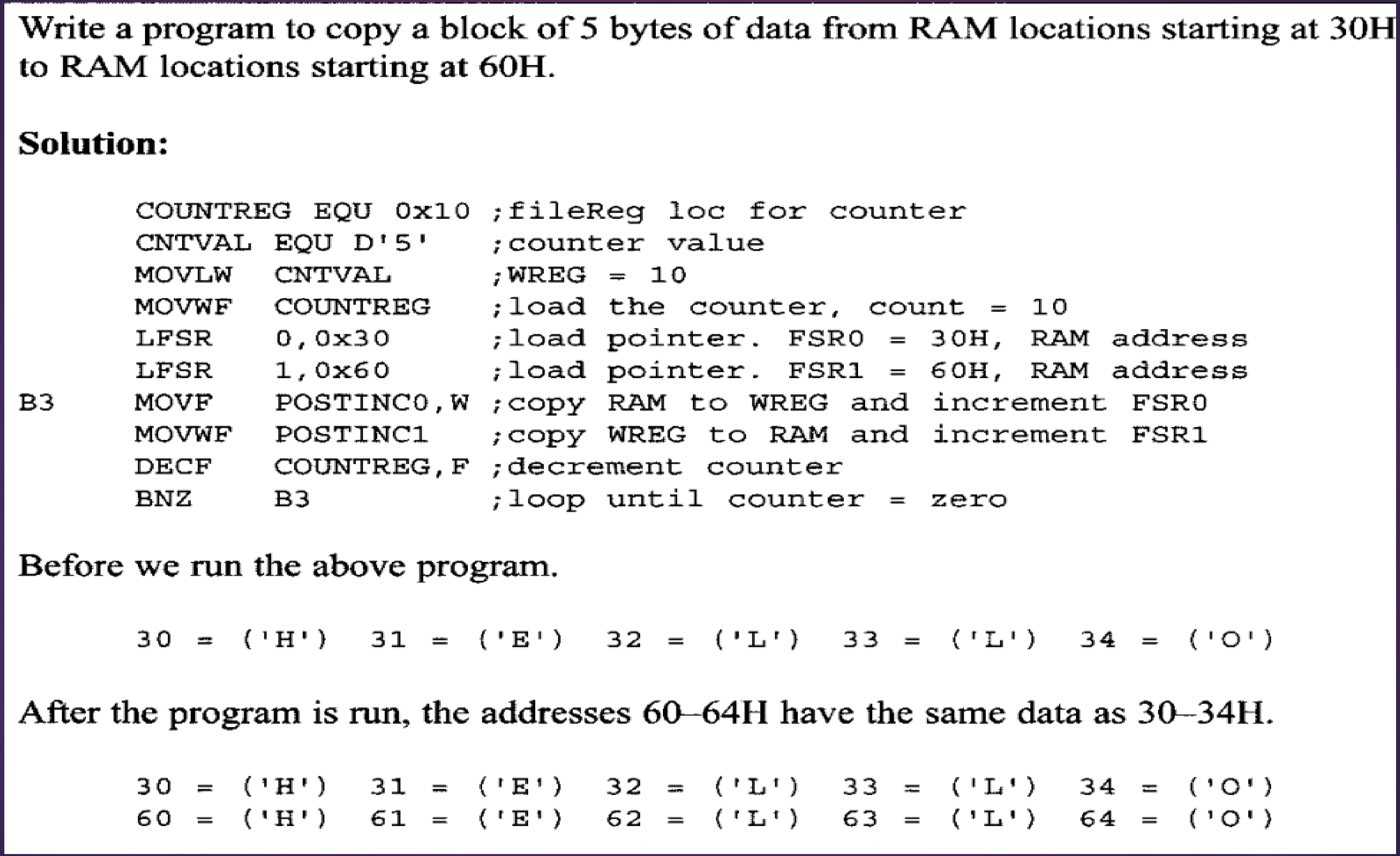
# Microprocessor Interfacing & Programming

LECTURE 18 & 19

## Example



Write a program to copy a block of 5 bytes of data from RAM locations starting at 30H to RAM locations starting at 60H.

**Solution:**

```
      COUNTREG EQU 0x10 ;fileReg loc for counter
      CNTVAL EQU D'5'   ;counter value
      MOVLW   CNTVAL    ;WREG = 10
      MOVWF   COUNTREG  ;load the counter, count = 10
      LFSR    0,0x30    ;load pointer. FSR0 = 30H, RAM address
      LFSR    1,0x60    ;load pointer. FSR1 = 60H, RAM address
B3    MOVF    POSTINC0,W ;copy RAM to WREG and increment FSR0
      MOVWF   POSTINC1  ;copy WREG to RAM and increment FSR1
      DECF    COUNTREG,F ;decrement counter
      BNZ     B3        ;loop until counter = zero
```

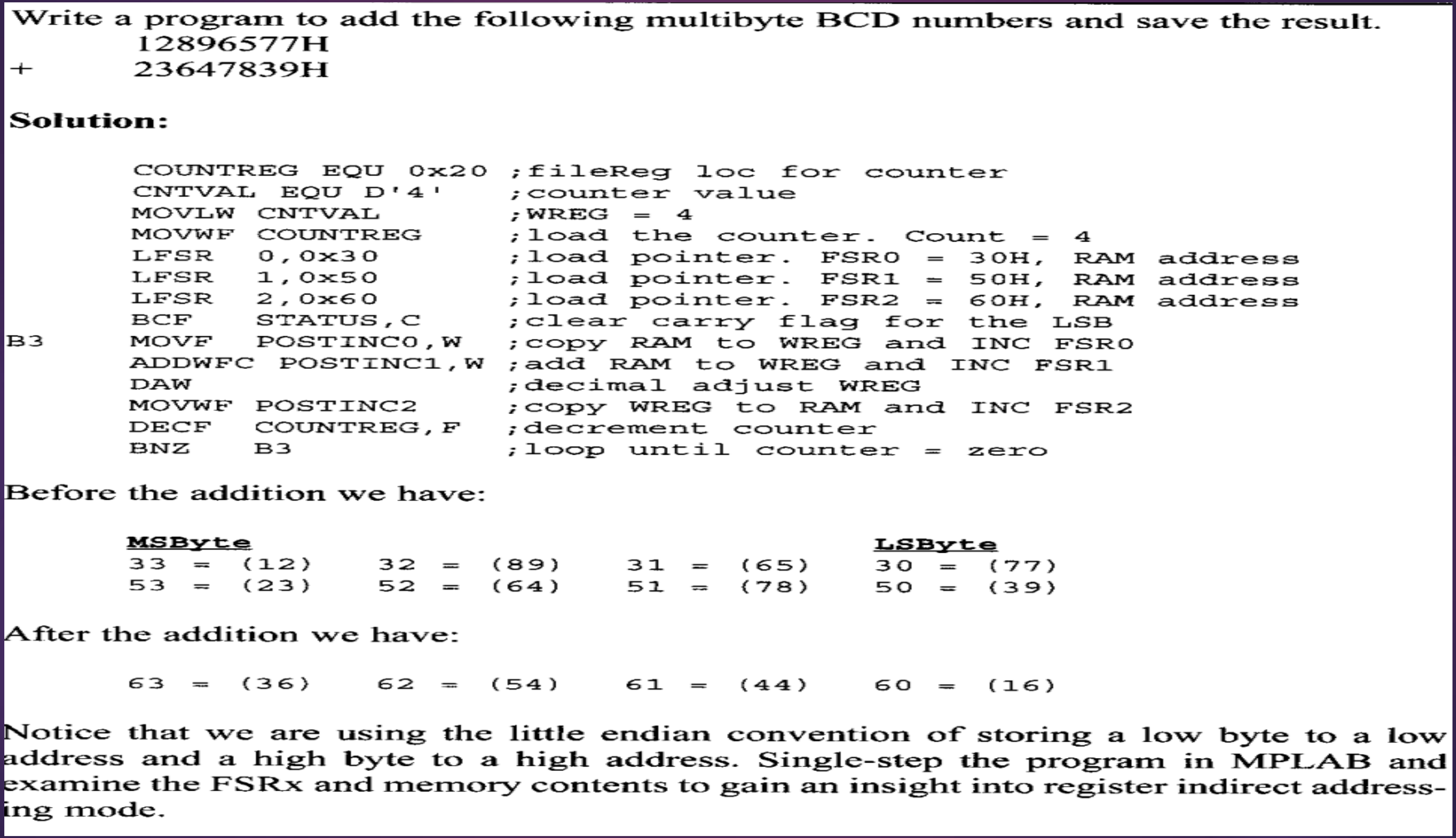
Before we run the above program.

```
      30 = ('H')  31 = ('E')  32 = ('L')  33 = ('L')  34 = ('O')
```

After the program is run, the addresses 60–64H have the same data as 30–34H.

```
      30 = ('H')  31 = ('E')  32 = ('L')  33 = ('L')  34 = ('O')
      60 = ('H')  61 = ('E')  62 = ('L')  63 = ('L')  64 = ('O')
```

