8051 MICROCONTROLLERS

The 8051 Microcontroller and Embedded Systems: Using Assembly and C
Mazidi, Mazidi and McKinlay

Chung-Ping Young
楊中平

Home Automation, Networking, and Entertainment Lab

Dept. of Computer Science and Information Engineering
National Cheng Kung University, TAIWAN

OUTLINES

- Microcontrollers and embedded processors
- Overview of the 8051 family

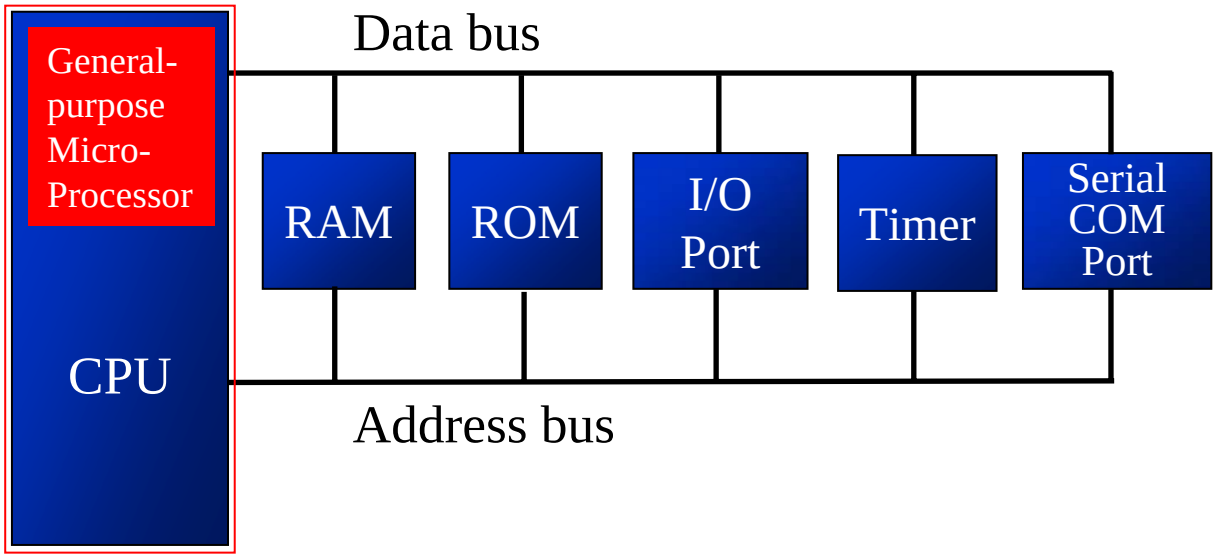
MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

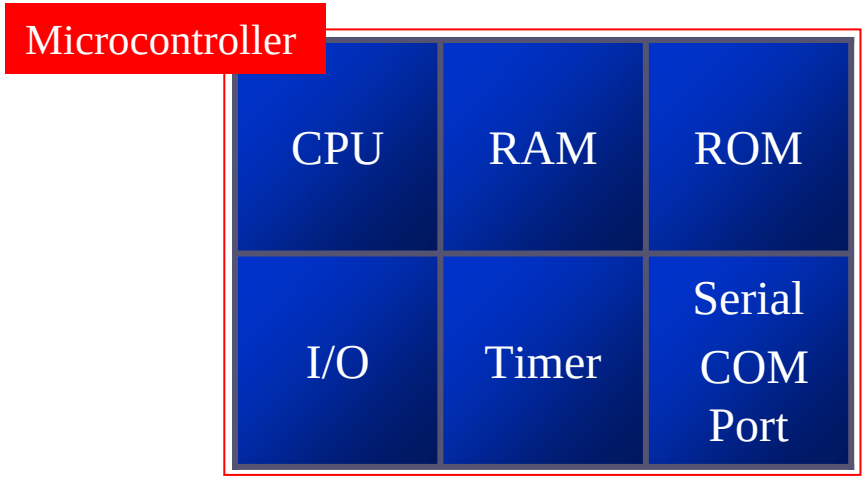
Microcontroller vs. General-Purpose Microprocessor

- General-purpose microprocessors contains
  - No RAM
  - No ROM
  - No I/O ports
- Microcontroller has
  - CPU (microprocessor)
  - RAM
  - ROM
  - I/O ports
  - Timer
  - ADC and other peripherals

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Microcontroller vs. General-Purpose Microprocessor (cont')





MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Microcontroller vs. General-Purpose Microprocessor (cont')

- General-purpose microprocessors
  - Must add RAM, ROM, I/O ports, and timers externally to make them functional
  - Make the system bulkier and much more expensive
  - Have the advantage of versatility on the amount of RAM, ROM, and I/O ports
- Microcontroller
  - The fixed amount of on-chip ROM, RAM, and number of I/O ports makes them ideal for many applications in which cost and space are critical
  - In many applications, the space it takes, the power it consumes, and the price per unit are much more critical considerations than the computing power

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Microcontrollers for Embedded Systems

- An embedded product uses a microprocessor (or microcontroller) to do one task and one task only
  - There is only one application software that is typically burned into ROM
- A PC, in contrast with the embedded system, can be used for any number of applications
  - It has RAM memory and an operating system that loads a variety of applications into RAM and lets the CPU run them
  - A PC contains or is connected to various embedded products
    - Each one peripheral has a microcontroller inside it that performs only one task

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Microcontrollers for Embedded Systems (cont')

- Home
  - Appliances, intercom, telephones, security systems, garage door openers, answering machines, fax machines, home computers, TVs, cable TV tuner, VCR, camcorder, remote controls, video games, cellular phones, musical instruments, sewing machines, lighting control, paging, camera, pinball machines, toys, exercise equipment
- Office
  - Telephones, computers, security systems, fax machines, microwave, copier, laser printer, color printer, paging
- Auto
  - Trip computer, engine control, air bag, ABS, instrumentation, security system, transmission control, entertainment, climate control, cellular phone, keyless entry

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

x86 PC Embedded Applications

- Many manufactures of general-purpose microprocessors have targeted their microprocessor for the high end of the embedded market
  - There are times that a microcontroller is inadequate for the task
- When a company targets a general-purpose microprocessor for the embedded market, it optimizes the processor used for embedded systems
- Very often the terms *embedded processor* and *microcontroller* are used interchangeably

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

x86 PC Embedded Applications (cont')

- One of the most critical needs of an embedded system is to decrease power consumption and space
- In high-performance embedded processors, the trend is to integrate more functions on the CPU chip and let designer decide which features he/she wants to use
- In many cases using x86 PCs for the high-end embedded applications
  - Saves money and shortens development time
    - A vast library of software already written
    - Windows is a widely used and well understood platform

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Choosing a Microcontroller

- 8-bit microcontrollers
  - Motorola's 6811
  - Intel's 8051
  - Zilog's Z8
  - Microchip's PIC
- There are also 16-bit and 32-bit microcontrollers made by various chip makers

