ĐHBK Tp HCM-Khoa Đ-ĐT
BMĐT
GVPT: Hồ Trung Mỹ
Môn học: Vi Xử Lý

CHƯƠNG 5
Lập trình C cho 8051

Overview

- Assembly versus C
- C data types
- Using C in Keil
- 8051 specific extensions
  - SFRs, bit variables
  - I/O programming
  - Timer/Counter
  - Serial Port
  - Create ISRs

C versus Assembly

- ***Advantages***
  - High-level, structured programming language
  - Compiler relieves programmer from some of the hardware details
  - Easier to write large, complex software
  - Programs more readable

- ***Disadvantages***
  - Generally larger machine code
  - Less control and ability to interact with hardware
  - Unclear number of cycles to do something

Square wave, assembly

- **Goal: square wave on P1.0**
  - High level for 500 ticks, low level for 500 ticks

```
; Use Timer0 to create a square wave on P1.0
Start:
        MOV   TMOD, #01h          ; 16-bit mode on timer0
        MOV   TH0,  #0FEh         ; We start off 2^16 - 500
        MOV   TL0,  #0CH
        SETB  TR0                 ; Start timer0
Wait:
        JNB   TF0, Wait           ; Wait for overflow
        CLR   TR0                 ; Stop timer
        CLR   TF0                 ; Clear timer overflow flag
        CPL   P1.0                ; Toggle port bit
        JMP   Start               ; Repeat forever
```

Square wave, C

```
// Use Timer0 to create a 1kHz square
#include <REG52.H>

sbit portbit = P1^0;

void main()
{
    TMOD = 1;                   // 16-bit mode on timer0
    while (1)                   // Loop forever
    {
        TH0 = 0xFE;             // Start timer at 2^16-500
        TL0 = 0x0C;
        TR0 = 1;                // Start timer0
        while (TF0 != 1)        // Wait for overflow
            ;
        TR0 = 0;                // Stop timer
        TF0 = 0;                // Clear timer overflow flag
        portbit = !portbit;     // Toggle port bit
    }
}
```

Gets a bunch of names for SFRs, etc.

Special 8051 SFR bit data type, ^ specifies which bit

no boolean data type!

C data types

Standard data types in 8051 C

Data type	Bytes	Min value	Max value
signed char	1	-128	+127
unsigned char	1	0	255
signed short	2	-32768	+32767
unsigned short	2	0	65535
signed int	2	-32768	+32767
unsigned int	2	0	65535
signed long	4	-2,147,483,648	+2,147,483,647
unsigned long	4	0	4,294,967,295
float	4	±1.175494e-38	±3.402823e+38
double	4	±1.175494e-38	±3.402823e+38

8051 extension types

- 8051 extension types
  - **bit**
    - 8051 bit addressable memory
    - 20h to 2Fh
  - **sbit**
    - A bit inside an SFR (e.g. P1.0)
  - **sfr**
    - Entire byte of an SFR
  - **sfr16**
    - 16-bit SFRs (e.g. DPTR)

- Declaration
  - ***sbit, sfr, and sfr16***
    - *Declare outside of main() program*
    - *Essentially a friendly EQU for an SFR or SFR bit*
  - ***bit***
    - *Declare anywhere a normal variable can be declared*

Data type example

```
#include <REG52.h>

sbit portbit = P1^0;

void main()
{
    signed char     a = 0;
    unsigned char   b = 0;

    signed short    c = 0;
    unsigned short  d = 0;

    signed int      e = 0;
    unsigned int    f = 0;

    signed long     g = 0;
    unsigned long   h = 0;

    float           i = 0.0;
    double          j = 0.0;

    bit             k = 0;

    k = 1;

    unsigned char   l = 0;
    bit             m = 0;
}
```

P1.0

Compile-time error, must declare variables at start of {}-block

#include <REG52.h> (or reg51.h)

- #include inserts text from another file

```
/*--------------------------------------------------------------------------
REG52.H

Header file for generic 80C52 and 80C32 microcontroller.
Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.
All rights reserved.
--------------------------------------------------------------------------*/

#ifndef __REG52_H__
#define __REG52_H__

/*  BYTE Register  */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
...
```

Comments and literals

- Keil accepts C or C++ style comments:

```
// this line will be ignored by the compiler

/* these lines will
   be ignored by the compiler */


unsigned char i;   // this is ignored
unsigned char j;   /* so is this */
```