# Example



Write a program to add the following multibyte BCD numbers and save the result.

```
        12896577H
+       23647839H
```

**Solution:**

```
        COUNTREG EQU 0x20 ;fileReg loc for counter
        CNTVAL EQU D'4'   ;counter value
        MOVLW  CNTVAL     ;WREG = 4
        MOVWF  COUNTREG   ;load the counter. Count = 4
        LFSR   0,0x30     ;load pointer. FSR0 = 30H, RAM address
        LFSR   1,0x50     ;load pointer. FSR1 = 50H, RAM address
        LFSR   2,0x60     ;load pointer. FSR2 = 60H, RAM address
        BCF    STATUS,C   ;clear carry flag for the LSB
B3      MOVF   POSTINC0,W ;copy RAM to WREG and INC FSR0
        ADDWFC POSTINC1,W ;add RAM to WREG and INC FSR1
        DAW               ;decimal adjust WREG
        MOVWF  POSTINC2   ;copy WREG to RAM and INC FSR2
        DECF   COUNTREG,F ;decrement counter
        BNZ    B3         ;loop until counter = zero
```

Before the addition we have:

```
        MSByte                                  LSByte
        33 = (12)   32 = (89)   31 = (65)   30 = (77)
        53 = (23)   52 = (64)   51 = (78)   50 = (39)
```

After the addition we have:

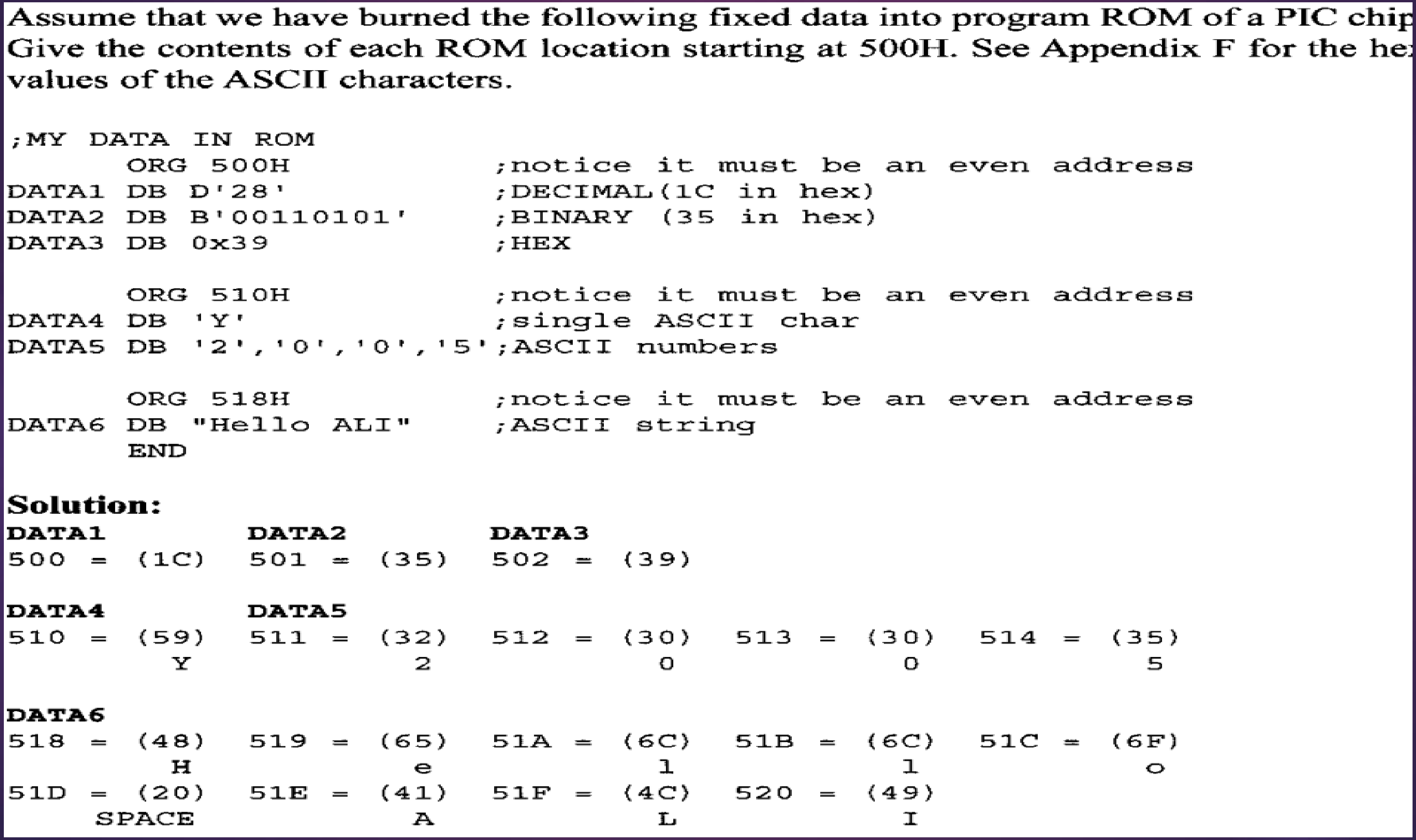
```
        63 = (36)   62 = (54)   61 = (44)   60 = (16)
```

Notice that we are using the little endian convention of storing a low byte to a low address and a high byte to a high address. Single-step the program in MPLAB and examine the FSRx and memory contents to gain an insight into register indirect addressing mode.

## Look-up Table

- We can use the code space to store fixed data using DB (define byte) directive.
- The DB data directive is widely used to allocate ROM program (code) memory in byte sized chunks.
- DB is used to define an 8 bit fixed data.
- When DB is used, the numbers can be in decimal, binary, hex or ASCII formats.
- DB is used to define ASCII strings.

# Example



Assume that we have burned the following fixed data into program ROM of a PIC chip
Give the contents of each ROM location starting at 500H. See Appendix F for the he
values of the ASCII characters.

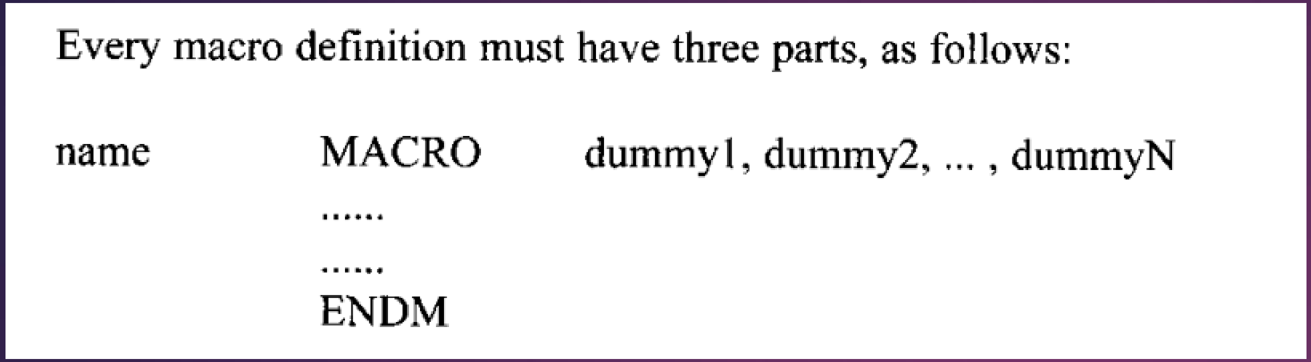
```
;MY DATA IN ROM
      ORG 500H               ;notice it must be an even address
DATA1 DB D'28'               ;DECIMAL(1C in hex)
DATA2 DB B'00110101'         ;BINARY (35 in hex)
DATA3 DB 0x39                ;HEX

      ORG 510H               ;notice it must be an even address
DATA4 DB 'Y'                 ;single ASCII char
DATA5 DB '2','0','0','5';ASCII numbers

      ORG 518H               ;notice it must be an even address
DATA6 DB "Hello ALI"         ;ASCII string
      END
```

**Solution:**

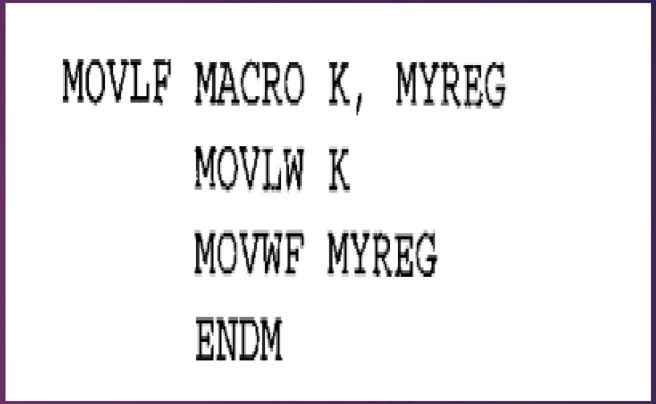
```
DATA1           DATA2           DATA3
500 = (1C)      501 = (35)      502 = (39)

DATA4           DATA5
510 = (59)      511 = (32)      512 = (30)      513 = (30)      514 = (35)
       Y               2               0               0               5

DATA6
518 = (48)      519 = (65)      51A = (6C)      51B = (6C)      51C = (6F)
       H               e               l               l               o
51D = (20)      51E = (41)      51F = (4C)      520 = (49)
     SPACE             A               L               I
```