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Criteria for Choosing a Microcontroller

- Meeting the computing needs of the task at hand efficiently and cost effectively
  - Speed
  - Packaging
  - Power consumption
  - The amount of RAM and ROM on chip
  - The number of I/O pins and the timer on chip
  - How easy to upgrade to higher-performance or lower power-consumption versions
  - Cost per unit

MICRO-CONTROLLERS AND EMBEDDED PROCESSORS

Criteria for Choosing a Microcontroller (cont')

- Availability of software development tools, such as compilers, assemblers, and debuggers
- Wide availability and reliable sources of the microcontroller
  - The 8051 family has the largest number of diversified (multiple source) suppliers
    - Intel (original)
    - Atmel
    - Philips/Signetics
    - AMD
    - Infineon (formerly Siemens)
    - Matra
    - Dallas Semiconductor/Maxim

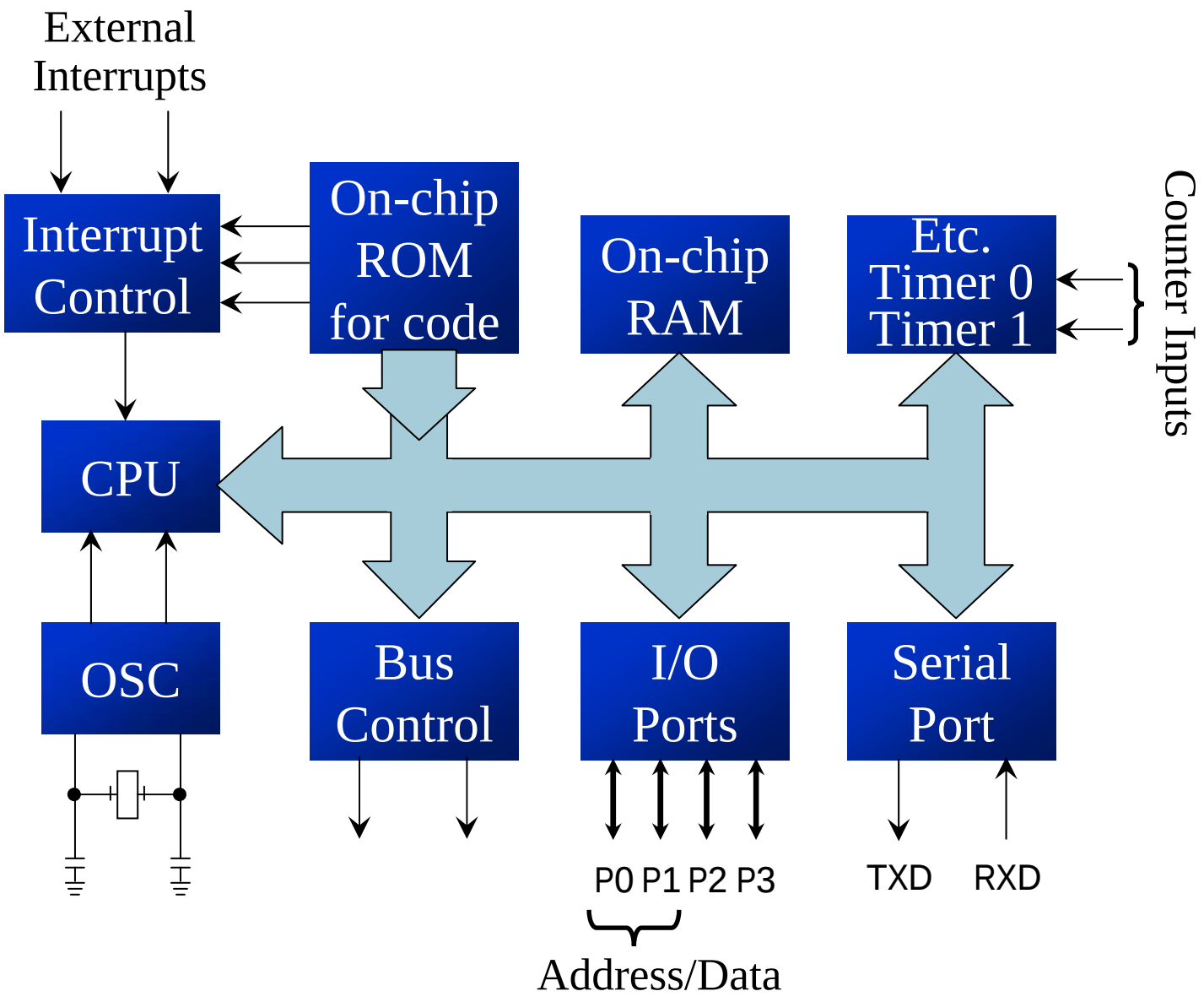
OVERVIEW OF 8051 FAMILY

8051 Microcontroller

- Intel introduced 8051, referred as MCS-51, in 1981
  - The 8051 is an 8-bit processor
    - The CPU can work on only 8 bits of data at a time
  - The 8051 had
    - 128 bytes of RAM
    - 4K bytes of on-chip ROM
    - Two timers
    - One serial port
    - Four I/O ports, each 8 bits wide
    - 6 interrupt sources
- The 8051 became widely popular after allowing other manufactures to make and market any flavor of the 8051, but remaining code-compatible

OVERVIEW OF 8051 FAMILY

8051 Microcontroller (cont')



- The 8051 is a subset of the 8052
- The 8031 is a ROM-less 8051
  - Add external ROM to it
  - You lose two ports, and leave only 2 ports for I/O operations

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

OVERVIEW OF 8051 FAMILY

Various 8051 Microcontrollers

- 8751 microcontroller
  - UV-EPROM
    - PROM burner
    - UV-EPROM eraser takes 20 min to erase
- AT89C51 from *Atmel Corporation*
  - Flash (erase before write)
    - ROM burner that supports flash
    - A separate eraser is not needed
- DS89C4x0 from *Dallas Semiconductor*, now part of *Maxim Corp.*
  - Flash
    - Comes with on-chip loader, loading program to on-chip flash via PC COM port

OVERVIEW OF 8051 FAMILY

Various 8051 Microcontrollers (cont')

- DS5000 from *Dallas Semiconductor*
  - NV-RAM (changed one byte at a time), RTC (real-time clock)
    - Also comes with on-chip loader
- OTP (one-time-programmable) version of 8051
- 8051 family from *Philips*
  - ADC, DAC, extended I/O, and both OTP and flash

