C Programming in 8051

Instructor

Zhizheng Wu

吴智政

Sino European school of Technology of Shanghai University

- Review of C basics
- C data type for 8051
- I/O programming in 8051 C
- Logic operation in 8051 C
- Data serialization in 8051 C
- Create a time delay in 8051 C

REVIEW OF C BASICS

- Of higher level languages, C is the closest to assembly language
  - bit manipulation instructions
  - pointers (indirect addressing)
- Most microcontrollers have available C compilers
- Writing in C simplifies code development for large projects.

REVIEW OF C BASICS

- 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

a) It is easier and less time consuming to write in C than Assembly

b) C is easier to modify and update

c) You can use code available in function libraries

d) C code is portable to other microcontroller with little of no modification

- Available C Compilers
  - Keil – integrated with the IDE we have been using for labs.
  - Reads51 – available on web site (http://www.rigelcorp.com/reads51.htm)
  - Freeware: SDCC - Small Device C Compiler (http://sdcc.sourceforge.net/)
  - Other freeware versions …

- Basic C Program Structure

1. Compiler directives and include files
2. Declarations of global variables and constants
3. Declaration of functions
4. Main function
5. Sub-functions

Example: Example_C.pdf

- Basic C Program Loop Statement
  - While loop:

    while (condition) { statements }

    while condition is true, execute statements

    if there is only one statement, we can lose the {}

    Example: while (1) ; // loop forever

- Basic C Program Loop Statement
  - For statement:

    for (initialization; condition; increment) {statements}

    - *initialization* done before statement is executed

    - *condition* is tested, if true, execute *statements*
    - do *increment* step and go back and test condition again

    - repeat last two steps until condition is not true

- Basic C Program Loop Statement

Examle:

```
for (n = 0; n<1000; n++)
```

n++ means n = n + 1

Be careful with signed integers!

```
for (i=0; i < 33000; i++) {LED = ~LED};
```

Why is this an infinite loop?

- Basic C Program Loop Statement

  - Do – While Loop

```
do
        statements
while (expression);
```

Test made at the bottom of the loop

- Basic C Program Decision Statement
  - Decision – if statement

```
if (condition1)
    {statements_1}
else if (condition2)
    {statements_2}
…
else
{statements_n}
```

- Basic C Program Decision Statement
  - Decision – switch statement

```
switch (expression) {
        case const-expr: statements
        case const-expr: statements
        default: statements
}
```

DATA TYPE

• All variables must be declared at top of program, before the first statement.

Data Type	Size in Bits	Data Range/Usage
unsiged char	8-bit	0 to 255
(signed) char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65535
(signd) int	16-bit	–32,768 to +32,767
sbit	1-bit	SFR bit-addressable only
bit	1-bit	RAM bit-addressable only
sfr	8-bit	RAM addresses 80 - FFH only

- **unsigned char** and **signed char**

  - 8-bit data type.

  - Need to explicitly include the key word *unsigned* in front of *char* to indicate an unsigned data type.

  - Default char type is *signed char*. Therefore, there is no need to explicitly include the key work *signed*.

  Example:

```
#include <reg51.h>
void main(void)
{
unsigned char x;
signed char y;
…
```

- **unsigned int** and **signed int**

  - 16-bit data type.
  - Need to explicitly include the key word *unsigned* in front of *int* to indicate an unsigned data type.
  - Default *int* type is *signed int*. Therefore, there is no need to explicitly include the key work *signed*.

  Example:

```
#include <reg51.h>
void main(void)
{
unsigned int x;
signed int y;
…
```

- **sbit (single bit)** and **bit**

  - *sbit* is used to access single-bit addressable registers, such as the single-bit addressable SFR registers which include the SFRs for ports P0, P1, P2, and P3.
  - *bit* is used to access single bits in the RAM bit-addressable memory space 20H – 2FH.

  Example:

```
#include <reg51.h>
sbit  BIT0 = P1^0;
bit  BIT1;
void main(void)
{
…
```

- **sfr**

  - sfr data type is used to access byte-size SFR registers.

  Example:

```
//Using sfr data type
sfr P0 = 0x80          // Declaring P0 using sfr type
void main(void)
{
…
P0++;                  // increment P0
…
```

DATA TYPE

Example:

```
#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++)
        }
```

DATA TYPE

- 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

Example:

```
#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;
        }
}
```

- **Byte size I/O**

➢ **Use P0 – P3 labels as defined in the header file to access the ports P0-P3.**

Example:

```
#include <reg51.h>
#define LED P2
bit  bit1;
void main(void)
{
…
LED++;
…
```

- **Bit-addressable I/O programming**

➢ **To access bit y in port x, use Px^y to refer to that particular bit.**

Example:

```
#include <reg51.h>
sbit  bit1 = P0.5
void main(void)
{
…
bit1=1;
…
```

- **Bit-addressable I/O programming**

➢ **To access bit y in port x, use Px^y to refer to that particular bit.**

Example:

```
#include <reg51.h>
sbit  bit1 = P0.5
void main(void)
{
…
bit1=1;
…
```

- **Bit-wise operator:** The AND, OR, XOR, and inverting operations can be performed on bits as well as on 8-bit data types.

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

- **Logical operators**

  AND (&&), OR (||), and NOT (!)

While (P1.0 && P1.1)

……

• There are two bit-wise shift operations performed using shift right (>>) and shift left (<<) instructions.

Examples:

```
#include <reg51.h>
sbit  bit1 = P0.5
void main(void)
{
…
P0 = 0x21 & 0xA2;
P1 = P1^0xFF;
P2 = 0x32>>2;        //shifting right twice
…
```

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
  - ➢ 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

Example: 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;
}
}
```

TIME DELAY

- There are two ways to create a time delay in 8051 C
  - Using the 8051 timer
  - Using a simple *for* loop
- be mindful of three factors that can affect the accuracy of the delay
  - ① The number of machine cycle and the number of clock periods per machine cycle
  - ② The crystal frequency connected to the X1 – X2 input pins
  - ③ Compiler choice
    - C compiler converts the C statements and functions to Assembly language instructions
    - Different compilers produce different code

- 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++);
}
}
```

C for Large Projects

- Use functions to make programs modular
- Break project into separate files if the programs get too large
- Use header (#include) files to hold definitions used by several programs
- Keep main program short and easy to follow
- Consider multi-tasking or multi-threaded implementations

- Multitasking: Perception of multiple tasks being executed simultaneously.
  - Usually a feature of an operating system and tasks are separate applications.
  - Embedded systems are usually dedicated to one application.
- Multithreading: Perception of multiple tasks within a single application being executed.
  - Example: Create square wave on P1.0 while echoing characters you type.