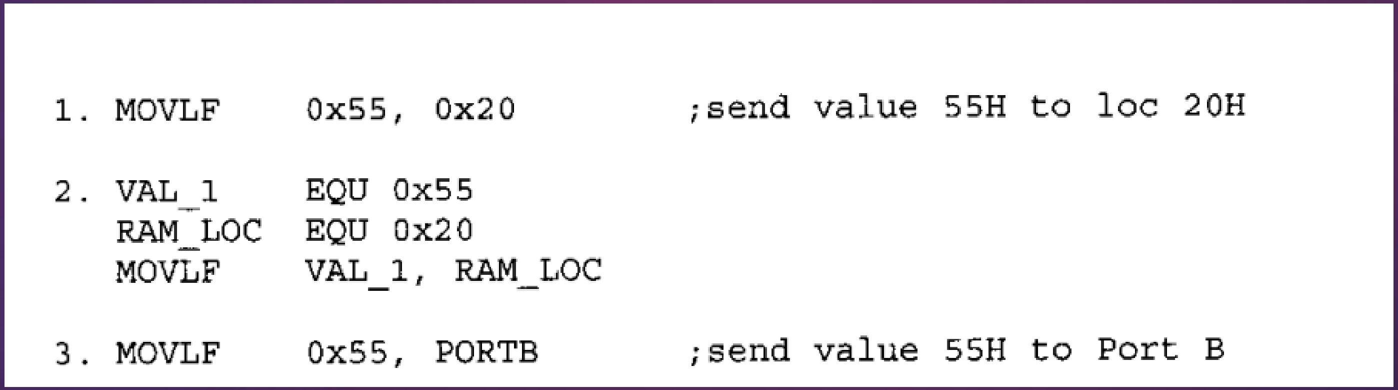
## Macros

- There are applications in assembly language programming where a group of instructions perform a task that is used repeatedly.
- For example, moving data into a RAM location is done repeatedly in the same program.
- Therefore, to reduce time and to reduce the possibility of errors, the concept of macros was introduced.
- Macros allows the programmer to write the task once only, and to invoke it whenever it is needed.

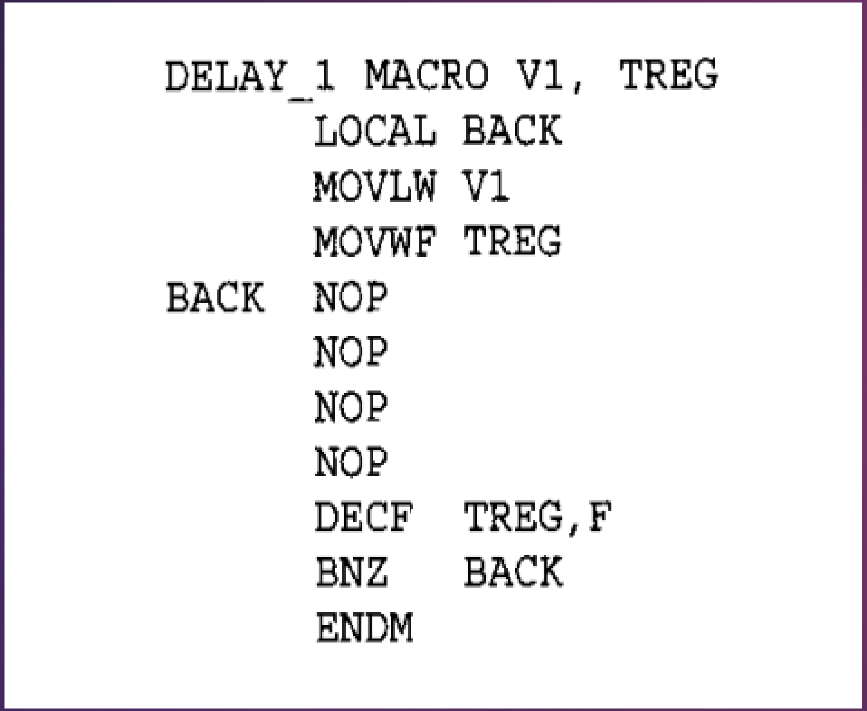

Every macro definition must have three parts, as follows:

```
name        MACRO       dummy1, dummy2, ... , dummyN
            ......
            ......
            ENDM
```

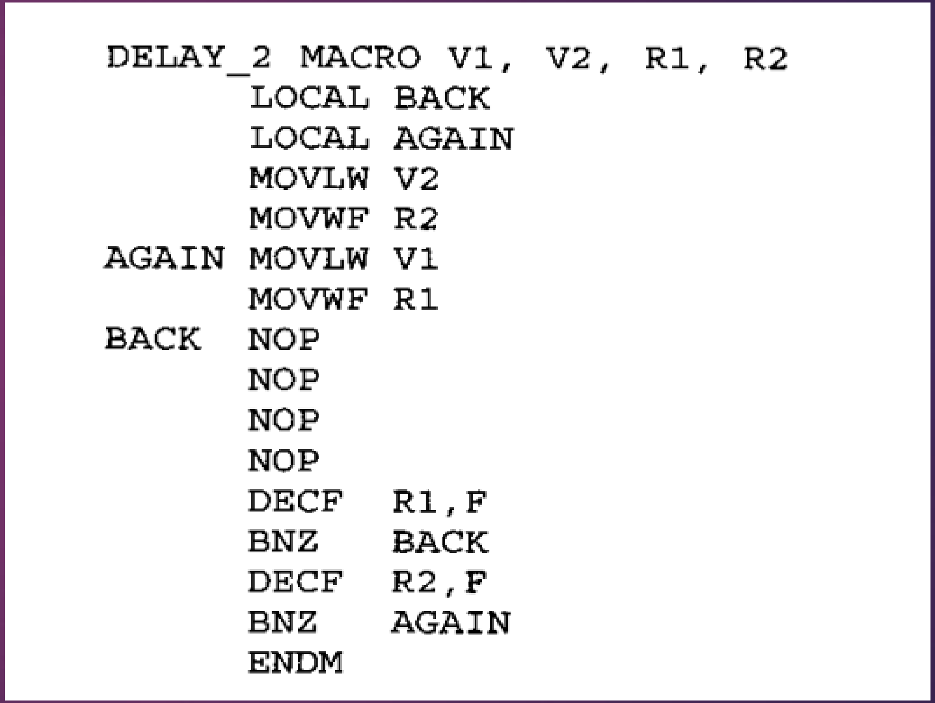

```
MOVLF MACRO K, MYREG
      MOVLW K
      MOVWF MYREG
      ENDM
```


```
1. MOVLF     0x55, 0x20        ;send value 55H to loc 20H

2. VAL_1     EQU 0x55
   RAM_LOC   EQU 0x20
   MOVLF     VAL_1, RAM_LOC

3. MOVLF     0x55, PORTB       ;send value 55H to Port B
```