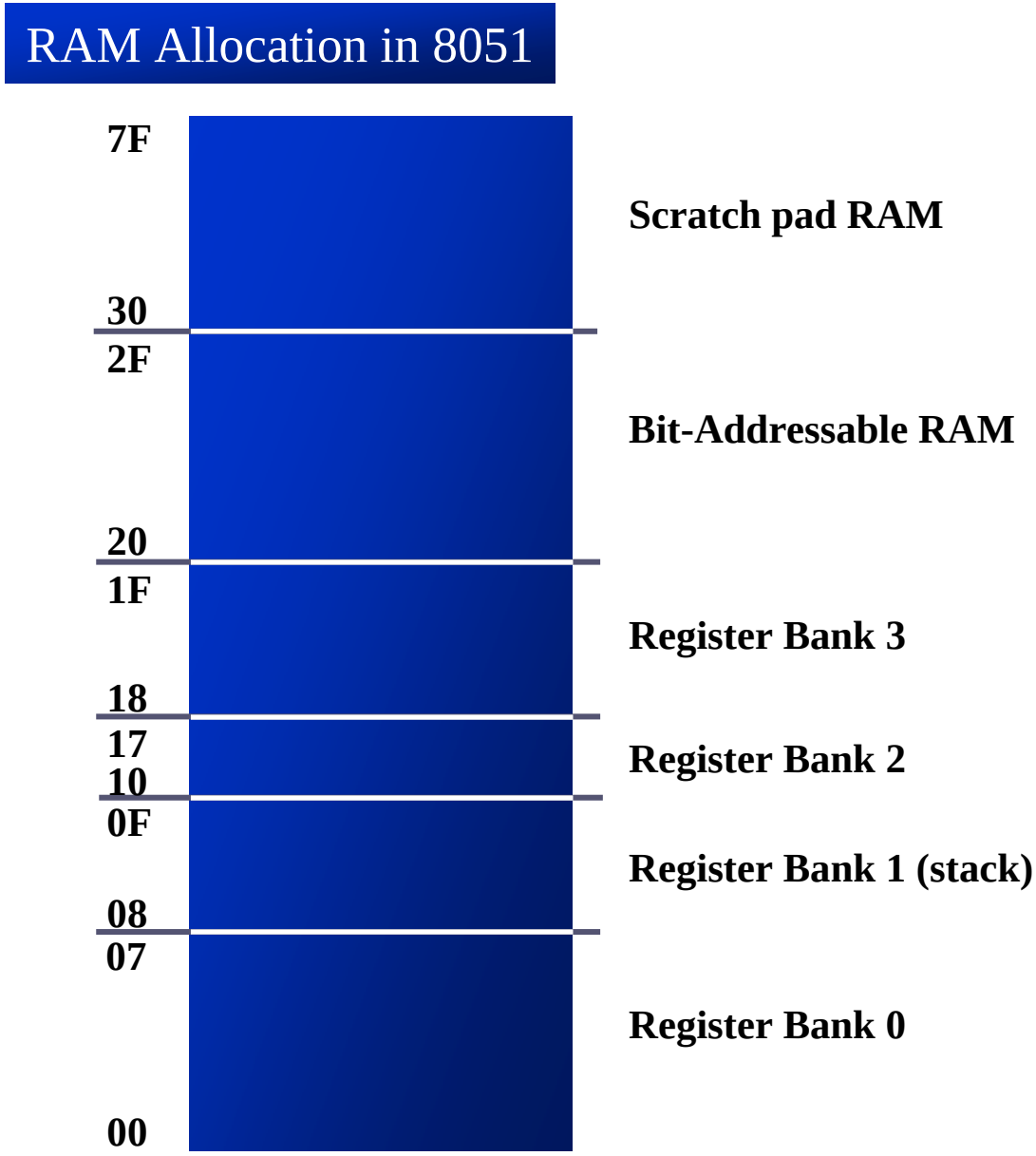
REGISTER BANKS AND STACK

RAM Memory Space Allocation

- There are 128 bytes of RAM in the 8051
  - Assigned addresses 00 to 7FH
- The 128 bytes are divided into three different groups as follows:
  1) A total of 32 bytes from locations 00 to 1F hex are set aside for register banks and the stack
  2) A total of 16 bytes from locations 20H to 2FH are set aside for bit-addressable read/write memory
  3) A total of 80 bytes from locations 30H to 7FH are used for read and write storage, called *scratch pad*

8051 REGISTER BANKS AND STACK

RAM Memory Space Allocation (cont')



8051 REGISTER BANKS AND STACK

Register Banks

- These 32 bytes are divided into 4 banks of registers in which each bank has 8 registers, R0-R7
  - RAM location from 0 to 7 are set aside for bank 0 of R0-R7 where R0 is RAM location 0, R1 is RAM location 1, R2 is RAM location 2, and so on, until memory location 7 which belongs to R7 of bank 0
  - It is much easier to refer to these RAM locations with names such as R0, R1, and so on, than by their memory locations
- Register bank 0 is the default when 8051 is powered up

8051 REGISTER BANKS AND STACK

Register Banks (cont')

Register banks and their RAM address

Bank 0		Bank 1		Bank 2		Bank 3	
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

8051 REGISTER BANKS AND STACK

Register Banks (cont')

- We can switch to other banks by use of the PSW register
  - Bits D4 and D3 of the PSW are used to select the desired register bank
  - Use the bit-addressable instructions SETB and CLR to access PSW.4 and PSW.3

PSW bank selection

	RS1(PSW.4)	RS0(PSW.3)
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

8051 PROGRAMMING IN C

The 8051 Microcontroller and Embedded Systems: Using Assembly and C
Mazidi, Mazidi and McKinlay

Chung-Ping Young
楊中平

Home Automation, Networking, and Entertainment Lab

Dept. of Computer Science and Information Engineering
National Cheng Kung University, TAIWAN

WHY PROGRAM 8051 IN C

- Compilers produce hex files that is downloaded to ROM of microcontroller
  - The size of hex file is the main concern
    - Microcontrollers have limited on-chip ROM
    - Code space for 8051 is limited to 64K bytes
- C programming is less time consuming, but has larger hex file size
- The reasons for writing programs in C
  - It is easier and less time consuming to write in C than Assembly
  - C is easier to modify and update
  - You can use code available in function libraries
  - C code is portable to other microcontroller with little of no modification

DATA TYPES

- A good understanding of C data types for 8051 can help programmers to create smaller hex files
  - Unsigned char
  - Signed char
  - Unsigned int
  - Signed int
  - Sbit (single bit)
  - Bit and sfr

DATA TYPES

Unsigned char

- The character data type is the most natural choice
  - 8051 is an 8-bit microcontroller
- Unsigned char is an 8-bit data type in the range of 0 – 255 (00 – FFH)
  - One of the most widely used data types for the 8051
    - Counter value
    - ASCII characters
- C compilers use the signed char as the default if we do not put the keyword *unsigned*

DATA TYPES

Unsigned char (cont')

Write an 8051 C program to send values 00 – FF to port P1.

Solution:

```
#include <reg51.h>
void main(void)
  {
    unsigned char z;
    for (z=0;z<=255;z++)
       P1=z;
  }
```

1. Pay careful attention to the size of the data
2. Try to use unsigned *char* instead of *int* if possible

Write an 8051 C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to port P1.