- C format for decimal/hex/octal:

```
unsigned char i = 100;    // 100 as a base 10 literal
unsigned char j = 0x64;   // 100 in hex, indicated by leading 0x
unsigned char k = 0144;   // 100 in octal, indicated by the leading 0
```

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

Unsigned char – Example

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];
  }
```

Unsigned char – Example (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;
      }
  }
```

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];
  }
```

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

Single Bit

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

Using C in Keil

cuu duong than cong . com

Blinking LEDs example

- Goal: Make LEDs blink every second
- Akempt 1: Big do-nothing loop

```
#include <REG52.h>

void main()
{
   unsigned int i = 0;
   unsigned char lights = 0xFF;
   P0 = lights;
   while (1)
   {
      for (i = 0; i < 30000; i++)
         ;
      lights = ~lights;
      P0 = lights;
   }
}
```

Creating a delay function

```
#include <REG52.h>

void delay(const unsigned int ms)
{
   unsigned int x;
   for (x = 0; x < ms; x++)
   {
      unsigned int y;
      for (y = 0; y <= 113; y++)
         ;
   }
}

void main()
{
   unsigned char lights = 0xFF;
   P0 = lights;
   while (1)
   {
      delay(1000);
      lights = ~lights;
      P0 = lights;
   }
}
```

We promise not to change function parameter

Found by trial and error

"Super Loop" architecture

Function order

- Function order matters in C

```
#include <REG52.h>

void main()
{
   unsigned char lights = 0xFF;
   P0 = lights;
   while (1)
   {
      delay(1000);
      lights = ~lights;
      P0 = lights;
   }
}

void delay(const unsigned int ms)
{
   unsigned int x;
   unsigned int y;

   for (x = 0; x < ms; x++)
   {
      for (y = 0; y <= 113; y++)
         ;
   }
}
```



```
Build target 'Target 1'
compiling DelayFunction.c...
DelayFunction.c(8): warning C206: 'delay': missing function-prototype
DelayFunction.c(8): error C267: 'delay': requires ANSI-style prototype
DelayFunction.c(14): error C231: 'delay': redefinition
DelayFunction.c(23): error C231: 'delay': redefinitionTarget not created
```

Function prototypes

- Declare you will later implement a function

```
#include <REG52.h>

void delay(const unsigned int ms);

void main()
{
   unsigned char lights = 0xFF;
   P0 = lights;
   while (1)
   {
      delay(1000);
      lights = ~lights;
      P0 = lights;
   }
}
void delay(const unsigned int ms)
{
   unsigned int x;
   unsigned int y;

   for (x = 0; x < ms; x++)
   {
      for (y = 0; y <= 113; y++)
         ;
   }
}
```

Parameters & return values

- Functions can:
  - Take parameters as input, empty ()'s if none
  - Return a value as output (or void if none)
  - In C, Compiler automatically maps to registers and/or memory addresses

- Example:
  - Goal: toggle LEDs if any button pushed
    - Create function to counting # of currently down buttons

Button counter

```
sbit BUTTON0 = P2^0;
sbit BUTTON1 = P2^1;
sbit BUTTON2 = P2^2;
sbit BUTTON3 = P2^3;

// Return number of buttons currently pushed
unsigned char getNumButtons()
{
    unsigned char result = 0;
    if (!BUTTON0)
        result++;
    if (!BUTTON1)
        result++;
    if (!BUTTON2)
        result++;
    if (!BUTTON3)
       result++;
    return result;
}
void main()
{
    unsigned char lights = 0xFF;
    P0 = lights;
    while (1)
   {
      if (getNumButtons() > 0)
      {
         while (getNumButtons() != 0)
            ;
         lights = ~lights;
         P0 = lights;
      }
   }
}
```

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.

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
       }
  }
```

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
       }
  }
```

Bit-addressable I/O

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;
                cuu duong than cong . com
void main(void)
  {
    mybit=1;                  //make mybit an input
    while (1)
       {
         if (mybit==1)
            P0=0x55;
         else
            P2=0xAA;
       }
  }
```

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);
           }
       }
  }
```

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
       }
  }
```

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);
       }
  }
```

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.

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	

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
  }
```

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);
       }
  }
```

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
       }
  }
```

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;

...
```

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;
  }
```

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;
  }
```

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

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];
  }
```

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 – Example 1

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 – Example 2

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 – Example 2 (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

Data Serialization

- Serializing data is a way of sending abyte of data one bit at a time through a single pin of microcontroller
  - Using the serial port
  - 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 (con't)

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 (con't)

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 (con't)

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 (con't)

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.