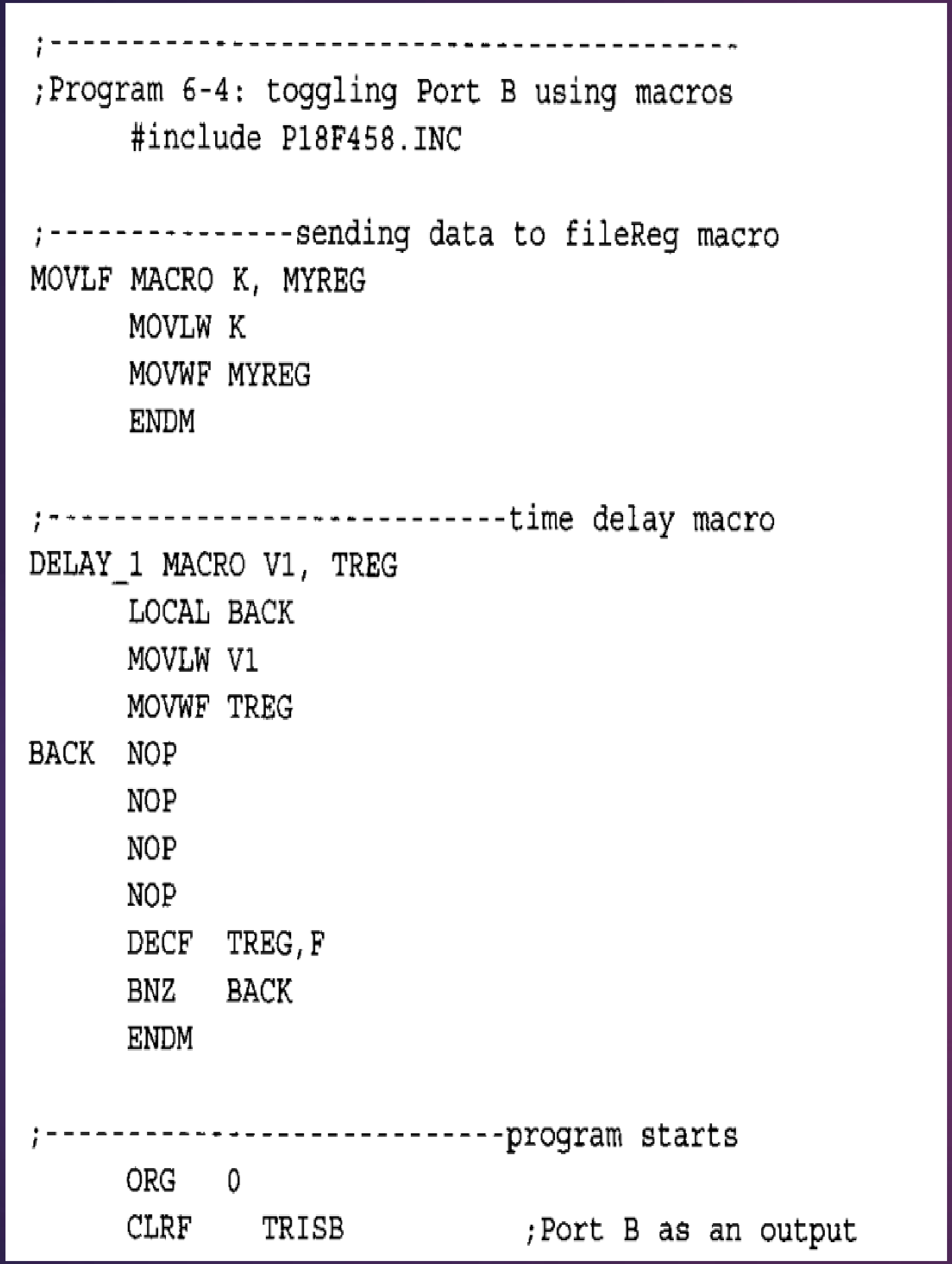
# Local directive



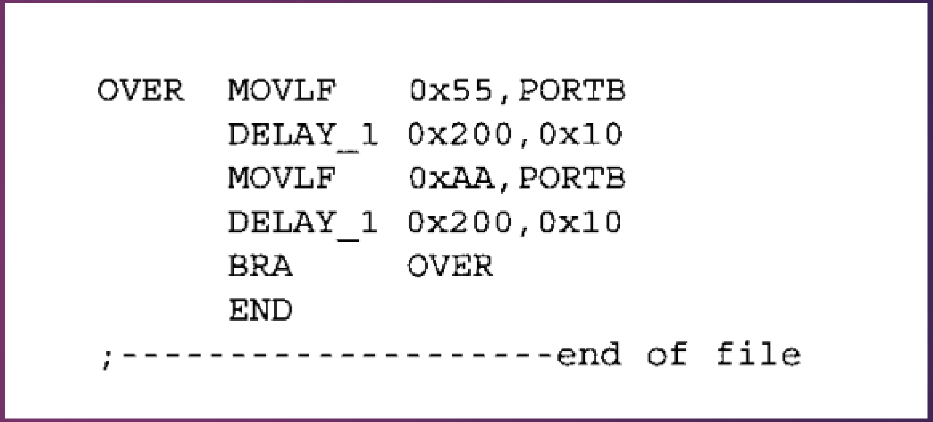
```
DELAY_1 MACRO V1, TREG
        LOCAL BACK
        MOVLW V1
        MOVWF TREG
BACK    NOP
        NOP
        NOP
        NOP
        DECF  TREG,F
        BNZ   BACK
        ENDM
```



```
DELAY_2 MACRO V1, V2, R1, R2
        LOCAL BACK
        LOCAL AGAIN
        MOVLW V2
        MOVWF R2
AGAIN   MOVLW V1
        MOVWF R1
BACK    NOP
        NOP
        NOP
        NOP
        DECF  R1,F
        BNZ   BACK
        DECF  R2,F
        BNZ   AGAIN
        ENDM
```



```
;-----------------------------------------
;Program 6-4: toggling Port B using macros
      #include P18F458.INC

;---------------sending data to fileReg macro
MOVLF MACRO K, MYREG
      MOVLW K
      MOVWF MYREG
      ENDM

;--------------------------time delay macro
DELAY_1 MACRO V1, TREG
      LOCAL BACK
      MOVLW V1
      MOVWF TREG
BACK  NOP
      NOP
      NOP
      NOP
      DECF  TREG,F
      BNZ   BACK
      ENDM

;----------------------------program starts
      ORG   0
      CLRF    TRISB           ;Port B as an output
```



```
OVER  MOVLF    0x55,PORTB
      DELAY_1  0x200,0x10
      MOVLF    0xAA,PORTB
      DELAY_1  0x200,0x10
      BRA      OVER
      END
;--------------------end of file
```

## PIC Programming in C

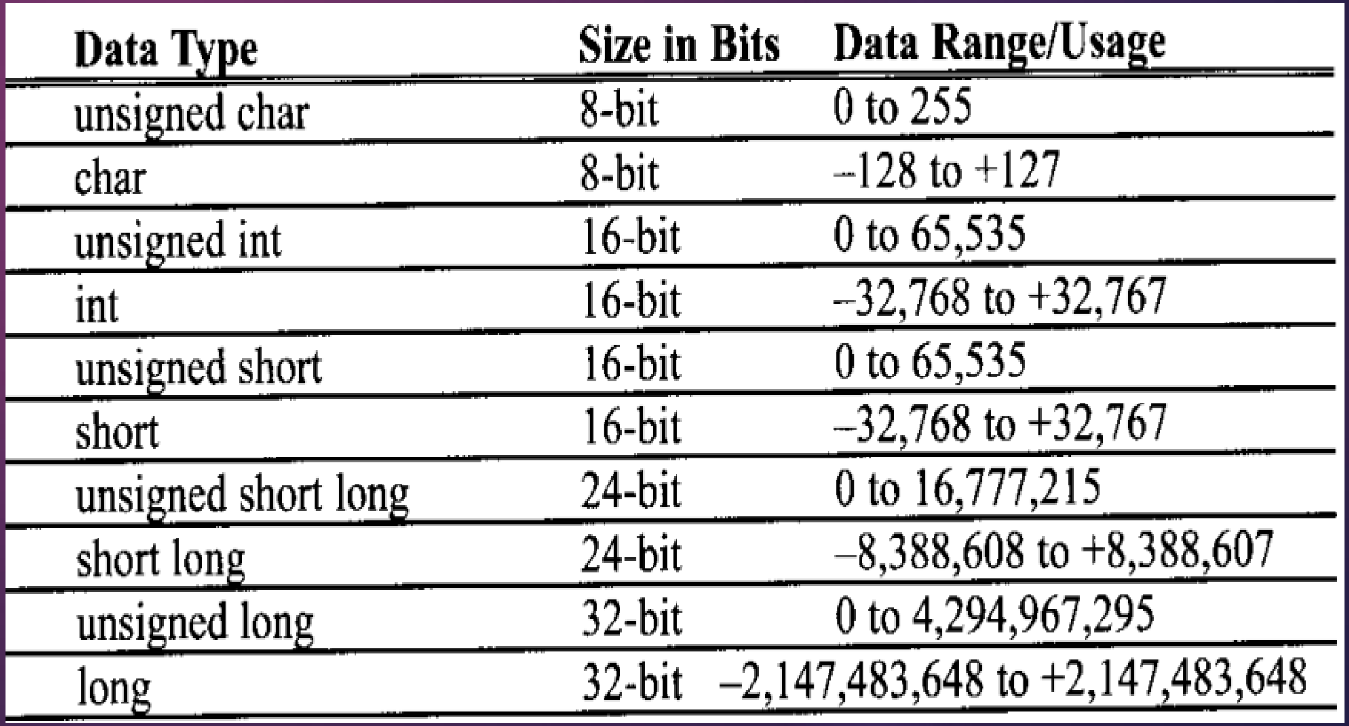
- Assembly language produces a hex file that is much smaller than C.
- C programming is less time consuming and much easier to write, but the hex file produced is much larger than if we used Assembly language.