Solution:

```
#include <reg51.h>
void main(void)
  {
    unsigned char mynum[]="012345ABCD";
    unsigned char z;
    for (z=0;z<=10;z++)
       P1=mynum[z];
  }
```

DATA TYPES

Unsigned char (cont')

Write an 8051 C program to toggle all the bits of P1 continuously.

Solution:

```
//Toggle P1 forever
#include <reg51.h>
void main(void)
  {
    for (;;)
      {
         p1=0x55;
         p1=0xAA;
       }
  }
```

DATA TYPES

Signed char

- The signed char is an 8-bit data type
  - Use the MSB D7 to represent – or +
  - Give us values from –128 to +127
- We should stick with the unsigned char unless the data needs to be represented as signed numbers
  - temperature

Write an 8051 C program to send values of –4 to +4 to port P1.

Solution:

```
//Singed numbers
#include <reg51.h>
void main(void)
  {
    char mynum[]={+1,-1,+2,-2,+3,-3,+4,-4};
    unsigned char z;
    for (z=0;z<=8;z++)
       P1=mynum[z];
  }
```

DATA TYPES

Unsigned and Signed int

- The unsigned int is a 16-bit data type
  - Takes a value in the range of 0 to 65535 (0000 – FFFFH)
  - Define 16-bit variables such as memory addresses
  - Set counter values of more than 256
  - Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file
- Signed int is a 16-bit data type
  - Use the MSB D15 to represent – or +
  - We have 15 bits for the magnitude of the number from –32768 to +32767

DATA TYPES

Single Bit (cont')

Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

Solution:

```
#include <reg51.h>
sbit MYBIT=P1^0;

void main(void)
  {
    unsigned int z;
    for (z=0;z<=50000;z++)
       {
         MYBIT=0;
         MYBIT=1;
       }
  }
```

sbit keyword allows access to the single bits of the SFR registers

DATA TYPES

Bit and sfr

- The bit data type allows access to single bits of bit-addressable memory spaces 20 – 2FH
- To access the byte-size SFR registers, we use the sfr data type

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
(signed) char	8-bit	-128 to +127
unsigned int	16-bit	0 to 65535
(signed) int	16-bit	-32768 to +32767
sbit	1-bit	SFR bit-addressable only
bit	1-bit	RAM bit-addressable only
sfr	8-bit	RAM addresses 80 – FFH only

TIME DELAY

- There are two way s to create a time delay in 8051 C
  - Using the 8051 timer (Chap. 9)
  - Using a simple for loop

    be mindful of three factors that can affect the accuracy of the delay
    - The 8051 design
      - The number of machine cycle
      - The number of clock periods per machine cycle
    - The crystal frequency connected to the X1 – X2 input pins
    - Compiler choice
      - C compiler converts the C statements and functions to Assembly language instructions
      - Different compilers produce different code

TIME DELAY (cont')

Write an 8051 C program to toggle bits of P1 continuously forever with some delay.

Solution:

```
//Toggle P1 forever with some delay in between
//"on" and "off"
#include <reg51.h>
void main(void)
  {
    unsigned int x;
    for (;;)                    //repeat forever
      {
         p1=0x55;
         for (x=0;x<40000;x++); //delay size
                                //unknown
         p1=0xAA;
         for (x=0;x<40000;x++);
       }
  }
```

We must use the oscilloscope to measure the exact duration

TIME DELAY (cont')

Write an 8051 C program to toggle bits of P1 ports continuously with a 250 ms.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
  {
    while (1)                 //repeat forever
       {
         p1=0x55;
         MSDelay(250);
         p1=0xAA;
         MSDelay(250);
       }
  }

void MSDelay(unsigned int itime)
  {
    unsigned int i,j;
    for (i=0;i<itime;i++)
       for (j=0;j<1275;j++);
  }
```

I/O PROGRAMMING

Byte Size I/O

LEDs are connected to bits P1 and P2. Write an 8051 C program that shows the count from 0 to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

Solution:

```
#include <reg51.h>
#defind LED P2;

void main(void)
  {
    P1=00;              //clear P1
    LED=0;              //clear P2
    for (;;)            //repeat forever
       {
         P1++;          //increment P1
         LED++;         //increment P2
       }
  }
```

Ports P0 – P3 are byte-accessable and we use the P0 – P3 labels as defined in the 8051/52 header file.

I/O PROGRAMMING

Byte Size I/O (cont')

Write an 8051 C program to get a byte of data form P1, wait 1/2 second, and then send it to P2.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);

void main(void)
  {
    unsigned char mybyte;
    P1=0xFF;                  //make P1 input port
    while (1)
       {
         mybyte=P1;           //get a byte from P1
         MSDelay(500);
         P2=mybyte;           //send it to P2
       }
  }
```

I/O PROGRAMMING

Byte Size I/O (cont')

Write an 8051 C program to get a byte of data form P0. If it is less than 100, send it to P1; otherwise, send it to P2.

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsigned char mybyte;
    P0=0xFF;                  //make P0 input port
    while (1)
       {
         mybyte=P0;           //get a byte from P0
         if (mybyte<100)
            P1=mybyte;        //send it to P1
         else
            P2=mybyte;        //send it to P2
       }
  }
```

I/O PROGRAMMING

Bit-addressable I/O

Write an 8051 C program to toggle only bit P2.4 continuously without disturbing the rest of the bits of P2.

Solution:

```
//Toggling an individual bit
#include <reg51.h>
sbit mybit=P2^4;

void main(void)
  {
    while (1)
       {
         mybit=1;            //turn on P2.4
         mybit=0;            //turn off P2.4
       }
  }
```

Ports P0 – P3 are bit-addressable and we use *sbit* data type to access a single bit of P0 - P3

Use the Px^y format, where x is the port 0, 1, 2, or 3 and y is the bit 0 – 7 of that port

I/O PROGRAMMING

Bit-addressable I/O (cont')

Write an 8051 C program to monitor bit P1.5. If it is high, send 55H to P0; otherwise, send AAH to P2.

Solution:

```
#include <reg51.h>
sbit mybit=P1^5;

void main(void)
  {
    mybit=1;                  //make mybit an input
    while (1)
       {
         if (mybit==1)
            P0=0x55;
         else
            P2=0xAA;
       }
  }
```

I/O PROGRAMMING

Bit-addressable I/O (cont')

A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. Write an 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
sbit Dsensor=P1^1;
sbit Buzzer=P1^7;

void main(void)
  {
    Dsensor=1;              //make P1.1 an input
    while (1)
       {
         while (Dsensor==1)//while it opens
           {
              Buzzer=0;
              MSDelay(200);
              Buzzer=1;
              MSDelay(200);
           }
       }
  }
```

I/O PROGRAMMING

Bit-addressable I/O (cont')

The data pins of an LCD are connected to P1. The information is latched into the LCD whenever its Enable pin goes from high to low. Write an 8051 C program to send "The Earth is but One Country" to this LCD.

