CHAPTER 19

8051 PROGRAMMING IN C

OBJECTIVES

In this chapter, you will learn

- Examine the C data type for the 8051.
- Code 8051 C programs for time delay and I/O operations.
- Code 8051 C programs for I/O bit manipulation.
- Code 8051 C programs for logic and arithmetic operations.
- Code 8051 C programs for ASCII and BCD data conversion.
- Code 8051 C programs for binary (hex) to decimal conversion.
- Code 8051 C programs to use the 8051 code space.
- Code 8051 C programs for data serialization.

Why program the 8051 in C?

Compilers produce hex files that we download into the ROM of the microcontroller. The size of the hex file produced by the compiler is one of the main concerns of microcontroller programmers, for two reasons:

1. Microcontrollers have limited on-chip ROM.
2. The code space for the 8051 is limited to 64K bytes.

How does the choice of programming language affect the compiled program size? While Assembly language produces a hex file that is much smaller than C, programming in Assembly language is tedious and time consuming. C programming, on the other hand, is less time consuming and much easier to write, but the hex file size produced is much larger than if we used Assembly language. The following are some of the major reasons for writing programs in C instead of Assembly:

1. It is easier and less time consuming to write in C than Assembly.
2. C is easier to modify and update.
3. You can use code available in function libraries.
4. C code is portable to other microcontrollers with little or no modification.

The study of C programming for the 8051 is the main topic of this chapter. In Section 19-1, we discuss data types and time delays. I/O programming is shown in Section 19-2. The logic operations AND, OR, XOR, inverter, and shift are discussed in Section 19-3. Section 19-4 describes ASCII and BCD conversions and checksums. In Section 19-5 we show how 8051 C compilers use the program (code) ROM space for data. Finally, in Section 19-6 data serialization for 8051 is shown.

SECTION 19.1: DATA TYPES AND TIME DELAY IN 8051 C

In this section we first discuss C data types for the 8051 and then provide code for time delay functions.

19.1.1: C data types for the 8051

Since one of the goals of 8051 C programmers is to create smaller hex files, it is worthwhile to re-examine C data types for 8051 C. In other words, a good understanding of C data types for the 8051 can help programmers to create smaller hex files. In this section we focus on the specific C data types that are most useful and widely used for the 8051 microcontroller.

19.1.2: Unsigned char

Since the 8051 is an 8-bit microcontroller, the character data type is the most natural choice for many applications. The unsigned char is an 8-bit data type that takes a value in the range of 0 - 255 (00 - FFH). It is one of the most widely used data types for the 8051. In many situations, such as setting a counter value, where there is no need for signed data we should use the unsigned char instead of the signed char. Remember that C compilers use the signed char as the default if we do not put the keyword *unsigned* in front of the char (see Example 19-1). We can also use the unsigned char data type for a string of ASCII characters, including extended ASCII characters. Example 19-2 shows a string of ASCII characters. See Example 19-3 for toggling ports.

In declaring variables, we must pay careful attention to the size of the data and try to use unsigned char instead of int if possible. Because the 8051 has a limited number of registers and data RAM locations, using the int in place of the char data type can lead to a larger size hex file. Such a misuse of the data types in compilers such as Microsoft Visual C++ for x86 IBM PCs is not a significant issue.

Example 19-1

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;

  }
```

Run the above program on your simulator to see how P1 displays values 00 - FFH in binary.

Example 19-2

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];
  }
```

Run the above program on your simulator to see how P1 displays values 30H, 31H, 32H, 33H, 34H, 35H, 41H, 42H, 43H, and 44H, the hex values for ASCII 0, 1, 2, and so on.

Example 19-3

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

Solution:

```
// Toggle P1 forever
#include <reg51.h>
void main(void)
  {
    for(;;)          //repeat forever
      {
        P1=0x55;     //0x indicates the data is in hex (binary)
        P1=0xAA;
      }
  }
```

Run the above program on your simulator to see how P1 toggles continuously. Examine the asm code generated by the C compiler.

19.1.3: Signed char

The signed char is an 8-bit data type that uses the most significant bit (D7 of D7 - D0) to represent the – or + value. As a result, we have only 7 bits for the magnitude of the signed number, giving us values from –128 to +127. In situations where + and – are needed to represent a given quantity such as temperature, the use of the signed char data type is a must.

Again notice that if we do not use the keyword *unsigned*, the default is the signed value. For that reason we should stick with the unsigned char unless the data needs to be represented as signed numbers.

19.1.4: Unsigned int

The unsigned int is a 16-bit data type that takes a value in the range of 0 to 65535 (0000 - FFFFH). In the 8051, unsigned int is used to define 16-bit variables such as memory addresses. It is also used to set counter values of more than 256. Since the 8051 is an 8-bit microcontroller and the int data type takes two bytes of RAM, we must not use the int data type unless we have to. Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file. Such misuse is not a big deal in PCs with 256 megabytes of memory, 32-bit Pentium registers and memory accesses, and a bus speed of 133 MHz. However, for 8051 programming do not use unsigned int in places where unsigned char will do the job. Of course the compiler will not generate an error for this misuse, but the overhead in hex file size is noticeable. Also in situations where there is no need for signed data (such as setting counter values), we should use unsigned int instead of signed int. This gives a much wider range for data declaration. Again, remember that the C compiler uses signed int as the default if we do not use the keyword *unsigned*.

19.1.5: Signed int

Signed int is a 16-bit data type that uses the most significant bit (D15 of D15 - D0) to represent the – or + value. As a result, we have only 15 bits for the magnitude of the number, or values from –32,768 to +32,767.