**Reasons for writing programs in C instead of Assembly:**

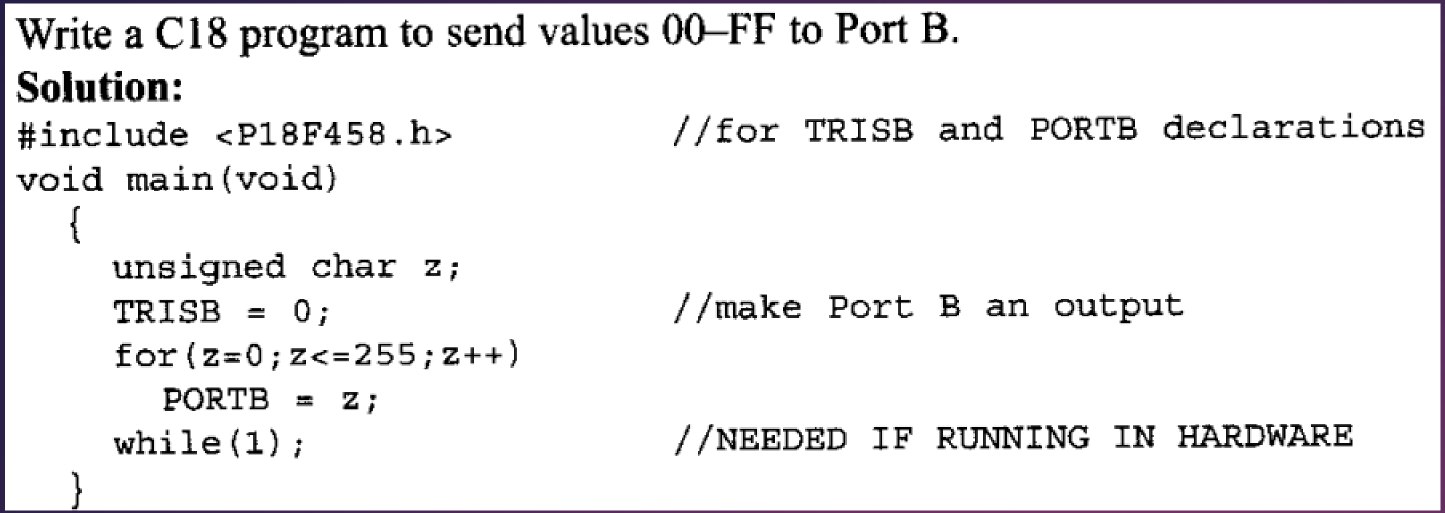
1. It is easier and less time consuming.
2. C is easier to modify and update.
3. You can use code avaialable in function libraries.
4. C code is portable to other microcontrollers with little or no modification.

# Data Types in C

- The goal is to create smaller hex files.
- Natural choice for many applications is unsigned char, like for counter value.
- C compilers use the signed char as the default unless we put the keyword unsigned in front of the char.



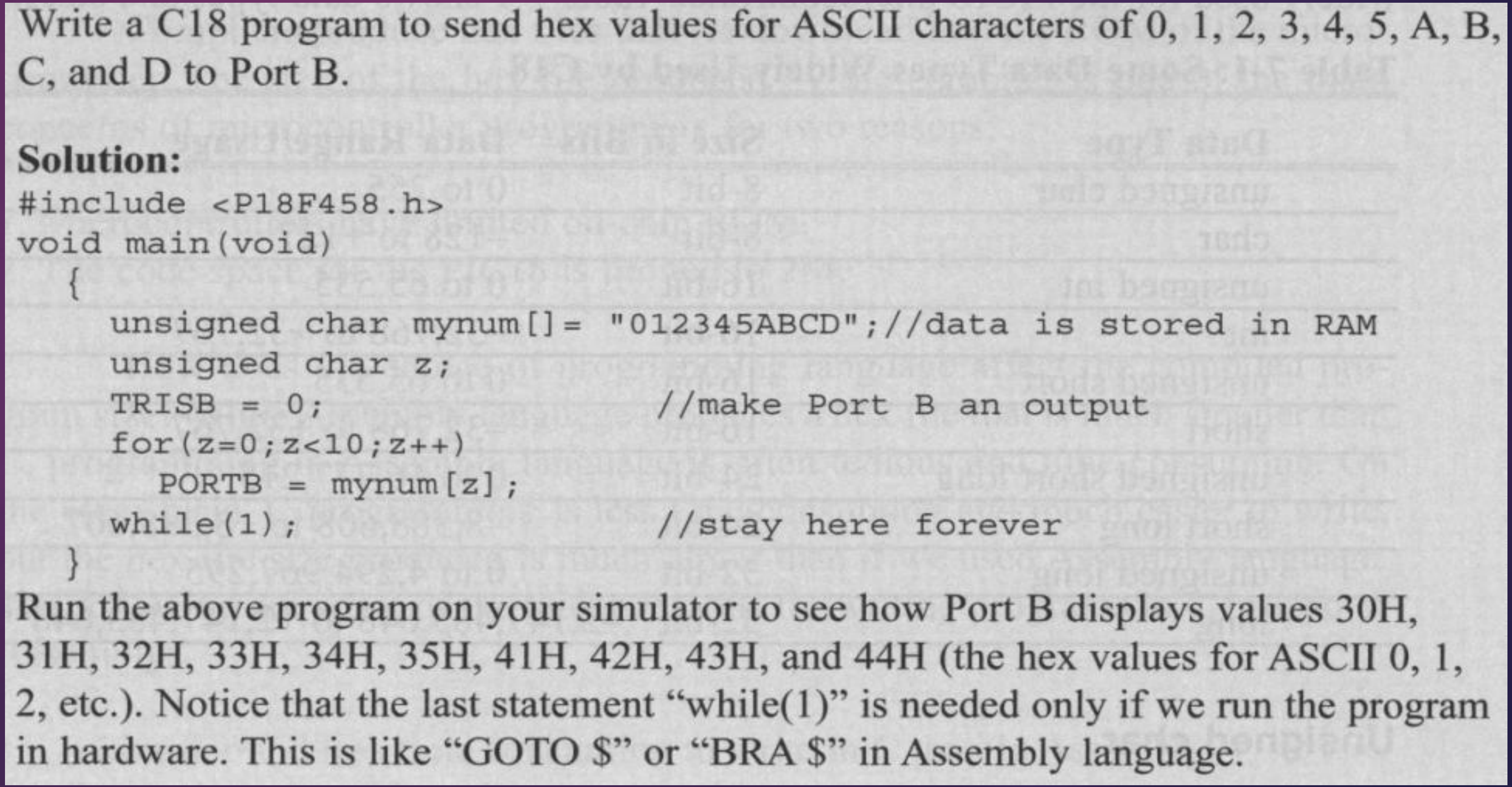
| Data Type | Size in Bits | Data Range/Usage |
|---|---|---|
| unsigned char | 8-bit | 0 to 255 |
| char | 8-bit | –128 to +127 |
| unsigned int | 16-bit | 0 to 65,535 |
| int | 16-bit | –32,768 to +32,767 |
| unsigned short | 16-bit | 0 to 65,535 |
| short | 16-bit | –32,768 to +32,767 |
| unsigned short long | 24-bit | 0 to 16,777,215 |
| short long | 24-bit | –8,388,608 to +8,388,607 |
| unsigned long | 32-bit | 0 to 4,294,967,295 |
| long | 32-bit | –2,147,483,648 to +2,147,483,648 |



Write a C18 program to send values 00–FF to Port B.

**Solution:**

```
#include <P18F458.h>          //for TRISB and PORTB declarations
void main(void)
  {
    unsigned char z;
    TRISB = 0;                //make Port B an output
    for(z=0;z<=255;z++)
      PORTB = z;
    while(1);                 //NEEDED IF RUNNING IN HARDWARE
  }
```