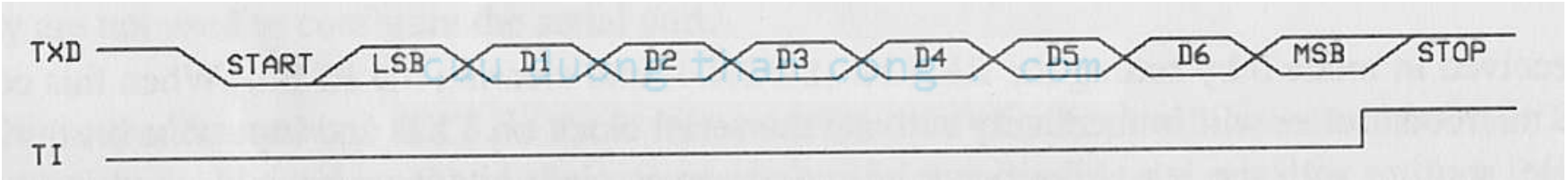
Solution:

```
#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;
  }
```

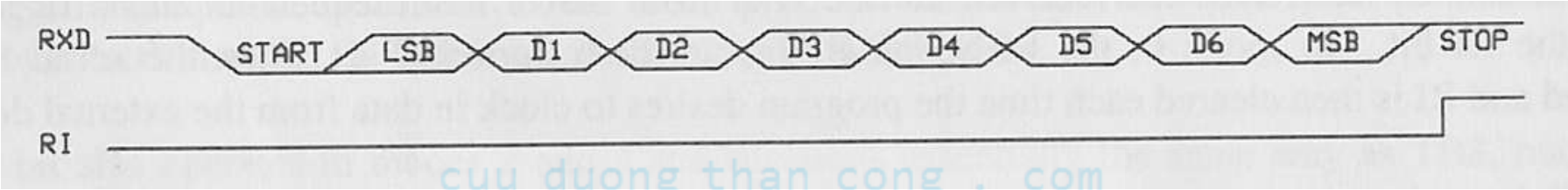
Serial Port – Serial mode 1

- 8-bit UART
  - Most common 8052 serial mode
  - SM0 = 0, SM1 = 1
  - Variable baud rate based on timer overflow
- Transmit:
  - Put a byte in SBUF SFR
  - Ten bits sent = start bit (0) + 8 data bits + stop bit (1)
  - TI bit set when last bit sent



Serial mode 1

- Receive:
  - Recognize incoming start bit on RXD
    - Stop bit (1), start bit (0)
    - Wait for 1-0 transition
  - Ten bits received = start bit (0) + 8 data bits + stop bit (1)
  - RI bit set when last bit received



Serial Mode 1 procedure

- Step 1: Set mode bits in SCON
- Step 2: Start timer1 based on baud rate
- Step 3: Read/write byte
- Step 4: Wait for RI/TI bit to be set

```
SCON = 0x50;   // 8-bit UART, timer1 as baud rate generator
TMOD = 0x20;   // Set timer1 to 8-bit auto-reload
TH1  = 253;    // Reload value of 253 (9600 baud)
TR1  = 1;      // Start timer1

while (1)
{
   while (!RI) {}  // Wait to receive a byte
   ch = SBUF;      // Read character from serial port
   RI = 0;         // Clear the receive flag
   TI = 0;         // Clear the transmit flag
   SBUF = ch;      // Echo the character back to serial port
   while (!TI) {}  // Wait for byte to be transmitted
}
```

Serial Port – Example

Write an 8051 C program 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;   SCON=0x50;
    TH1=0xFA;  //4800 baud rate
    TR1=1;
    while (1) {
          SBUF='A';
          while (TI==0);
          TI=0;
    }
}
```

Programming Timers 0 and 1 in 8051 C

8051 Timers in C

- In 8051 C we can access the timer registers TH, TL, and TMOD directly using the reg51.h header file.
  - See Example Timer-Ex 1
- Timers 0 and 1 delay using mode 1
  - See Example Timer-Ex 2
- Timers 0 and 1 delay using mode 2
  - See Examples 9-23 and 9-24
  - Look by yourself

Timer-Ex 1 (1/2)

Write an 8051 C program to toggle bits of P1 continuously with some time delay. Use Timer 0, 16-bit mode.

Solution:

```
#include <reg51.h>
void T0Delay(void);
void main(void) {
  while(1) {          //repeat forever
    P1=0x55;
    T0Delay();  //time delay
    P1=0xAA;
    T0Delay();     }}
```

Timer-Ex 1 (2/2)

Assume XTML=11.0592MHz for the AT89C51

FFFFH-3500H+1=CB00H=51968

1.085μs ×51968 ≈ 56.384ms

```
void T0Delay() {
  TMOD=0x01; //Timer 0, Mode 1
  TL0=0x00; TH0=0x35; //initial value
  TR0=1;      //turn on T0
  while (TF==0); //text TF to roll over
  TR0=0;      //turn off T0
  TF0=0; }    //clear TF0
```

Timer-Ex 2 (1/3)

A switch is connected to P1.7. Write an 8051 C program to monitor SW and create the following frequencies on P1.5.

SW=0: 500 Hz; SW=1: 750 Hz. Using Timer 0, mode 1.

Assume XTML=11.0592MHz for the AT89C51

Solution:

```
#include <reg51.h>
sbit mybit=P1^5;
sbit SW=P1^7;
void T0Delay(void);
```

Timer-Ex 2 (2/3)

```
void main(void) {
  SW=1;              //make P1.7 an input pin
  while(1) {         //repeat forever
    mybit=~mybit;    //toggle
    if (SW==0)
      T0Delay(0);    //500Hz time delay
    else
      T0Delay(1);    //750Hz time delay
}}
```

Timer-Ex 2 (3/3)

```
void T0Delay(unsigned char c) {
  TMOD=0x01; //Timer 0, Mode 1
  if (c==0) { TL0=0x67; TH0=0xFC };
    // FFFFH-FC67H+1=921, about 999.285μs, 500
  Hz
  else      { TL0=0x9A; TH0=0xFD };
    // FFFFH-FD9AH+1=614, about 666.19μs, 750 Hz
  TR0=1;
  while (TF==0);
  TR0=0;
  TF0=0;
}
```

8051 Counters in C

- External pulses to T0 (P3.4) and T1 (P3.5).
- See following example

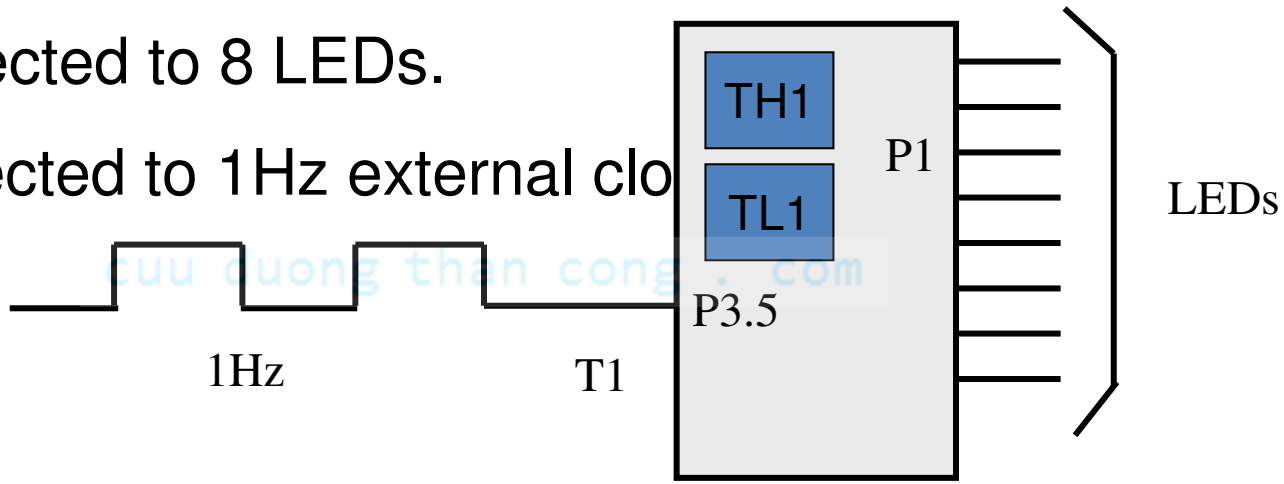
Couter – Example (1/2)

Assume that a 1-Hz external clock is being fed into T1 (P3.5). Write a C program for counter 1 in mode 2 to count up and display the state of the TL1 count on P1. Start the count at 0H.

Solution:

P1 is connected to 8 LEDs.

T1 is connected to 1Hz external clo



Couter – Example (2/2)