Solution:

```
#include <reg51.h>
#define LCDData P1  //LCDData declaration
sbit En=P2^0;       //the enable pin

void main(void)
  {
    unsigned char message[]
              ="The Earth is but One Country";
    unsigned char z;
    for (z=0;z<28;z++) //send 28 characters
       {
         LCDData=message[z];
         En=1;    //a high-
         En=0;    //-to-low pulse to latch data
       }
  }
```

I/O PROGRAMMING

Accessing SFR Addresses 80 - FFH

Write an 8051 C program to toggle all the bits of P0, P1, and P2 continuously with a 250 ms delay. Use the sfr keyword to declare the port addresses.

Another way to access the SFR RAM space 80 – FFH is to use the *sfr* data type

Solution:

```
//Accessing Ports as SFRs using sfr data type
sfr P0=0x80;
sfr P1=0x90;
sfr P2=0xA0;
void MSDelay(unsigned int);

void main(void)
  {
    while (1)
       {
         P0=0x55;
         P1=0x55;
         P2=0x55;
         MSDelay(250);
         P0=0xAA;
         P1=0xAA;
         P2=0xAA;
         MSDelay(250);
       }
  }
```

I/O PROGRAMMING

Accessing SFR Addresses 80 - FFH (cont')

Write an 8051 C program to turn bit P1.5 on and off 50,000 times.

Solution:

```
sbit MYBIT=0x95;

void main(void)
  {
    unsigned int z;
    for (z=0;z<50000;z++)
       {
         MYBIT=1;
         MYBIT=0;
       }
  }
```

We can access a single bit of any SFR if we specify the bit address

Notice that there is no `#include <reg51.h>`. This allows us to access any byte of the SFR RAM space 80 – FFH. This is widely used for the new generation of 8051 microcontrollers.

I/O PROGRAMMING

Using bit Data Type for Bit-addressable RAM

Write an 8051 C program to get the status of bit P1.0, save it, and send it to P2.7 continuously.

Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;             //use bit to declare
                        //bit- addressable memory

void main(void)
  {
    while (1)
       {
         membit=inbit;    //get a bit from P1.0
         outbit=membit;   //send it to P2.7
       }
  }
```

We use bit data type to access data in a bit-addressable section of the data RAM space 20 – 2FH

LOGIC OPERATIONS

Bit-wise Operators in C

- Logical operators
  - AND (&&), OR (||), and NOT (!)
- Bit-wise operators
  - AND (&), OR (|), EX-OR (^), Inverter (~), Shift Right (>>), and Shift Left (<<)
    - These operators are widely used in software engineering for embedded systems and control

Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

LOGIC OPERATIONS

Bit-wise Operators in C (cont')

Run the following program on your simulator and examine the results.

Solution:

```
#include <reg51.h>

void main(void)
  {
    P0=0x35 & 0x0F;          //ANDing
    P1=0x04 | 0x68;          //ORing
    P2=0x54 ^ 0x78;          //XORing
    P0=~0x55;                //inversing
    P1=0x9A >> 3;            //shifting right 3
    P2=0x77 >> 4;            //shifting right 4
    P0=0x6 << 4;             //shifting left 4
  }
```

LOGIC OPERATIONS

Bit-wise Operators in C (cont')

Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay. Using the inverting and Ex-OR operators, respectively.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);

void main(void)
  {
    P0=0x55;
    P2=0x55;
    while (1)
       {
         P0=~P0;
         P2=P2^0xFF;
         MSDelay(250);
       }
  }
```

LOGIC OPERATIONS

Bit-wise Operators in C (cont')

Write an 8051 C program to get bit P1.0 and send it to P2.7 after inverting it.

Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;

void main(void)
  {
    while (1)
       {
         membit=inbit;   //get a bit from P1.0
         outbit=~membit; //invert it and send
                          //it to P2.7
       }
  }
```

LOGIC OPERATIONS

Bit-wise Operators in C (cont')

Write an 8051 C program to read the P1.0 and P1.1 bits and issue an ASCII character to P0 according to the following table.

P1.1	P1.0	
0	0	send '0' to P0
0	1	send '1' to P0
1	0	send '2' to P0
1	1	send '3' to P0

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsignbed char z;
    z=P1;
    z=z&0x3;

...
```

LOGIC OPERATIONS

Bit-wise Operators in C (cont')

```
...
switch (z)
        {
          case(0):
            {
               P0='0';
               break;
            }
          case(1):
            {
               P0='1';
               break;
            }
          case(2):
            {
               P0='2';
               break;
            }
          case(3):
            {
               P0='3';
               break;
            }
        }
   }
```

DATA CONVERSION

Packed BCD to ASCII Conversion

Write an 8051 C program to convert packed BCD 0x29 to ASCII and display the bytes on P1 and P2.

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsigned char x,y,z;
    unsigned char mybyte=0x29;
    x=mybyte&0x0F;
    P1=x|0x30;
    y=mybyte&0xF0;
    y=y>>4;
    P2=y|0x30;
  }
```

DATA CONVERSION

ASCII to Packed BCD Conversion

Write an 8051 C program to convert ASCII digits of '4' and '7' to packed BCD and display them on P1.

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsigned char bcdbyte;
    unsigned char w='4';
    unsigned char z='7';
    w=w&0x0F;
    w=w<<4;
    z=z&0x0F;
    bcdbyte=w|z;
    P1=bcdbyte;
  }
```

DATA CONVERSION

Checksum Byte in ROM

Write an 8051 C program to calculate the checksum byte for the data 25H, 62H, 3FH, and 52H.

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsigned char mydata[]={0x25,0x62,0x3F,0x52};
    unsigned char sum=0;
    unsigned char x;
    unsigned char chksumbyte;
    for (x=0;x<4;x++)
       {
         P2=mydata[x];
         sum=sum+mydata[x];
         P1=sum;
       }
    chksumbyte=~sum+1;
    P1=chksumbyte;
  }
```

DATA CONVERSION

Checksum Byte in ROM (cont')

Write an 8051 C program to perform the checksum operation to ensure data integrity. If data is good, send ASCII character 'G' to P0. Otherwise send 'B' to P0.

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsigned char mydata[]
                    ={0x25,0x62,0x3F,0x52,0xE8};
    unsigned char shksum=0;
    unsigned char x;
    for (x=0;x<5;x++)
       chksum=chksum+mydata[x];
    if (chksum==0)
       P0='G';
    else
       P0='B';
}
```

DATA CONVERSION

Binary (hex) to Decimal and ASCII Conversion

Write an 8051 C program to convert 11111101 (FD hex) to decimal and display the digits on P0, P1 and P2.