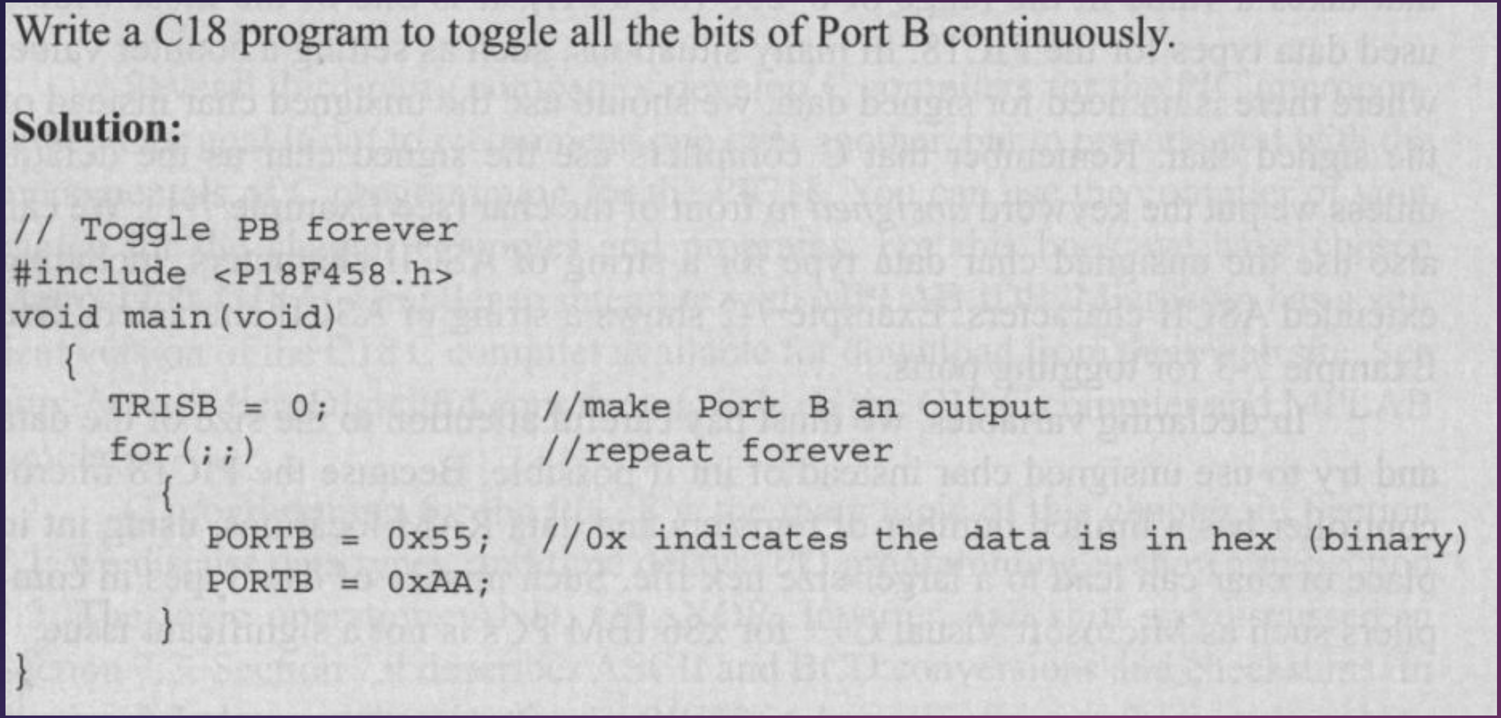


Write a C18 program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to Port B.

**Solution:**

```
#include <P18F458.h>
void main(void)
  {
    unsigned char mynum[]= "012345ABCD";//data is stored in RAM
    unsigned char z;
    TRISB = 0;                  //make Port B an output
    for(z=0;z<10;z++)
      PORTB = mynum[z];
    while(1);                   //stay here forever
  }
```

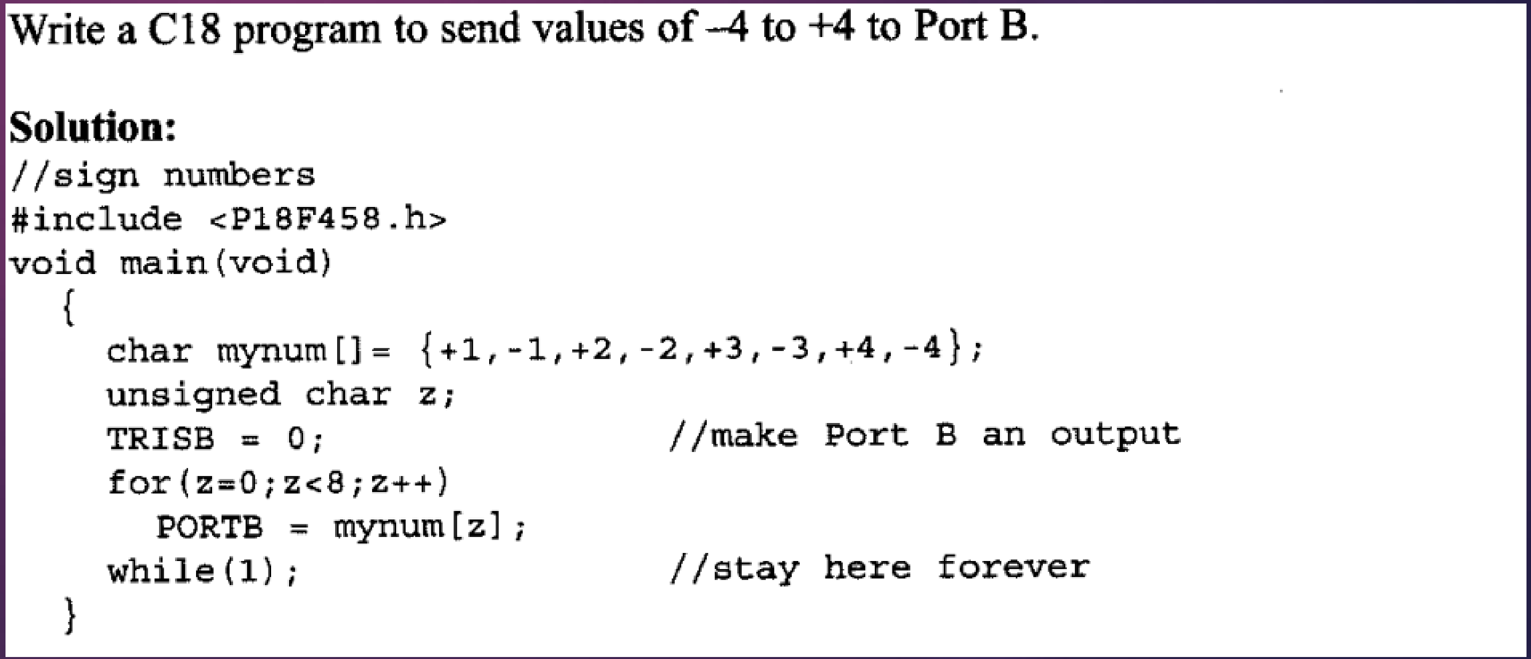
Run the above program on your simulator to see how Port B displays values 30H, 31H, 32H, 33H, 34H, 35H, 41H, 42H, 43H, and 44H (the hex values for ASCII 0, 1, 2, etc.). Notice that the last statement "while(1)" is needed only if we run the program in hardware. This is like "GOTO $" or "BRA $" in Assembly language.



Write a C18 program to toggle all the bits of Port B continuously.

**Solution:**

```
// Toggle PB forever
#include <P18F458.h>
void main(void)
  {
    TRISB = 0;          //make Port B an output
    for(;;)             //repeat forever
      {
        PORTB = 0x55;   //0x indicates the data is in hex (binary)
        PORTB = 0xAA;
      }
}
```



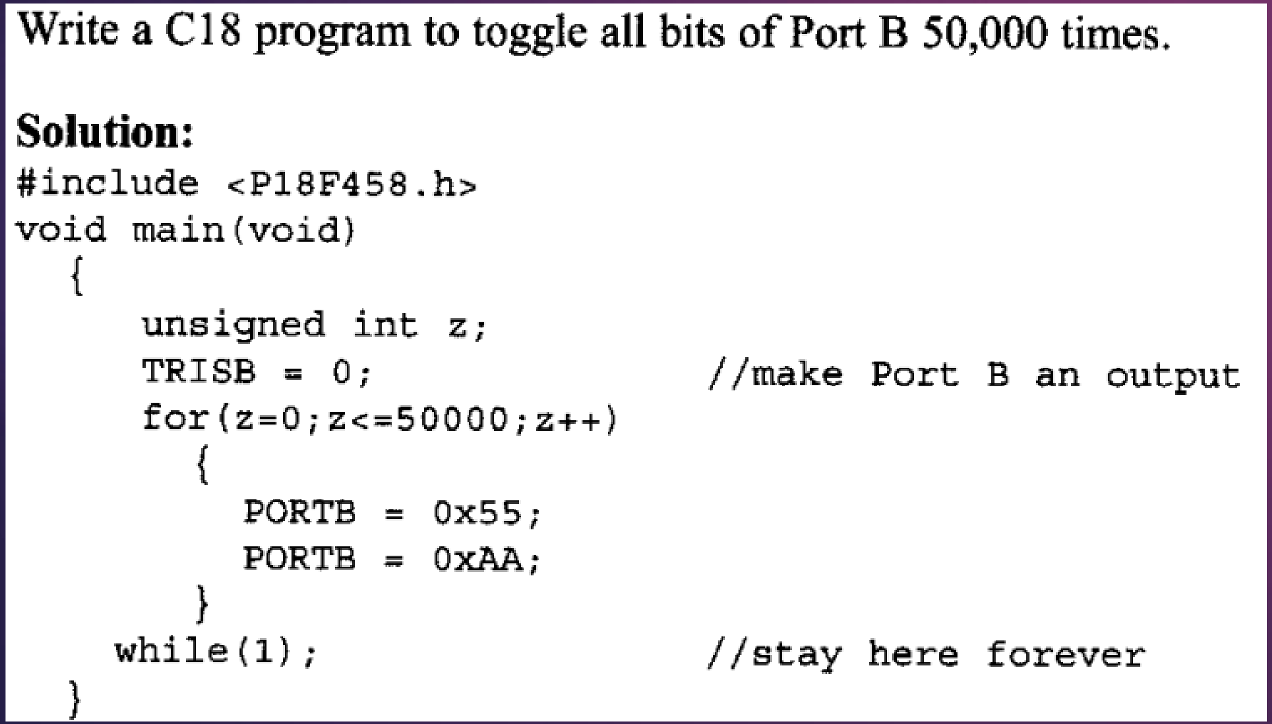
Write a C18 program to send values of –4 to +4 to Port B.