19.1.6: Sbit (single bit)

The sbit keyword is a widely used 8051 C data type designed specifically to access single-bit addressable registers. It allows access to the single bits of the SFR registers. As we saw in Chapter 17, some of the SFRs are bit-addressable. Among the SFRs that are widely used and are also bit-addressable are ports P0 - P3. We can use sbit to access the individual bits of the ports as shown in Example 19-5.

Example 19-4

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

Solution:

```
//sign 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];
  }
```

Run the above program on your simulator to see how P1 displays values of 1, FFH, 2, FEH, 3, FDH, 4, and FCH, the hex values for +1, –1, +2, –2, and so on.

Example 19-5

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;      //notice that sbit is
                        //declared outside of main

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

Run the above program on your simulator to see how P1.0 toggles continuously.

19.1.7: Bit and sfr

The bit data type allows access to single bits of bit-addressable memory spaces 20 - 2FH. Notice that while the sbit data type is used for bit-addressable SFRs, the bit data type is used for the bit-addressable section of RAM space 20 - 2FH. To access the byte-size SFR registers, we use the sfr data type. We will see the use of sbit, bit, and sfr data types in Table 19-1.

19.1.8: Time delay

There are two ways to create a time delay in 8051 C:

1. Using a simple for loop
2. Using the 8051 timers

In either case, when we write a time delay we must use the oscilloscope to measure the duration of our time delay. Next, we use the for loop to create time delays. Discussion of the use of the 8051 timer to create time delays is postponed until Chapter 20.

Table 19-1: Some Widely Used Data Types for 8051 C

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

In creating a time delay using a for loop, we must be mindful of three factors that can affect the accuracy of the delay.

1. The 8051 design. Since the original 8051 was designed in 1980, both the fields of IC technology and microprocessor architectural design have seen great advancements. As we saw in Chapter 15, the number of machine cycles and the number of clock periods per machine cycle vary among different versions of the 8051/52 microcontroller. While the original 8051/52 design used 12 clock periods per machine cycle, many of the newer generations of the 8051 use fewer clocks per machine cycle. For example, the DS5000 uses 4 clock periods per machine cycle, while the DS89C420 uses only one clock per machine cycle.
2. The crystal frequency connected to the X1 - X2 input pins. The duration of the clock period for the machine cycle is a function of this crystal frequency.
3. Compiler choice. The third factor that affects the time delay is the compiler used to compile the C program. When we program in Assembly language, we can control the exact instructions and their sequences used in the delay subroutine. In the case of C programs, it is the C compiler that converts the C statements and functions to Assembly language instructions. As a result, different compilers produce different code. In other words, if we compile a given 8051 C programs with different compilers, each compiler produces different hex code.

For the above reasons, when we write time delays for C, we must use the oscilloscope to measure the exact duration. Look at Examples 19-6 through 19-8.

Example 19-6

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++);
      }
  }
```

Example 19-7

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

Solution:

The program below is tested for the DS89C420 with XTAL = 11.0592 MHz.

```
#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++);
}
```

Run the above program on your Trainer and use the oscilloscope to measure the delay.

Example 19-8

Write a 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay.

Solution:

```
//This program is tested for the DS89C420 with XTAL = 11.0592 MHz
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
  while(1) //another way to do it forever
    {
      P0=0x55;
      P2=0x55;
      MSDelay(250);
      P0=0xAA;
      P2=0xAA;
      MSDelay(250);
    }
}
void MSDelay(unsigned int itime)
{
  unsigned int i, j;
  for(i=0;i<itime;i++)
    for(j=0;j<1275;j++);
}
```

19.1.9: Review questions

1. Give the magnitude of the unsigned char and signed char data types.
2. Give the magnitude of the unsigned int and signed int data types.
3. If we are declaring a variable for a person's age, we should use the ___ data type.
4. True or false. Using a for loop to create a time delay is not recommended if you want your code be portable to other 8051 versions.
5. Give three factors that can affect the delay size.

SECTION 19.2: I/O PROGRAMMING IN 8051 C

In this section we look at C programming of the I/O ports for the 8051. We look at both byte and bit I/O programming.

19.2.1: Byte size I/O

As we stated in Chapter 16, ports P0 - P3 are byte-accessible. We use the P0 - P3 labels as defined in the 8051/52 C header file. See Example 19-9. Examine the next few examples to get a better understanding of how ports are accessed in 8051 C.

Example 19-9

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>
#define LED P2          //notice how we can define P2
void main(void)
  {
    P1=00;              //clear P1
    LED=0;              //clear P2
    for(;;)             //repeat forever
      {
        P1++;           //increment P1
        LED++;          //increment P2
      }
  }
```

Example 19-10

Write an 8051 C program to get a byte of data from 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 an input port
    while(1)
      {
        mybyte=P1;          //get a byte from P1
        MSDelay(500);
        P2=mybyte;          //send it to P2
      }
  }
void MSDelay(unsigned int itime)
  {
    unsigned int i, j;
    for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);
  }
```

Example 19-11

Write an 8051 C program to get a byte of data from 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 an input port
  while(1)
    {
      mybyte=P0;              //get a byte from P0
      if(mybyte<100)
        P1=mybyte;            //send it to P1 if less than 100
      else
        P2=mybyte;            //send it to P2 if more than 100
    }
}
```

19.2.2: Bit-addressable I/O programming

The I/O ports of P0 - P3 are bit-addressable. We can access a single bit without disturbing the rest of the port. We use the sbit data type to access a single bit of P0 - P3. One way to do that is to use the Px^y format where *x* is the port 1, 2, or 3, and *y* is the bit 0 - 7 of that port. For example, P1^7 indicates P1.7. When using this method, you need to include the reg51.h file. Study the next few examples to become familiar with the syntax.