```
#include <reg51.h>
sbit T1=P3^5;
void main(void) {
  T1=1;              //make T1 an input pin
  TMOD=0x60;         //counter 1, mode 2
  TH1=0;           //reload value
  while(1) {           //repeat forever
    do {TR1=1; P1=TL1;} while (TF1==0);
    TR1=0; TF1=0;     //clear flags
}}
```

Interrupt functions

Interrupt functions

- Interrupt service routines (ISRs)
  - Special extension to denote ISR function
  - Keyword "interrupt" and a type number

Interrupt number	Description
0	External0
1	Timer0
2	External1
3	Timer1
4	Serial port
5	Timer2

Square wave using ISR

- Goal: Square wave on P1.0 using interrupts

```
// Use Timer0 and interrupt to create a 1kHz square wave on P1.0
#include <REG52.H>

sbit portbit = P1^0;

void main()
{
   TMOD = 0x02;                    // Set timer0 to auto-reload mode
   TH0  = -50;                     // 50 microseconds between overflow
   TL0  = -50;                     // Start timer at beginning value
   TR0  = 1;                       // Start timer0
   IE   = 0x82;                    // Enable timer0 interrupt
   while (1)                       // Loop forver
      ;
}

void timer0ISR() interrupt 1
{
    portbit = !portbit;            // Toggle port bit
}
```

Timer Interrupt (1/3)

Write an 8051 C program that continuously gets a single bit of data form P1.7 and sends it to P1.0. While simultaneously creating a square wave of 200 µs period on pin P2.5.

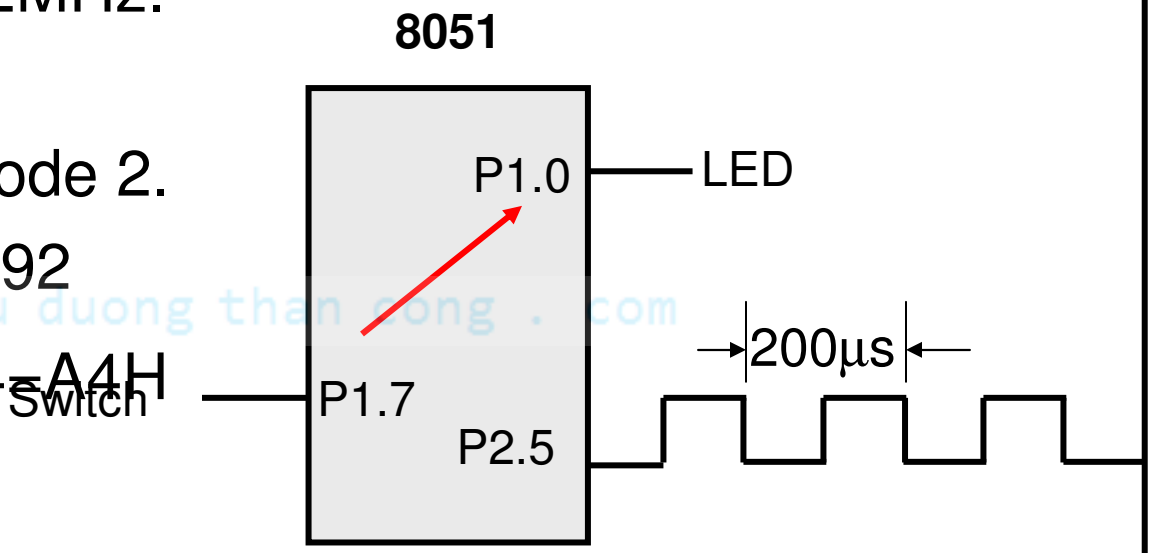
Use timer 0 to create the square wave. Assume XTML=11.0592MHz.

Solution:

Using Timer 0, mode 2.

100µs /1.085µs =92

TH0=256-92=164=A4H



Timer Interrupt (2/3)