**Solution:**

```
//sign numbers
#include <P18F458.h>
void main(void)
  {
    char mynum[]= {+1,-1,+2,-2,+3,-3,+4,-4};
    unsigned char z;
    TRISB = 0;                  //make Port B an output
    for(z=0;z<8;z++)
      PORTB = mynum[z];
    while(1);                   //stay here forever
  }
```

# Unsigned int

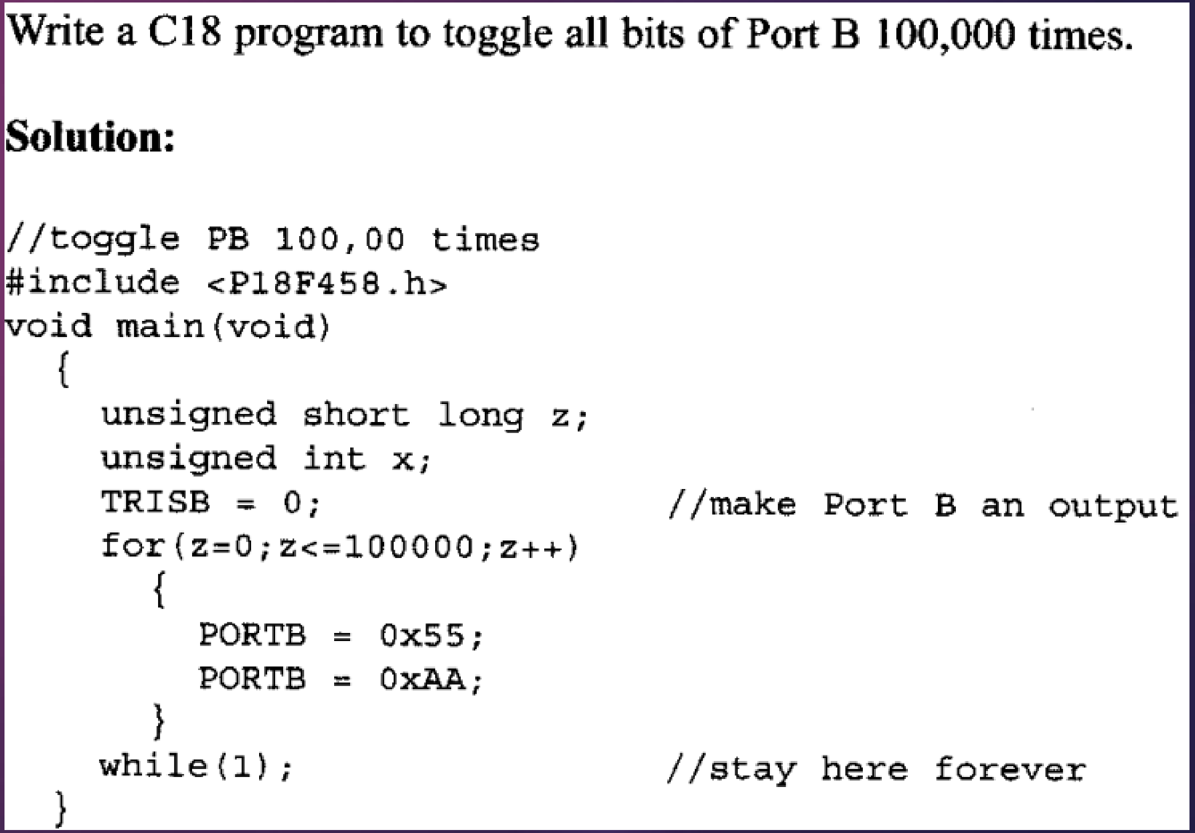
- It is 16 bit data type and is used to define 16 bit variables such as memory addresses.
- It is also used to set counter values of more than 256.
- It takes 2 bytes of RAM.



Write a C18 program to toggle all bits of Port B 50,000 times.

**Solution:**

```
#include <P18F458.h>
void main(void)
  {
      unsigned int z;
      TRISB = 0;               //make Port B an output
      for(z=0;z<=50000;z++)
        {
          PORTB = 0x55;
          PORTB = 0xAA;
        }
     while(1);                 //stay here forever
  }
```



Write a C18 program to toggle all bits of Port B 100,000 times.

**Solution:**

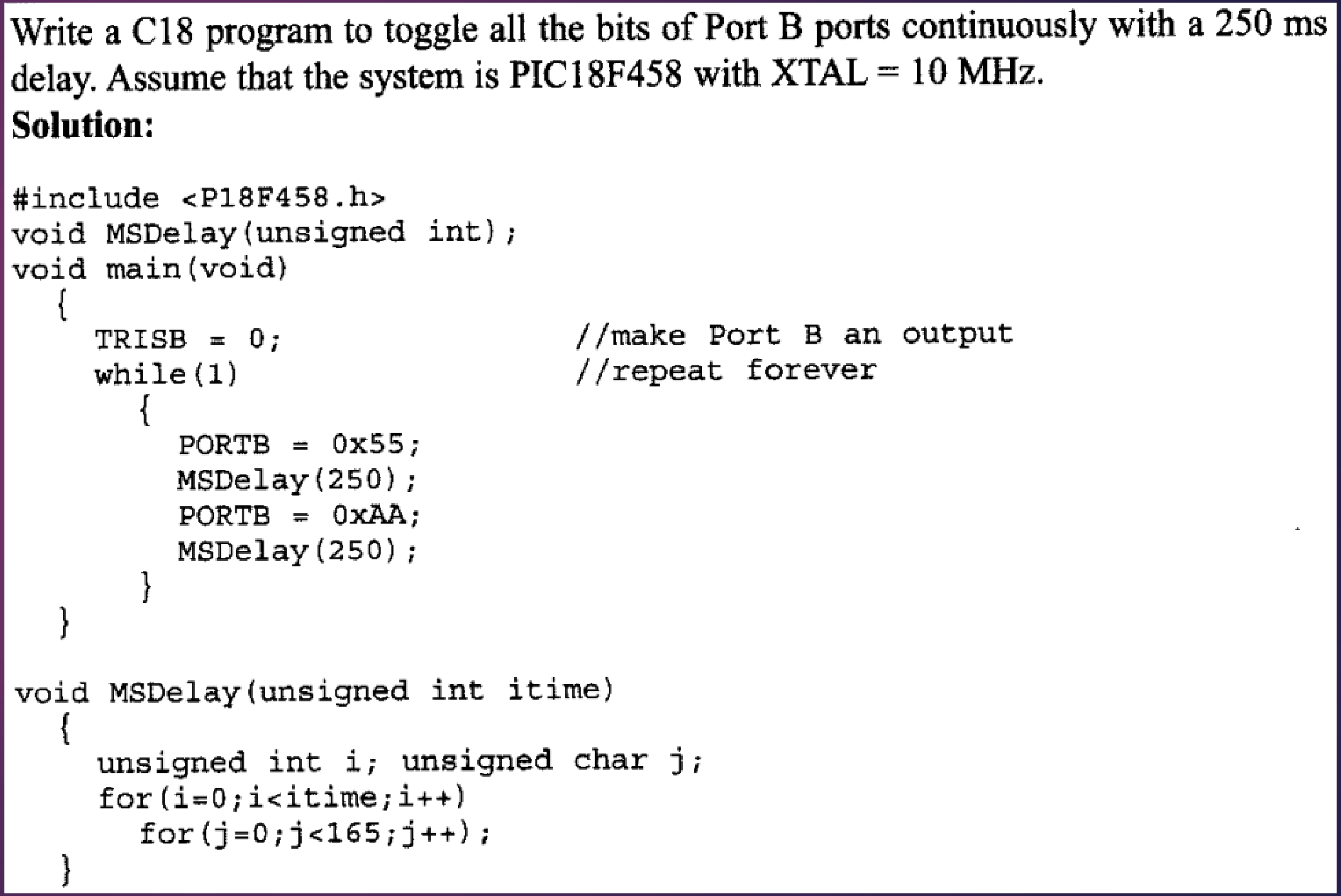
```
//toggle PB 100,00 times
#include <P18F458.h>
void main(void)
  {
     unsigned short long z;
     unsigned int x;
     TRISB = 0;               //make Port B an output
     for(z=0;z<=100000;z++)
       {
         PORTB = 0x55;
         PORTB = 0xAA;
       }
     while(1);                //stay here forever
  }
```

## Time Delay in C

- 2 ways to create a time delay in C:

1. Using a simple for loop.
2. Using the PIC18 timers.

- In either case, when we write a time delay we must use the oscilloscope to measure the duration of our time delay.