Example 19-12

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;      //notice the way single bit is declared
void main(void)
{
  while(1)
    {
      mybit=1;          //turn on P2.4
      mybit=0;          //turn off P2.4
    }
}
```

Example 19-13

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;      //notice the way single bit is declared
void main(void)
  {
    mybit=1;            //make mybit an input
    while(1)
      {
        if(mybit==1)
          P0=0x55;
        else
          P2=0xAA;
      }
  }
```

Example 19-14

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;  //notice the way single bit is defined
sbit Buzzer = P1^7;
void main(void)
  {
    Dsensor=1;          //make P1.1 an input
    while(Dsensor==1)
      {
        buzzer=0;
        MSDelay(200);
        buzzer=1;
        MSDelay(200);
      }
  }
void MSDelay(unsigned int itime)
  {
    unsigned int i, j;
    for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);
  }
```

Example 19-15

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 all the 28 characters
      {
        LCDData=message[z];
        En=1;                   //a high-
        En=0;                   //-to-low pulse to latch the LCD data
      }
  }
```

Run the above program on your simulator to see how P1 displays each character of the message. Meanwhile, monitor bit P2.0 after each character is issued.

19.2.3: Accesssing SFR addresses 80 - FFH

Another way to access the SFR RAM space 80 - FFH is to use the sfr data type. This is shown in Example 19-16. We ca also access a single bit of any SFR if we specify the bit address as shown in Example 19-17. Both the bit and byte addresse for the P0 - P3 ports are given in Table 19-2. Notice in Examples 19-16 and 19-17, that there is no #include <reg51.h> state ment. This allows us to access any byte of the SFR RAM space 80 - FFH. This is a method widely used for the new genera tion of 8051 microcontrollers, and we will use it in future chapters.

Table 19-2: Single Bit Addresses of Ports

P0	Addr	P1	Addr	P2	Addr	P3	Addr	Port's Bit
P0.0	80H	P1.0	90H	P2.0	A0H	P3.0	B0H	D0
P0.1	81H	P1.1	91H	P2.1	A1H	P3.1	B1H	D1
P0.2	82H	P1.2	92H	P2.2	A2H	P3.2	B2H	D2
P0.3	83H	P1.3	93H	P2.3	A3H	P3.3	B3H	D3
P0.4	84H	P1.4	94H	P2.4	A4H	P3.4	B4H	D4
P0.5	85H	P1.5	95H	P2.5	A5H	P3.5	B5H	D5
P0.6	86H	P1.6	96H	P2.6	A6H	P3.6	B6H	D6
P0.7	87H	P1.7	97H	P2.7	A7H	P3.7	B7H	D7

Example 19-16

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.

Solution:

```
// Accessing Ports as SFRs using the sfr data type
sfr P0 = 0x80;              //declaring P0 using sfr data type
sfr P1 = 0x90;
sfr P2 = 0xA0;
void MSDelay(unsigned int);
void main(void)
  {
    while(1)                //do it forever
      {
        P0=0x55;
        P1=0x55;
        P2=0x55;
        MSDelay(250);       //250 ms delay
        P0=0xAA;
        P1=0xAA;
        P2=0xAA;
        MSDelay(250);
      }
  }

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

Example 19-17

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

Solution:

```
sbit MYBIT = 0x95;   //another way to declare bit P1^5
void main(void)
  {
    unsigned int z;
    for(z=0;z<50000;z++)
      {
        MYBIT=1;
        MYBIT=0;
      }
  }
```

19.2.4: Using bit data type for bit-addressable RAM

The sbit data type is used for bit-addressable SFR registers only. Sometimes we need to store some data in a bit-addressable section of the data RAM space 20 - 2FH. To do that, we use the bit data type, as shown in Example 19-18.

Example 19-18

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;          //sbit is used to declare SFR bits
bit membit;                  //notice we use bit to declare
                             //bit-addressable memory
void main(void)
  {
    while(1)
      {
        membit=inbit;        //get a bit from P1.0
        outbit=membit;       //and send it to P2.7
      }
  }
```

19.2.5: Review questions

1. The address of P1 is _____.
2. Write a short program that toggles all bits of P2.
3. Write a short program that toggles only bit P1.0.
4. True or false. The sbit data type is used for both SFR and RAM single-bit addressable locations.
5. True or false. The bit data type is used only for RAM single-bit addressable locations.

SECTION 19.3: LOGIC OPERATIONS IN 8051 C

One of the most important and powerful features of the C language is its ability to perform bit manipulation. Because many books on C do not cover this important topic, it is appropriate to discuss it in this section. This section describes the action of bit-wise logic operators and provides some examples of how they are used.

19.3.1: Bit-wise operators in C

While every C programmer is familiar with the logical operators AND (&&), OR (||), and NOT (!), many C programmers are less familiar with the bitwise operators AND (&), OR (|), EX-OR (^), Inverter (~), Shift Right (>>), and Shift Left (<<). These bit-wise operators are widely used in software engineering for embedded systems and control; consequently, understanding and mastery of them are critical in microprocessor-based system design and interfacing. See Table 19-3.

Table 19-3: Bit-wise Logic Operators for C

		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	

The following shows some examples using the C logical operators.

1. 0x35 & 0x0F = 0x05 /* ANDing */
2. 0x04 | 0x68 = 0x6C /* ORing: */
3. 0x54 ^ 0x78 = 0x2C /* XORing */
4. ~0x55 = 0xAA /* Inverting 55H */

Examples 19-19 and 19-20 show the usage bit-wise operators.