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsigned char x,binbyte,d1,d2,d3;
    binbyte=0xFD;
    x=binbyte/10;
    d1=binbyte%10;
    d2=x%10;
    d3=x/10;
    P0=d1;
    P1=d2;
    P2=d3;
  }
```

ACCESSING CODE ROM

RAM Data Space Usage by 8051 C Compiler

- The 8051 C compiler allocates RAM locations
  - Bank 0 – addresses 0 – 7
  - Individual variables – addresses 08 and beyond
  - Array elements – addresses right after variables
    - Array elements need contiguous RAM locations and that limits the size of the array due to the fact that we have only 128 bytes of RAM for everything
  - Stack – addresses right after array elements

ACCESSING CODE ROM

RAM Data Space Usage by 8051 C Compiler (cont')

Compile and single-step the following program on your 8051 simulator. Examine the contents of the 128-byte RAM space to locate the ASCII values.

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsigned char mynum[]="ABCDEF"; //RAM space
    unsigned char z;
    for (z=0;z<=6;z++)
       P1=mynum[z];
  }
```

ACCESSING CODE ROM

RAM Data Space Usage by 8051 C Compiler (cont')

Write, compile and single-step the following program on your 8051 simulator. Examine the contents of the code space to locate the values.

Solution:

```
#include <reg51.h>

void main(void)
  {
    unsigned char mydata[100]; //RAM space
    unsigned char x,z=0;
    for (x=0;x<100;x++)
       {
         z--;
         mydata[x]=z;
         P1=z;
       }
  }
```

ACCESSING CODE ROM

8052 RAM Data Space

- One of the new features of the 8052 was an extra 128 bytes of RAM space
  - The extra 128 bytes of RAM helps the 8051/52 C compiler to manage its registers and resources much more effectively
- We compile the C programs for the 8052 microcontroller
  - Use the reg52.h header file
  - Choose the8052 option when compiling the program

ACCESSING CODE ROM (cont')

Compile and single-step the following program on your 8051 simulator. Examine the contents of the code space to locate the ASCII values.

Solution:

```
#include <reg51.h>

void main(void)
  {
    code unsigned char mynum[]="ABCDEF";
    unsigned char z;
    for (z=0;z<=6;z++)
       P1=mynum[z];
  }
```

To make the C compiler use the code space instead of the RAM space, we need to put the keyword code in front of the variable declaration

ACCESSING CODE ROM (cont')

Compare and contrast the following programs and discuss the advantages and disadvantages of each one.

(a)

```
#include <reg51.h>
void main(void)
  {
    P1='H';
    P1='E';
    P1='L';
    P1='L';
    P1='O';
  }

...
```

Short and simple, but the individual characters are embedded into the program and it mixes the code and data together

ACCESSING CODE ROM (cont')

```
...

(b)
#include <reg51.h>
void main(void)
  {
    unsigned char mydata[]="HELLO";
    unsigned char z;
    for (z=0;z<=5;z++)
       P1=mydata[z];
  }

(c)
#include <reg51.h>
void main(void)
  {
    code unsigned char mydata[]="HELLO";
    unsigned char z;
    for (z=0;z<=5;z++)
       P1=mydata[z];
  }
```

Use the RAM data space to store array elements, therefore the size of the array is limited

Use a separate area of the code space for data. This allows the size of the array to be as long as you want if you have the on-chip ROM.

However, the more code space you use for data, the less space is left for your program code

DATA SERIALIZATION

- Serializing data is a way of sending a byte of data one bit at a time through a single pin of microcontroller
  - Using the serial port (Chap. 10)
  - Transfer data one bit a time and control the sequence of data and spaces in between them
    - In many new generations of devices such as LCD, ADC, and ROM the serial versions are becoming popular since they take less space on a PCB

DATA SERIALIZATION (cont')

Write a C program to send out the value 44H serially one bit at a time via P1.0. The LSB should go out first.

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit regALSB=ACC^0;

void main(void)
  {
    unsigned char conbyte=0x44;
    unsigned char x;
    ACC=conbyte;
    for (x=0;x<8;x++)
       {
         P1b0=regALSB;
         ACC=ACC>>1;
       }
  }
```

DATA SERIALIZATION (cont')

Write a C program to send out the value 44H serially one bit at a time via P1.0. The MSB should go out first.

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit regAMSB=ACC^7;

void main(void)
  {
    unsigned char conbyte=0x44;
    unsigned char x;
    ACC=conbyte;
    for (x=0;x<8;x++)
       {
         P1b0=regAMSB;
         ACC=ACC<<1;
       }
  }
```

DATA SERIALIZATION (cont')

Write a C program to bring in a byte of data serially one bit at a time via P1.0. The LSB should come in first.

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit ACCMSB=ACC^7;
bit membit;

void main(void)
  {
    unsigned char x;
    for (x=0;x<8;x++)
       {
         membit=P1b0;
         ACC=ACC>>1;
         ACCMSB=membit;
       }
    P2=ACC;
  }
```

DATA SERIALIZATION (cont')

Write a C program to bring in a byte of data serially one bit at a time via P1.0. The MSB should come in first.

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit regALSB=ACC^0;
bit membit;

void main(void)
  {
    unsigned char x;
    for (x=0;x<8;x++)
       {
         membit=P1b0;
         ACC=ACC<<1;
         regALSB=membit;
       }
    P2=ACC;
  }
```

TIMER PROGRAMMING

The 8051 Microcontroller and Embedded Systems: Using Assembly and C
Mazidi, Mazidi and McKinlay

Chung-Ping Young
楊中平

Home Automation, Networking, and Entertainment Lab

Dept. of Computer Science and Information Engineering
National Cheng Kung University, TAIWAN