```
#include <reg51.h>
sbit SW=P1^7;
sbit IND=P1^0;
sbit WAVE=P2^5;

//Interrup subroutine for creating WAVE
void timer0(void) interrupt 1
{
  WAVE=~WAVE;   //toggle pin
}
```

Timer Interrupt (3/3)

```
void main(void) {
  //setup Timer Interrupt 1
  TMOD=0x02;
  TH0=0xA4; //TH0=-92
  IE=0x82; //enable interrupts for timer
  0
  TR0=1;    //start the timer 0
  //setup SW → IND
  SW=1;     //make P1.7 an input pin
  while(1) {
    IND=SW; //send switch to LED
}}
```

Timer & Serial Interrupts (1/4)

Write a C program that continuously gets a single bit of data from P1.7 and sends it to P1.0 in the main, while simultaneously

as same as Ex 11-14

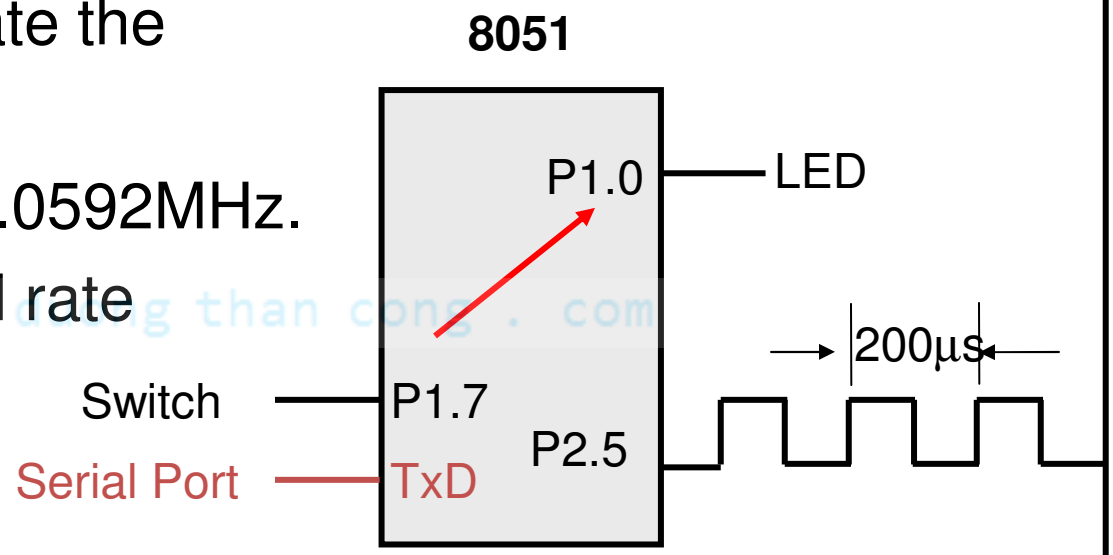
(a) creating a square wave of 200 μs period on pin P2.5.

(b) sending letter 'A' to the serial port.

Use timer 0 to create the square wave.

Assume XTML=11.0592MHz.

Use the 9600 baud rate



Timer & Serial Interrupts (2/4)

Solution:

```
#include <reg51.h>
sbit SW=P1^7;
sbit IND=P1^0;
sbit WAVE=P2^5;

//Interrup subroutine for creating WAVE
void timer0(void) interrupt 1
{
  WAVE=~WAVE;   //toggle pin
}
```

Timer & Serial Interrupts (3/4)

```
//Interrup subroutine for sending 'A'
void serial0() interrupt 4
{
  if (TI==1) {
    SUBF = 'A'; //send 'A'
    TI=0;       //clear interrupt
  }
  else {
    RI=0;       //clear interrupt
  }
}
```

Timer & Serial Interrupts (4/4)

```
void main(void) {
  SW=1;       //make P1.7 an input pin
  TH1= -3;    //9600 baud rate
  TMOD=0x22;//mode 2 for both timers
  TH0=0xA4; //TH0=-92
  SCON=0x50;
  TR0=1; TR1=1;
  IE=0x92;    //enable interrupts for timer
  0
  while(1) {
    IND=SW; //send switch to LED
}}
```