Example 19-19

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 times
    P2= 0x77 >> 4;       //shifting right 4 times
    P0= 0x6 << 4;        //shifting left 4 times
  }
```

Example 19-20

Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay. Use the inverting operator.

Solution:

The program below is tested for the DS89C420 with XTAL = 11.0592 MHz.

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
  {
    P0=0x55;
    P2=0x55;
    while(1)
      {
        P0=~P0;
        P2=~P2;
        MSDelay(250);
      }
  }
void MSDelay(unsigned int itime)
  {
    unsigned int i, j;
    for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);
  }
```

19.3.2: Bit-wise shift operation in C

There are two bit-wise shift operators in C: (1) shift right (>>), and (2) shift left (<<).

Their format in C is as follows:

data >> number of bits to be shifted right
data << number of bits to be shifted left

The following shows some examples of shift operators in C.

```
0x9A >> 3= 0x13     /* shifting right 3 times */
0x77 >> 4 = 0x07    /* shifting right 4 times */
0x6 << 4 = 0x60     /* shifting left 4 times */
```

Study Examples 19-21, 19-22, and 19-23, showing how the bit-wise operators are used in the 8051 C.

Example 19-21

Write an 8051 C program to toggle all the bits of P0, P1, and P2 continuously with a 250 ms delay. Use the Ex-OR operator.

Solution:

The program below is tested for the DS89C420 with XTAL = 11.0592 MHz.

```
#includee <reg51.h>
void MSDelay(unsigned int);
void main(void)
  {
    P0=0x55;
    P1=0x55;
    P2=0x55;
    while(1)
      {
        P0=P0^0xFF;
        P1=P1^0xFF;
        P2=P2^0xFF;
        MSDelay(250);
      }
  }
void MSDelay(unsigned int itime)
  {
    unsigned int i, j;
    for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);
  }
```

Example 19-22

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;              //sbit is used declare port (SFR) bits
bit membit;                    //notice this is bit-addressable memory
void main(void)
  {
    while(1)
      {
        membit=inbit;          //get a bit from P1.0
        outbit=~membit;        //invert it and send it to P2.7
      }
  }
```

Example 19-23

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)
  {
    unsigned char z;
    z=P1;                      //read P1
    z=z&0x3;                   //mask the unused bits
    switch(z)                  //make decision
      {
        case(0):
          {
            P0='0';            //issue ASCII 0
            break;
          }
        case(1):
          {
            P0='1';            //issue ASCII 1
            break;
          }
        case(2):
          {
            P0='2';            //issue ASCII 2
            break;
          }
        case(3):
          {
            P0='3';            //issue ASCII 3
            break;
          }
      }
  }
```

19.3.3: Review questions

Find the content of P1 after the following C code in each case.

1. (a) `P1=0x37&0xCA;` (b) `P1=0x37|0xCA;` (c) `P1=0x37^0xCA;`
2. To mask certain bits we must AND them with _____.
3. To set high certain bits we must OR them with _____.
4. Ex-ORing a value with itself results in _____.
5. Find the contents of P2 after execution of the following code.

```
P2=0;
P2=P2|0x99;
P2=~P2;
```

SECTION 19.4: DATA CONVERSION PROGRAMS IN 8051 C

Recall that BCD numbers were discussed in Chapter 18. As stated there, many newer microcontrollers have a real-time clock (RTC) where the time and date are kept even when the power is off. Very often the RTC provides the time and date in packed BCD. However, to display them they must be converted to ASCII. In this section we show the application of logic and rotate instructions in the conversion of BCD and ASCII.

19.4.1: ASCII numbers

On ASCII keyboards, when the key "0" is activated, "011 0000" (30H) is provided to the computer. Similarly, 31H (011 0001) is provided for the key "1", and so on, as shown in Table 19-4.

19.4.2: Packed BCD to ASCII conversion

The RTC provides the time of day (hour, minute, second) and the date (year, month, day) continuously, regardless of whether the power is on or off. However, this data is provided in packed BCD. To convert packed BCD to ASCII, it must first be converted to unpacked BCD. Then the unpacked BCD is tagged with 011 0000 (30H). The following demonstrates converting from packed BCD to ASCII. See also Example 19-24.

```
Packed BCD      Unpacked BCD          ASCII
0x29            0x02, 0x09            0x32, 0x39
00101001        00000010,00001001     00110010,00111001
```

Table 19-4: ASCII Code for Digits 0 - 9

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

19.4.3: ASCII to packed BCD conversion

To convert ASCII to packed BCD, it is first converted to unpacked BCD (to get rid of the 3), and then combined to make packed BCD. For example, 4 and 7 on the keyboard give 34H and 37H, respectively. The goal is to produce 47H or "0100 0111", which is packed BCD.

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 or 47H

After this conversion, the packed BCD numbers are processed and the result will be in packed BCD format. Chapter 23 discusses the RTC chip and uses the BCD and ASCII conversion programs shown in Examples 19-24 and 19-25.

Example 19-24

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;      //mask lower 4 bits
    P1 = x | 0x30;          //make it ASCII
    y = mybyte & 0xF0;      //mask upper 4 bits
    y = y >> 4;             //shift it to lower 4 bits
    P2 = y | 0x30;          //make it ASCII
  }
```

Example 19-25

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;     //mask 3
    w = w << 4;       //shift left to make upper BCD digit
    z = z & 0x0F;     //mask 3
    bcdbyte = w | z; //combine to make packed BCD
    P1 = bcdbyte;
  }
```

19.4.4: Checksum byte in ROM

To ensure the integrity of ROM contents, every system must perform the checksum calculation. The process of checksum will detect any corruption of the contents of ROM. One of the causes of ROM corruption is current surge, either when the system is turned on or during operation. To ensure data integrity in ROM, the checksum process uses what is called a *checksum byte*. The checksum byte is an extra byte that is tagged to the end of a series of bytes of data. To calculate the checksum byte of a series of bytes of data, the following steps can be taken.