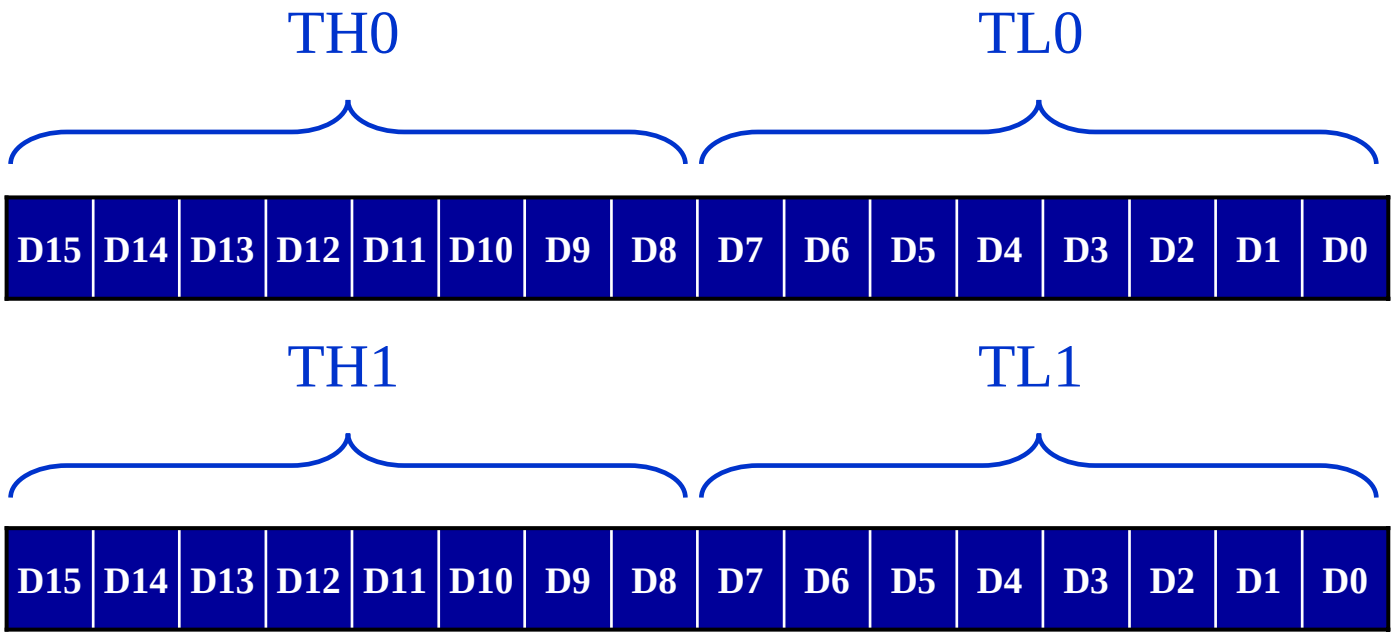
PROGRAMMING TIMERS

- 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

PROGRAMMING TIMERS

Timer 0 & 1 Registers

- Accessed as low byte and high byte
  - The low byte register is called TL0/TL1 and
  - The high byte register is called TH0/TH1
  - Accessed like any other register
    - `MOV TL0,#4FH`
    - `MOV R5,TH0`



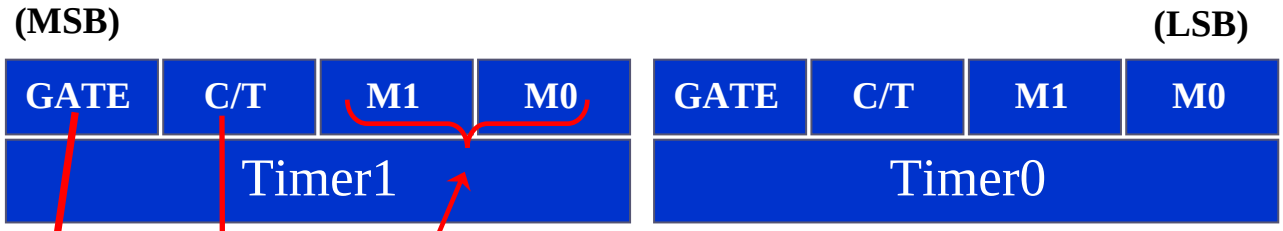
PROGRAMMING TIMERS

TMOD Register

- Both timers 0 and 1 use the same register, called TMOD (timer mode), to set the various timer operation modes
- TMOD is a 8-bit register
  - The lower 4 bits are for Timer 0
  - The upper 4 bits are for Timer 1
  - In each case,
    - The lower 2 bits are used to set the timer mode
    - The upper 2 bits to specify the operation

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**

Gating control when set.
Timer/counter is enable 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

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

PROGRAMMING TIMERS

TMOD Register (cont')

If C/T = 0, it is used as a timer for time delay generation. The clock source for the time delay is the crystal frequency of the 8051

Example 9-1

Indicate which mode and which timer are selected for each of the following.
(a) MOV TMOD, #01H (b) MOV TMOD, #20H (c) MOV TMOD, #12H

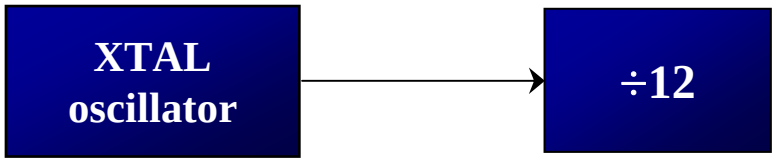
Solution:

We convert the value from hex to binary. From Figure 9-3 we have:
(a) TMOD = 00000001, mode 1 of timer 0 is selected.
(b) TMOD = 00100000, mode 2 of timer 1 is selected.
(c) TMOD = 00010010, mode 2 of timer 0, and mode 1 of timer 1 are selected.

Example 9-2

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

PROGRAMMING TIMERS

TMOD Register

GATE

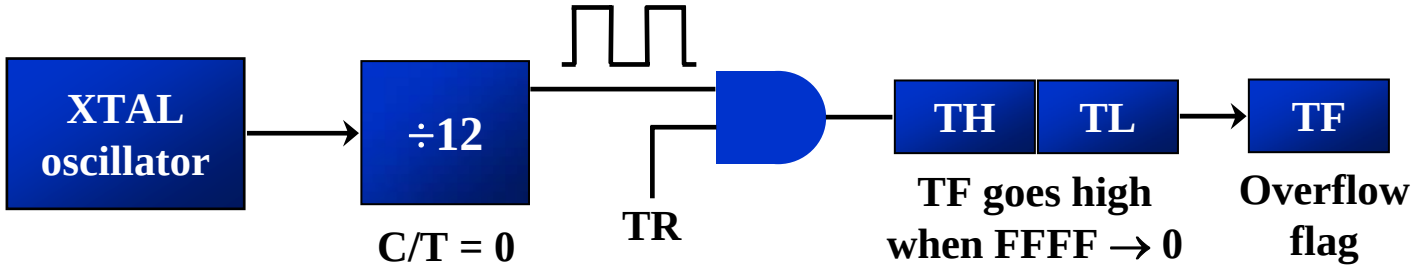
- Timers of 8051 do starting and stopping by either software or hardware control
  - In using software to start and stop the timer where GATE=0
    - The start and stop of the timer are controlled by way of software by the TR (timer start) bits TR0 and TR1
      - The SETB instruction starts it, and it is stopped by the CLR instruction
      - These instructions start and stop the timers as long as GATE=0 in the TMOD register
  - The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register

- Timer 0, mode 2
- C/T = 0 to use XTAL clock source
- gate = 0 to use internal (software) start and stop method.

Find the value for TMOD if we want to program timer 0 in mode 2, use 8051 XTAL for the clock source, and use instructions to start and stop the timer.

