 µs. 100/1.085 µs = 92, and TH0 = 256 - 92 = 164 or A4H

```
#include <reg51.h>
sbit SW   =P1^7;
sbit IND  =P1^0;
sbit WAVE =P2^5;
void timer0(void) interrupt 1 {
  WAVE=~WAVE;  //toggle pin
}
void main() {
  SW=1;           //make switch input
  TMOD=0x02;
  TH0=0xA4;       //TH0=-92
  IE=0x82;        //enable interrupt for timer 0
  while (1) {
    IND=SW;       //send switch to LED
  }
}
```

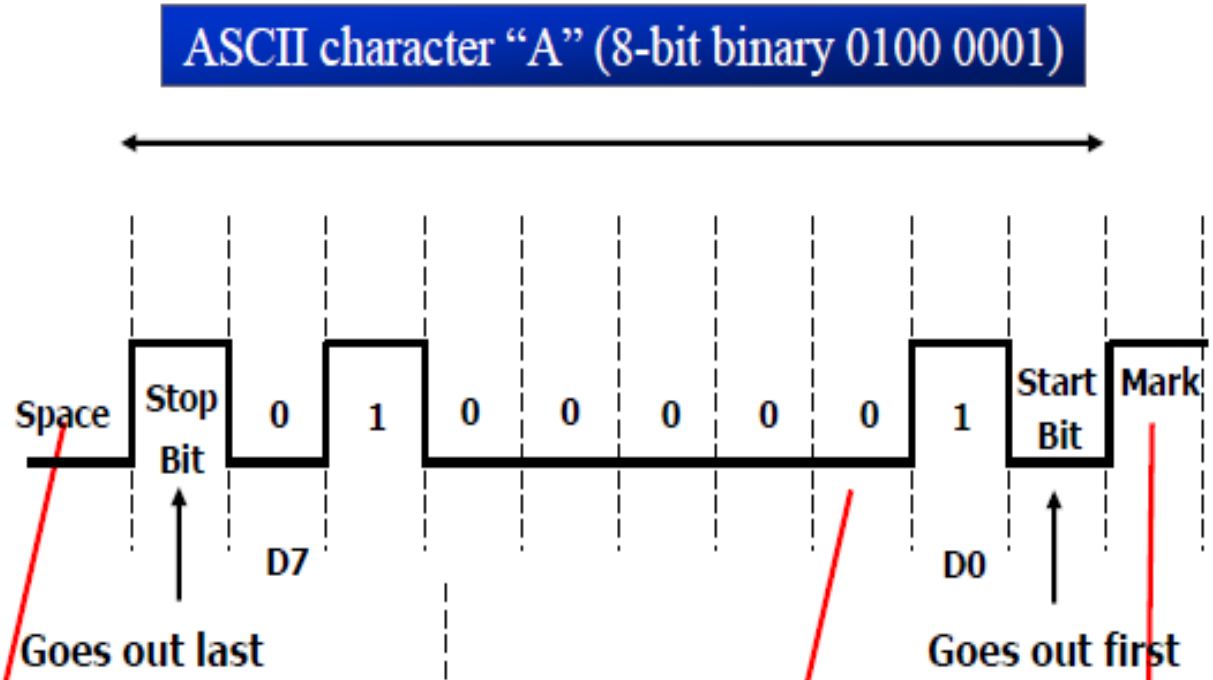
Serial communication

- Different ways of communication
  - wireless
  - Wired
- Protoc0l: set of rules agreed by both the sender and receiver on
  - How the data is packed
  - How many bits constitute a character
  - When the data begins and ends

BASICS OF SERIAL COMMUNICA-TION

Start and Stop Bits (cont')

- The start bit is always a 0 (low) and the stop bit(s) is 1 (high)



The 0 (low) is referred to as *space*

The transmission begins with a start bit followed by D0, the LSB, then the rest of the bits until MSB (D7), and finally, the one stop bit indicating the end of the character

When there is no transfer, the signal is 1 (high), which is referred to as *mark*

SERIAL COMMUNICA-TION PROGRAMMING

SBUF Register

- SBUF is an 8-bit register used solely for serial communication
  - For a byte data to be transferred via the TxD line, it must be placed in the SBUF register
    - The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line
  - SBUF holds the byte of data when it is received by 8051 RxD line
    - When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF

SERIAL COMMUNICA-TION PROGRAMMING

SCON Register

- SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

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

Note: *Make SM2, TB8, and RB8 =0*

SM0	SM1	
0	0	Serial Mode 0
0	**1**	**Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit**
1	0	Serial Mode 2
1	1	Serial Mode 3

Only mode 1 is of interest to us

M2

SERIAL COMMUNICA-TION PROGRAMMING (cont')

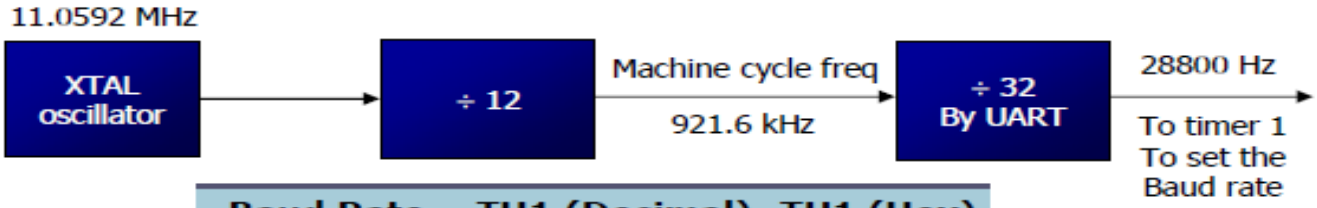
With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates. (a) 9600 (b) 2400 (c) 1200

Solution:

The machine cycle frequency of 8051 = 11.0592 / 12 = 921.6 kHz, and 921.6 kHz / 32 = 28,800 Hz is frequency by UART to timer 1 to set baud rate.

(a) 28,800 / 3 = 9600 where -3 = FD (hex) is loaded into TH1
(b) 28,800 / 12 = 2400 where -12 = F4 (hex) is loaded into TH1
(c) 28,800 / 24 = 1200 where -24 = E8 (hex) is loaded into TH1

Notice that dividing 1/12 of the crystal frequency by 32 is the default value upon activation of the 8051 RESET pin.



Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

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

SERIAL COMMUNICA-TION PROGRAMMING

Programming Serial Data Transmitting

- In programming the 8051 to transfer character bytes serially
  1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate
  2. The TH1 is loaded with one of the values to set baud rate for serial data transfer
  3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
  4. TR1 is set to 1 to start timer 1
  5. TI is cleared by `CLR TI` instruction
  6. The character byte to be transferred serially is written into SBUF register
  7. The TI flag bit is monitored with the use of instruction `JNB TI,xx` to see if the character has been transferred completely
  8. To transfer the next byte, go to step 5

SERIAL COMMUNICA-TION PROGRAMMING

Programming Serial Data Receiving

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

SERIAL PORT PROGRAMMING IN C

Transmitting and Receiving Data

Example 10-15

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

Solution:

```
#include <reg51.h>
void main(void){
  TMOD=0x20;          //use Timer 1, mode 2
  TH1=0xFA;           //4800 baud rate
  SCON=0x50;
  TR1=1;
  while (1) {
    SBUF='A';         //place value in buffer
    while (TI==0);
    TI=0;
  }
}
```

SERIAL PORT PROGRAMMING IN C

Transmitting and Receiving Data (cont')

Example 10-16

Write an 8051 C program to transfer the message "YES" serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

Solution:

```
#include <reg51.h>
void SerTx(unsigned char);
void main(void){
  TMOD=0x20;          //use Timer 1, mode 2
  TH1=0xFD;           //9600 baud rate
  SCON=0x50;
  TR1=1;              //start timer
  while (1) {
    SerTx('Y');
    SerTx('E');
    SerTx('S');
  }
}
void SerTx(unsigned char x){
  SBUF=x;             //place value in buffer
  while (TI==0);      //wait until transmitted
  TI=0;
}
```

SERIAL PORT PROGRAMMING IN C

Transmitting and Receiving Data (cont')

Example 10-17

Program the 8051 in C 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:

```
#include <reg51.h>
void main(void){
  unsigned char mybyte;
  TMOD=0x20;          //use Timer 1, mode 2
  TH1=0xFA;           //4800 baud rate
  SCON=0x50;
  TR1=1;              //start timer
  while (1) {         //repeat forever
    while (RI==0);    //wait to receive
    mybyte=SBUF;      //save value
    P1=mybyte;        //write value to port
    RI=0;
  }
}
```