1. Add the bytes together and drop the carries.
2. Take the 2's complement of the total sum. This is the checksum byte, which becomes the last byte of the series.

To perform the checksum operation, add all the bytes, including the checksum byte. The result must be zero. If it is not zero, one or more bytes of data have been changed (corrupted). To clarify these important concepts, see Example 19-26.

Example 19-26

Assume that we have 4 bytes of hexadecimal data: 25H, 62H, 3FH, and 52H. (a) Find the checksum byte, (b) perform the checksum operation to ensure data integrity, and (c) if the second byte 62H has been changed to 22H, show how checksum detects the error.

Solution:

(a) Find the checksum byte.

```
  25H
+ 62H
+ 3FH
+ 52H
-----
 118H
```

(Dropping carry of 1 and taking the 2's complement, we get E8H.)

(b) Perform the checksum operation to ensure data integrity.

```
  25H
+ 62H
+ 3FH
+ 52H
+ E8H
-----
 200H
```

(Dropping the carries we get 00, which means data is not corrupted.)

(c) If the second byte 62H has been changed to 22H, show how checksum detects the error.

```
  25H
+ 22H
+ 3FH
+ 52H
+ E8H
-----
 1C0H
```

(Dropping the carry, we get C0H, which means data is corrupted.)

Example 19-27

Write an 8051 C program to calculate the checksum byte for the data given in Example 19-26.

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];              //issue each byte to P2
        sum=sum+mydata[x];         //add them together
        P1=sum;                    //issue the sum to P1
      }
    chksumbyte=~sum+1;             //make 2's complement
    P1=chksumbyte;                 //show the checksum byte
}
```

Single-step the above program on the 8051 simulator and examine the contents of P1 and P2. Notice that each byte is put on P1 as they are added together.

Example 19-28

Write an 8051 C program to perform step (b) of Example 19-26. 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 chksum=0;
    unsigned char x;
    for(x=0;x<5;x++)
      chksum=chksum+mydata[x];   //add them together
    if(chksum==0)
      P0='G';
    else
      P0='B';
  }
```

19.4.5: Binary (hex) to decimal and ASCII conversion in 8051 C

The printf function is part of the standard I/O library in C and can do many things, including converting data fr binary (hex) to decimal, or vice versa. But printf takes a lot of memory space and increases your hex file substantia For this reason, in systems based on the 8051 microcontroller, it is better to write your own conversion function inst of using printf.

One of the most widely used conversions is the binary to decimal conversion. In devices such as ADC (Analog Digital Conversion) chips, the data is provided to the microcontroller in binary. In some RTCs, data such as time dates are also provided in binary. In order to display binary data we need to convert it to decimal and then to AS Since the hexadecimal format is a convenient way of representing binary data we refer to the binary data as hex.

ample 19-29

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

olution:

```
#includee <reg51.h>
void main(void)
{
  unsigned char x, binbyte, d1, d2, d3;
  binbyte = 0xFD;          //binary(hex) byte
  x = binbyte / 10;        //divide by 10
  d1 = binbyte % 10;       //find remainder (LSD)
  d2 = x % 10;             //middle digit
  d3 = x / 10;             //most significant digit (MSD)
  P0 = d1;
  P1 = d2;
  P2 = d3;
```

ary data 00 - FFH converted to decimal will give us 000 to 255. One way to do that is to divide it by 10 and keep the mainder, as was shown in Chapter 18. For example, 11111101 or FDH is 253 in decimal. The following is one version an algorithm for conversion of hex (binary) to decimal:

	Quotient	Remainder
FD/0A	19	3 (low digit) LSD
19/0A	2	5 (middle digit)
		2 (high digit) (MSD)

Example 19-29 shows the C program for that algorithm.

9.4.6: Review questions

For the following decimal numbers, give the packed BCD and unpacked BCD representations.
(a) 15 (b) 99
Show the binary and hex formats for "76" and its BCD version.
67H in BCD when converted to ASCII is ____H and ____H.
Does the following convert unpacked BCD in register A to ASCII?

```
mydata = 0x09 + 0x30;
```

Why is the use of packed BCD preferable to ASCII?
Which one takes more memory space: packed BCD or ASCII?
In Question 6, which is more universal?
Find the checksum byte for the following values; 22H, 76H, 5FH, 8CH, 99H.
To test data integrity, we add them together, including the checksum byte. Then drop the carries. The result must be equal to ____ if the data is not corrupted.
). An ADC provides an input of 0010 0110. What happens if we output that to the screen?

SECTION 19.5: ACCESSING CODE ROM SPACE IN 8051 C

Using the code (program) space for predefined data is the widely used option in the 8051, as we saw in Chapter 17. In that chapter we saw how to use the Assembly language instruction MOVC to access the data stored in the 8051 code space. In this chapter, we explore the same concept for 8051 C.