TMOD = 0000 0010

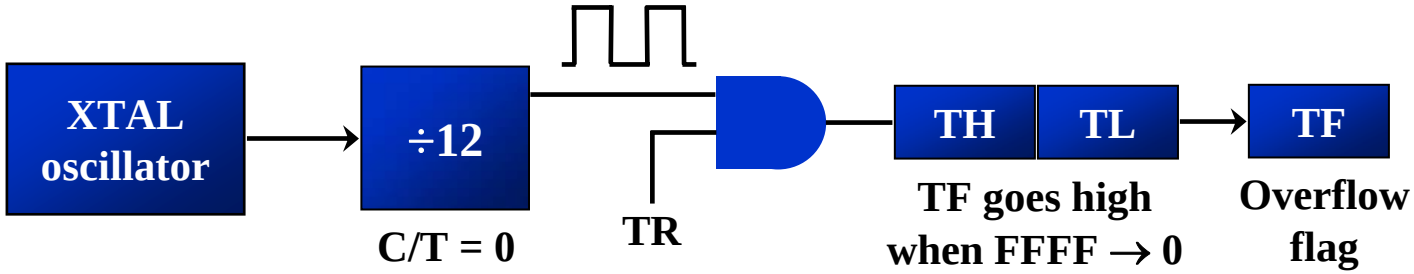
- The following are the characteristics and operations of mode1:
  1. It is a 16-bit timer; therefore, it allows value of 0000 to FFFFH to be loaded into the timer's register TL and TH
  2. After TH and TL are loaded with a 16-bit initial value, the timer must be started
     - This is done by `SETB TR0` for timer 0 and `SETB TR1` for timer 1
  3. After the timer is started, it starts to count up
     - It counts up until it reaches its limit of FFFFH



PROGRAMMING TIMERS

Mode 1 Programming (cont')

3. (cont')
   - 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
     - This timer flag can be monitored
   - When this timer flag is raised, one option would be to stop the timer with the instructions `CLR TR0` or `CLR TR1`, for timer 0 and timer 1, respectively
4. After the timer reaches its limit and rolls over, in order to repeat the process
   - TH and TL must be reloaded with the original value, and
   - TF must be reloaded to 0



- To generate a time delay
  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) with the `JNB TFx,target` instruction to see if it is raised
     - Get out of the loop when TF becomes high
  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

PROGRAMMING TIMERS

Mode 1 Programming

Steps to Mode 1 Program (cont')

Example 9-5

In Example 9-4, calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume XTAL = 11.0592 MHz.

Solution:

The timer works with a clock frequency of 1/12 of the XTAL frequency; therefore, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency. As a result, each clock has a period of T = 1/921.6kHz = 1.085us. In other words, Timer 0 counts up each 1.085 us resulting in delay = number of counts × 1.085us.

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

(a) in hex
(FFFF – YYXX + 1) × 1.085 us, where YYXX are TH, TL initial values respectively. Notice that value YYXX are in hex.

(b) in decimal
Convert YYXX values of the TH, TL register to decimal to get a NNNNN decimal, then (65536 - NNNN) × 1.085 us

Example 9-20

Write an 8051 C program to toggle all the bits of port P1 continuously with some delay in between. Use Timer 0, 16-bit mode to generate the delay.

Solution:

```
#include <reg51.h>
void T0Delay(void);
void main(void){
  while (1) {
    P1=0x55;
    T0Delay();
    P1=0xAA;
    T0Delay();
  }
}
void T0Delay(){
  TMOD=0x01;
  TL0=0x00;
  TH0=0x35;
  TR0=1;
  while (TF0==0);
  TR0=0;
  TF0=0;
}
```

FFFFH – 3500H = CAFFH

= 51967 + 1 = 51968

51968 × 1.085 μs = 56.384 ms is the approximate delay

PROGRAMMING TIMERS IN C

Calculating Delay Length Using Timers

- To speed up the 8051, many recent versions of the 8051 have reduced the number of clocks per machine cycle from 12 to four, or even one
- The frequency for the timer is always 1/12th the frequency of the crystal attached to the 8051, regardless of the 8051 version

PROGRAMMING TIMERS IN C

Times 0/1 Delay Using Mode 1 (16-bit Non Auto-reload)

Example 9-21

Write an 8051 C program to toggle only bit P1.5 continuously every 50 ms. Use Timer 0, mode 1 (16-bit) to create the delay. Test the program on the (a) AT89C51 and (b) DS89C420.

Solution:

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

FFFFH – 4BFDH = B402H

= 46082 + 1 = 46083

46083 × 1.085 μs = 50 ms

PROGRAMMING TIMERS IN C

Times 0/1 Delay Using Mode 1 (16-bit Non Auto-reload) (cont')

Example 9-22

Write an 8051 C program to toggle all bits of P2 continuously every 500 ms. Use Timer 1, mode 1 to create the delay.

Solution:

```
//tested for DS89C420, XTAL = 11.0592 MHz
#include <reg51.h>
void T1M1Delay(void);
void main(void){
  unsigned char x;
  P2=0x55;
  while (1) {
    P2=~P2;
    for (x=0;x<20;x++)
       T1M1Delay();
  }
}
void T1M1Delay(void){
  TMOD=0x10;
  TL1=0xFE;
  TH1=0xA5;
  TR1=1;
  while (TF1==0);
  TR1=0;
  TF1=0;
}
```

A5FEH = 42494 in decimal

65536 – 42494 = 23042

23042 × 1.085 μs = 25 ms and

20 × 25 ms = 500 ms

PROGRAMMING TIMERS IN C

Times 0/1 Delay Using Mode 1 (16-bit Non Auto-reload) (cont')

Example 9-25
A switch is connected to pin P1.2. Write an 8051 C program to monitor SW and create the following frequencies on pin P1.7:
SW=0: 500Hz
SW=1: 750Hz, use Timer 0, mode 1 for both of them.

Solution:

```
#include <reg51.h>
sbit mybit=P1^5;
sbit SW=P1^7;
void T0M1Delay(unsigned char);
void main(void){
  SW=1;
  while (1) {
    mybit=~mybit;
    if (SW==0)
       T0M1Delay(0);
    else
       T0M1Delay(1);
  }
}

.....
```

PROGRAMMING TIMERS IN C

Times 0/1 Delay Using Mode 1 (16-bit Non Auto-reload) (cont')

Example 9-25

```
.....

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

FC67H = 64615

65536 – 64615 = 921

921 × 1.085 μs = 999.285 μs

1 / (999.285 μs × 2) = 500 Hz

Example 9-23

Write an 8051 C program to toggle only pin P1.5 continuously every 250 ms. Use Timer 0, mode 2 (8-bit auto-reload) to create the delay.

Solution:

```
#include <reg51.h>
void T0M2Delay(void);
sbit mybit=P1^5;
void main(void){
  unsigned char x,y;
  while (1) {
    mybit=~mybit;
    for (x=0;x<250;x++)
       for (y=0;y<36;y++) //we put 36, not 40
          T0M2Delay();
  }
}
void T0M2Delay(void){
  TMOD=0x02;
  TH0=-23;
  TR0=1;
  while (TF0==0);
  TR0=0;
  TF0=0;
}
```

Due to overhead of the for loop in C, we put 36 instead of 40

256 – 23 = 233

23 × 1.085 μs = 25 μs and

25 μs × 250 × 40 = 250 ms

PROGRAMMING TIMERS IN C

Times 0/1 Delay Using Mode 2 (8-bit Auto-reload) (cont')

Example 9-24

Write an 8051 C program to create a frequency of 2500 Hz on pin P2.7. Use Timer 1, mode 2 to create delay.

Solution:

```
#include <reg51.h>
void T1M2Delay(void);
sbit mybit=P2^7;
void main(void){
  unsigned char x;
  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