19.5.1: RAM data space v. code data space

In the 8051 we have three spaces in which to store data. They are as follows:

1. The 128 bytes of RAM space with address range 00 - 7FH. (In the 8052, it is 256 bytes.) We can read (from) or write (into) this RAM space directly or indirectly using the R0 and R1 registers as we saw in Chapter 17.
2. The 64K bytes of code (program) space with addresses of 0000 - FFFFH. This 64K bytes of on-chip ROM space is used for storing programs (opcodes) and therefore is directly under the control of the program counter (PC). We can use the "`MOVC A, @A+DPTR`" Assembly language instruction to access it for data (see Chapter 17). There are two problems with using this code space for data. First, since it is ROM memory, we can burn our predefined data and tables into it. But we cannot write into it during the execution of the program. The second problem is that the more of this code space we use for data, the less is left for our program code. For example, if we have an 8051 chip such as DS89C420 with only 16K bytes of on-chip ROM, and we use 4K bytes of it to store some look-up table, only 12K bytes is left for the code program. For some applications this can be a problem. For this reason Intel created another memory space called *external memory* especially for data. This is discussed next very briefly and we postpone the full discussion to Chapter 22.
3. The 64K bytes of external memory, which can be used for both RAM and ROM. This 64K bytes is called external since we must use the MOVX Assembly language instruction to access it. At the time the 8051 was designed, the cost of on-chip ROM was very high; therefore, Intel used all the on-chip ROM for code but allowed connection to external RAM and ROM. In other words, we have a total of 128K bytes of memory space since the off-chip or external memory space of 64K bytes plus the 64K bytes of on-chip space provides you a total of 128K bytes of memory space. We will discuss the external memory expansion and how to access it for both Assembly and C in Chapter 22.

Next, we discuss on-chip RAM and ROM space usage by the 8051 C compiler. We have used the Proview32 C compiler to verify the concepts discussed next. Use the compiler of your choice to verify these concepts.

19.5.2: RAM data space usage by the 8051 C compiler

In Assembly language programming, as shown in Chapters 14 and 17, the 128 bytes of RAM space is used mainly by register banks and the stack. Whatever remains is used for scratch pad RAM. The 8051 C compiler first allocates the first 8 bytes of the RAM to bank 0 and then some RAM to the stack. Then it starts to allocate the rest to the variables declared by the C program. While in Assembly the default starting address for the stack is 08, the C compiler moves the stack's starting address to somewhere in the range of 50 - 7FH. This allows us to allocate contiguous RAM locations to array elements.

In cases where the program has individual variables in addition to array elements, the 8051 C compiler allocates RAM locations in the following order:

1.	Bank 0	addresses 0 - 7
2.	Individual variables	addresses 08 and beyond
3.	Array elements	addresses right after variables
4.	Stack	addresses right after array elements

You can verify the above order by running Example 19-30 on your 8051 C simulator and examining the contents of the data RAM space. Remember that 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. In the case of Example 19-31 the array elements are limited to around 100. Run Example 19-31 on your 8051 C simulator and examine the RAM space allocation. Keep changing the size of the array and monitor the RAM space to see what happens.

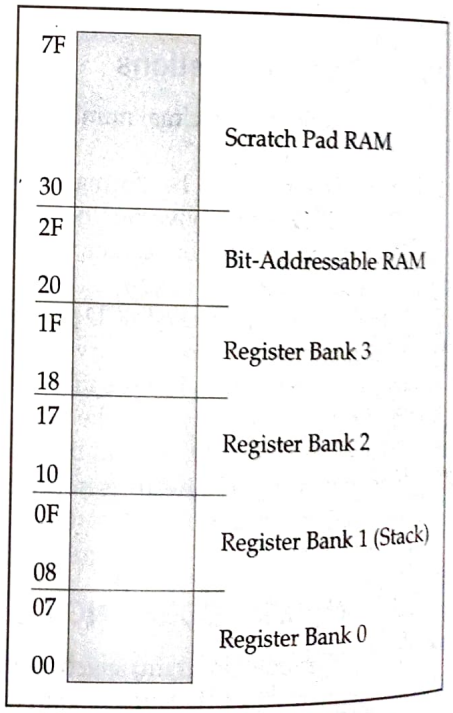


Figure 19-1. RAM allocation in the 8051

ample 19-30

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

ution:

```
nclude <reg51.h>
id main(void)
{
 unsigned char mynum[]= "ABCDEF";     //This uses RAM space
                                      //to store data
 unsigned char z;
 for(z=0;z<=6;z++)
   P1=mynum [z];
}
```

un the above program on your 8051 simulator and examine the RAM space to locate values 41H, 42H, 43H, 44H, c. the hex values for ASCII letters 'A', 'B', 'C', and so on.

ample 19-31

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];    //100 byte space in RAM
  unsigned char x,z=0;
  for(x=0;x<100;x++)
    {
      z--;                      //count down
      mydata[x]=z;              //save it in RAM
      P1=z;                     //give a copy to P1 too
    }
}
```

Run the above program on your 8051 simulator and examine the data RAM space to locate values FFH, FEH, FDH, and so on in RAM.

5.3: The 8052 RAM data space

Intel added some new features to the 8051 microcontroller and called it the 8052. One of the new features was an tra 128 bytes of RAM space. That means that the 8052 has 256 bytes of RAM space instead of 128 bytes. Remember t the 8052 is code-compatible with the 8051. This means that any program written for the 8051 will run on the 8052, t not the other way around since some features of the 8052 do not exist in the 8051. The extra 128 bytes of RAM helps e 8051/52 C compiler to manage its registers and resources much more effectively. Since the vast majority of the new rsions of the 8051 such as DS89C4x0 are really based on 8052 architecture, you should compile your C programs for e 8052 microcontroller. We do that by (1) using the reg52.h header file, and (2) choosing the 8052 option when compil- g the program.

Example 19-32

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";   //uses code space
                                            //for data
    unsigned char z;
    for(z=0;z<=6;z++)
      P1=mynum[z];
  }
```

Run the above program on your 8051 simulator and examine the code space to locate values 41H, 42H, 43H, 44H, etc., the hex values for ASCII characters of 'A', 'B', 'C', and so on.

19.5.4: Accessing code data space in 8051 C

In all our 8051 C examples so far, byte-size variables were stored in the 128 bytes of RAM. 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. The following are some examples:

```
code unsigned char mynum[]= "012345ABCD";     //use code space
code unsigned char weekdays=7, month=0x12;    //use code space
```

Example 19-32 shows how to use code space for data in 8051 C.

19.5.5: Compiler variations

Look at Example 19-33. It shows three different versions of a program that sends the string "HELLO" to the P1 port. Compile each program with the 8051 C compiler of your choice and compare the hex file size. Then compile each program on a different 8051 C compiler, and examine the hex file size to see the effectiveness of your C compiler. See www.MicroDigitalEd.com for 8051 C compilers.

Example 19-33

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';
  }
```

(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)
  {
    //Notice Keyword code
    code unsigned char mydata[]="HELLO";
    unsigned char z;
    for(z=0;z<=5;z++)
      P1=mydata[z]';
  }
```

Solution:

All the programs send out "HELLO" to P1, one character at a time, but they do it in different ways. The first one is short and simple, but the individual characters are embedded into the program. If we change the characters, the whole program changes. It also mixes the code and data together. The second one uses the RAM data space to store array elements, therefore the size of the array is limited. The third one uses 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. Both programs (b) and (c) are easily upgradable if we want to change the string itself or make it longer. That is not the case for program (a).

See the following Web sites for 8051 C compilers:

www.MicroDigitalEd.com

www.8052.com

19.5.6: Review questions

1. The 8051 has _____ bytes of data RAM, while the 8052 has _____ bytes.
2. The 8051 has _____ K bytes of code space and _____ K bytes of external data space.
3. True or false. The code space can be used for data but the external data space cannot be used for code.
4. Which space would you use to declare the following values for 8051 C?
   (a) the number of days in the week
   (b) the number of months in a year
   (c) a counter for a delay
5. In 8051 C, we should not use more than 100 bytes of the RAM data space for variables. Why?

SECTION 19.6: DATA SERIALIZATION USING 8051 C

Serializing data is a way of sending a byte of data one bit at a time through a single pin of microcontroller. There are two ways to transfer a byte of data serially:

1. Using the serial port. When using the serial port, the programmer has very limited control over the sequence of data transfer.
2. The second method of serializing data is to 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 printed circuit board.

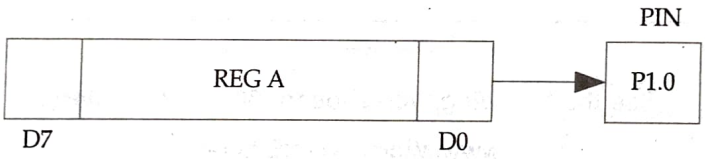
Examine the next four examples to see how data serialization is done in 8051 C.

Example 19-34

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:

```
//SERIALIZING DATA VIA P1.0 (SHIFTING RIGHT)
#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;
      }
  }
```



Example 19-35

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:

```
//SERIALIZING DATA VIA P1.0 (SHIFTING LEFT)
#include <reg51.h>
sbit P1b0 = P1^0;
sbit regAMSB = ACC^7;
void main(void)
  {
    unsigned char conbyte = 0x44;
```

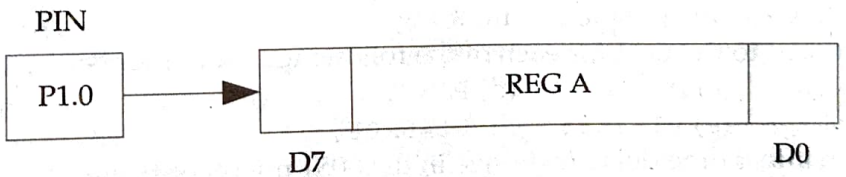
```
    unsigned char x;
    ACC = conbyte;
    for(x=0; x<8; x++)
      {
        P1b0 = regAMSB;
        ACC = ACC << 1;
      }
  }
```

Example 19-36

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:

```
//BRINGING IN DATA VIA P1.0 (SHIFTING RIGHT)
#include <reg51.h>
sbit P1b0 = P1^0;
sbit ACCMSB = ACC^7;
void main(void)
  {
    unsigned char conbyte = 0x44;
    unsigned char x;
    for(x=0; x<8; x++)
      {
        ACCMSB = P1b0;
        ACC = ACC >> 1;
      }
    P2=ACC;
  }
```



Example 19-37

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:

```
//BRINGING DATA IN VIA P1.0 (SHIFTING LEFT)
#include <reg51.h>
sbit P1b0 = P1^0;
sbit regALSB = ACC^0;
void main(void)
  {
    unsigned char x;
```

```
    for(x=0; x<8; x++)
      {
        regALSB = P1b0;
        ACC = ACC << 1;
      }
    P2=ACC;
}
```

SUMMARY

This chapter dealt with 8051 C programming, especially I/O programming and time delays in 8051 C. We also showed the logic operators AND, OR, XOR, and complement. In addition, some applications for these operators were discussed. This chapter also described BCD and ASCII formats and conversions in 8051 C. We also compared and contrasted the use of code space and RAM data space in 8051 C. The widely used technique of data serialization was also discussed.

PROBLEMS

SECTION 19-1: DATA TYPES AND TIME DELAY IN 8051 C

1. Indicate what data type you would use for each of the following variables:
   (a) the temperature
   (b) the number of days in a week
   (c) the number of days in a year
   (d) the number of months in a year
   (e) the counter to keep the number of people getting on a bus
   (f) the counter to keep the number of people going to a class
   (g) an address of 64K bytes RAM space
   (h) the voltage
   (i) a string for a message to welcome people to a building
2. Give the hex value that is sent to the port for each of the following C statements:
   (a) `P1=14;` (b) `P1=0x18;` (c) `P1='A';` (d) `P1=7;`
   (e) `P1=32;` (f) `P1=0x45;` (g) `P1=255;` (h) `P1=0x0F;`
3. Give three factors that can affect time delay code size in the 8051 microcontroller.
4. Of the three factors in Problem 3, which one can be set by the system designer?
5. Can the programmer set the number of clock cycles used to execute an instruction? Explain your answer.
6. Explain why various 8051 C compilers produce different hex file sizes.

SECTION 19-2: I/O PROGRAMMING IN 8051 C

7. What is the difference between the sbit and bit data types?
8. Write an 8051 C program to toggle all bits of P1 every 200 ms.
9. Use your 8051 C compiler to see the shortest time delay that you can produce.
10. Write a time delay function for 100 ms.
11. Write an 8051 C program to toggle only bit P1.3 every 200 ms.
12. Write an 8051 C program to count up P1 from 0 - 99 continuously.

SECTION 19-3: LOGIC OPERATIONS IN 8051 C

13. Indicate the data on the ports for each of the following.
    Note: The operations are independent of each other.

(a) `P1=0xF0&0x45;` (b) `P1=0xF0&0x56;`
(c) `P1=0xF0^0x76;` (d) `P2=0xF0&0x90;`
(e) `P2=0xF0^0x90;` (f) `P2=0xF0|0x90;`
(g) `P2=0xF0&0xFF;` (h) `P2=0xF0|0x99;`
(i) `P2=0xF0^0xEE;` (j) `P2=0xF0^0xAA;`

14. Find the contents of the port after each of the following operations.
(a) `P1=0x65&0x76;` (b) `P1=0x70|0x6B;`
(c) `P2=0x95^0xAA;` (d) `P2=0x5D&0x78;`
(e) `P2=0xC5|0x12;` (f) `P0=0x6A^0x6E;`
(g) `P1=0x37|0x26;`

15. Find the port value after each of the following is executed.
(a) `P1=0x65>>2;` (b) `P2=0x39<<2;`
(c) `P1=0xD4>>3;` (d) `P1=0xA7<<2;`

16. Show the C code to swap 0x95 to make it 0x59.
17. Write a C program that finds the number of zeros in an 8-bit data item.
18. A stepper motor uses the following sequence of binary numbers to move the motor. How would you generate them in 8051 C?
`1100,0110,0011,1001`

SECTION 19-4: DATA CONVERSION PROGRAMS IN 8051 C

19. Write a program to convert the following series of packed BCD numbers to ASCII. Assume that the packed BCD located in data RAM.
`76H, 87H, 98H, 43H`
20. Write a program to convert the following series of ASCII numbers to packed BCD. Assume that the ASCII data located in data RAM.
`"8767"`
21. Write a program to get an 8-bit binary number from P1, convert it to ASCII, and save the result if the input is packe BCD of 00 - 0x99. Assume P1 has 1000 1001 binary as input.

SECTION 19-5: ACCESSING CODE ROM SPACE IN 8051 C

22. Indicate what memory (embedded, data RAM, or code ROM space) you would use for the following variables:
(a) the temperature
(b) the number of days in week
(c) the number of days in a year
(d) the number of months in a year
(e) the counter to keep the number of people getting on a bus
(f) the counter to keep the number of people going to a class
(g) an address of 64K bytes RAM space
(h) the voltage
(i) a string for a message to welcome people to building
23. Discuss why the total size of your 8051 C variables should not exceed 100 bytes.
24. Why do we use the ROM code space for video game characters and shapes?
25. What is the drawback of using RAM data space for 8051 C variables?
26. What is the drawback of using ROM code space for 8051 C data?
27. Write an 8051 C program to send your first and last names to P2. Use ROM code space.

ANSWERS TO REVIEW QUESTIONS

SECTION 19-1: DATA TYPES AND TIME DELAY IN 8051 C

1. 0 to 255 for unsigned char and –128 to +127 for signed char
2. 0 to 65,535 for unsigned int and –32,768 to +32,767 for signed int
3. Unsigned char
4. True
5. (a) Crystal frequency of 8051 system, (b) 8051 machine cycle timing, and (c) compiler use for 8051 C

SECTION 19-2: I/O PROGRAMMING IN 8051 C

1. 90H
2.
```
#include <reg51.h>
void main()
  {
    P2 = 0x55;
    P2 = 0xAA
  }
```
3.
```
#include <reg51.h>
sbit P10bit = P1^0;
void main()
  {
    P10bit = 0;
    P10bit = 1;
  }
```
4. False, only to SFR bit
5. True

SECTION 19-3: LOGIC OPERATIONS IN 8051 C

1. (a) 02 (b) FFH (c) FDH
2. Zeros
3. One
4. All zeros
5. 66H

SECTION 19-4: DATA CONVERSION PROGRAMS IN 8051 C

1. (a) 15H = 0001 0101 packed BCD, 0000 0001,0000 0101 unpacked BCD
   (b) 99H = 1001 1001 packed BCD, 0000 1001,0000 1001 unpacked BCD
2. 3736H = 00110111 00110110B
   and in BCD we have 76H = 0111 0110B
3. 36H, 37H
4. Yes, since A = 39H
5. Space savings
6. ASCII
7. ASCII
8. 21CH
9. 00
10. First convert from binary to decimal, then to ASCII, then send to screen.

SECTION 19-5: ACCESSING CODE ROM SPACE IN 8051 C

1. 128, 256
2. 64K, 64K
3. True
4. (a) data space, (b) data space, (c) RAM space
5. The compiler starts storing variables in code space.