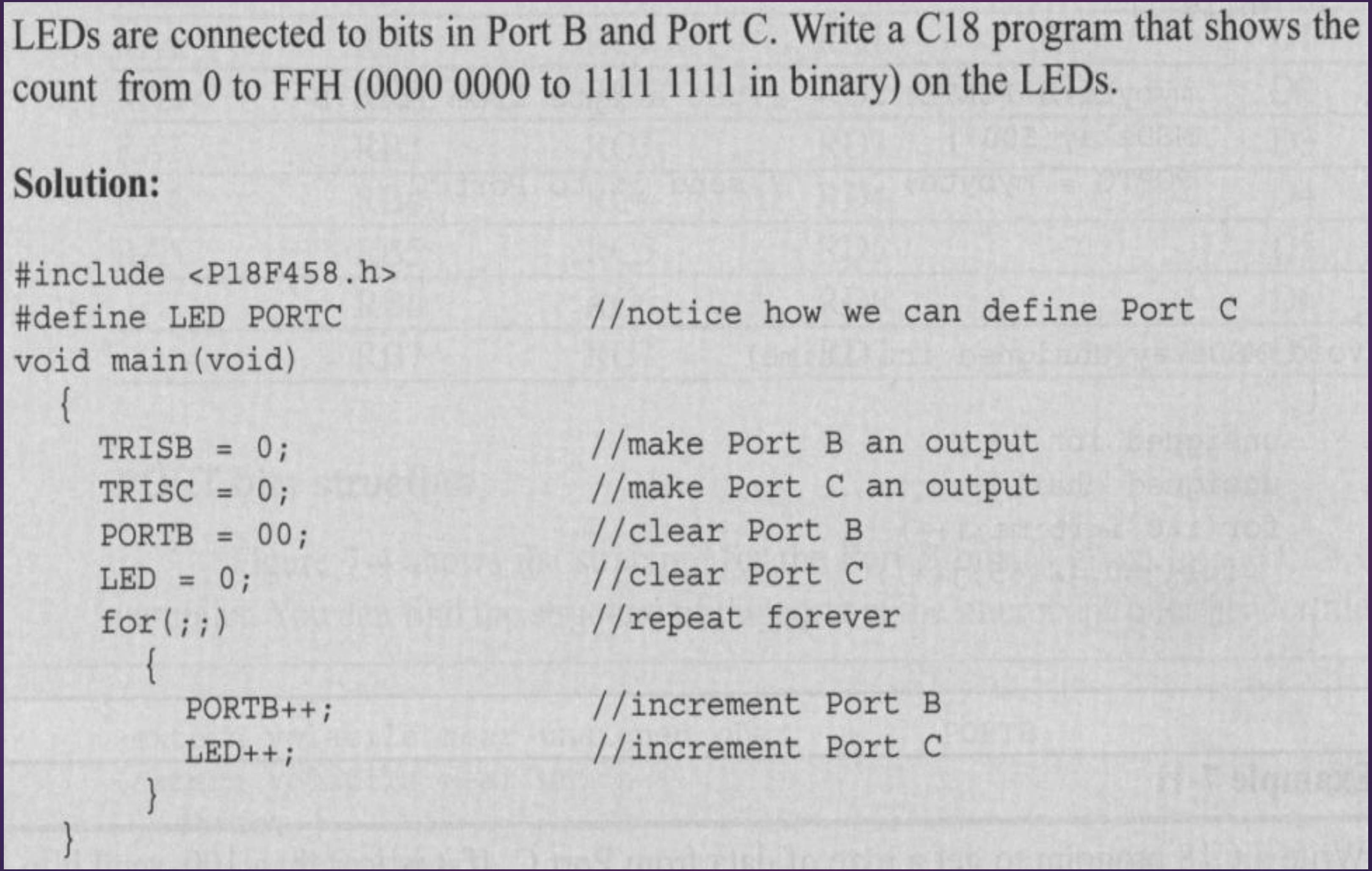


Write a C18 program to toggle all the bits of Port B ports continuously with a 250 ms delay. Assume that the system is PIC18F458 with XTAL = 10 MHz.

**Solution:**

```
#include <P18F458.h>
void MSDelay(unsigned int);
void main(void)
  {
    TRISB = 0;                    //make Port B an output
    while(1)                      //repeat forever
      {
        PORTB = 0x55;
        MSDelay(250);
        PORTB = 0xAA;
        MSDelay(250);
      }
  }

void MSDelay(unsigned int itime)
  {
    unsigned int i; unsigned char j;
    for(i=0;i<itime;i++)
      for(j=0;j<165;j++);
  }
```

# I/O Programming in C

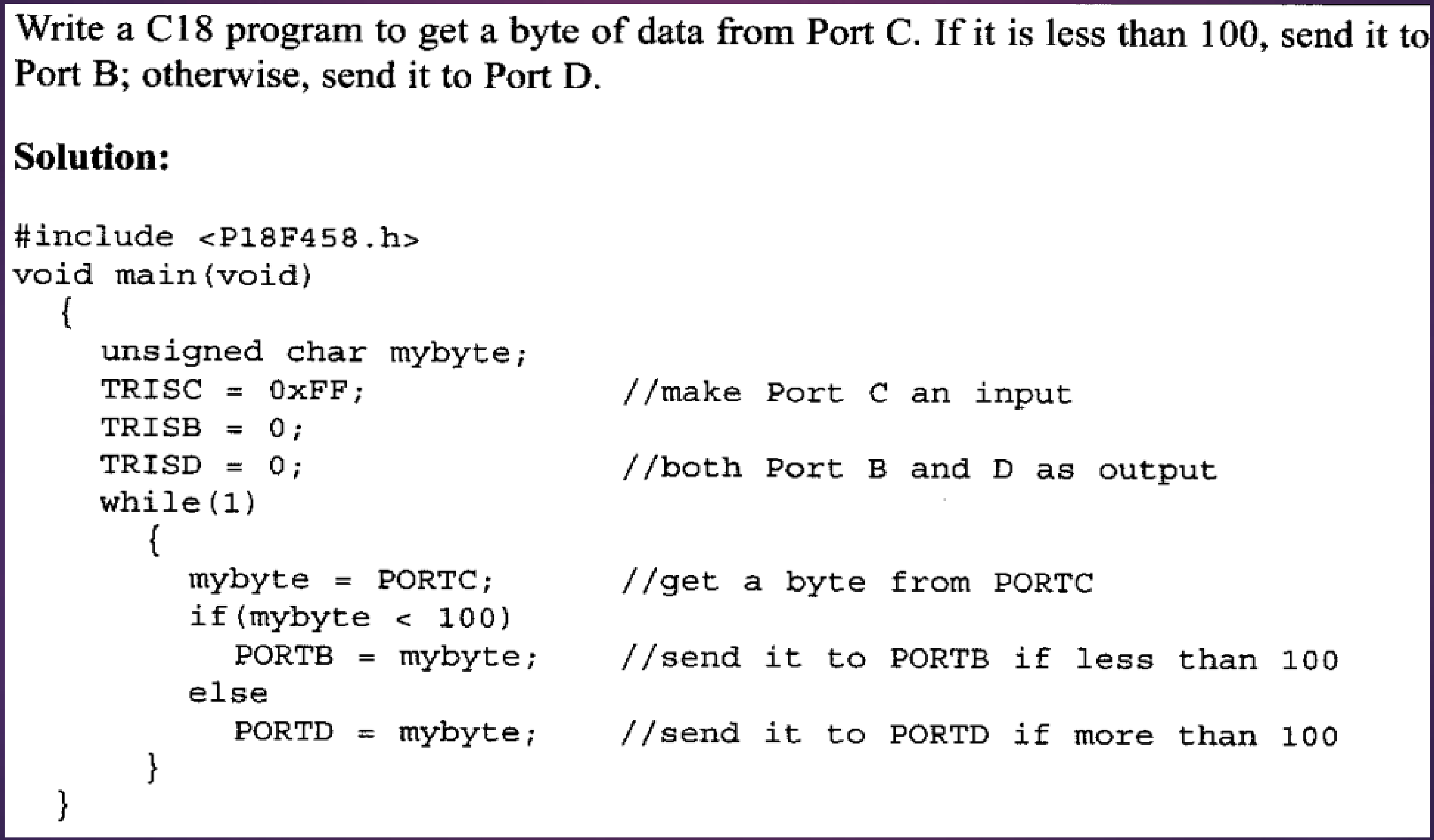


LEDs are connected to bits in Port B and Port C. Write a C18 program that shows the count from 0 to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

**Solution:**

```
#include <P18F458.h>
#define LED PORTC                   //notice how we can define Port C
void main(void)
  {
    TRISB = 0;                      //make Port B an output
    TRISC = 0;                      //make Port C an output
    PORTB = 00;                     //clear Port B
    LED = 0;                        //clear Port C
    for(;;)                         //repeat forever
      {
        PORTB++;                    //increment Port B
        LED++;                      //increment Port C
      }
  }
```

# Example



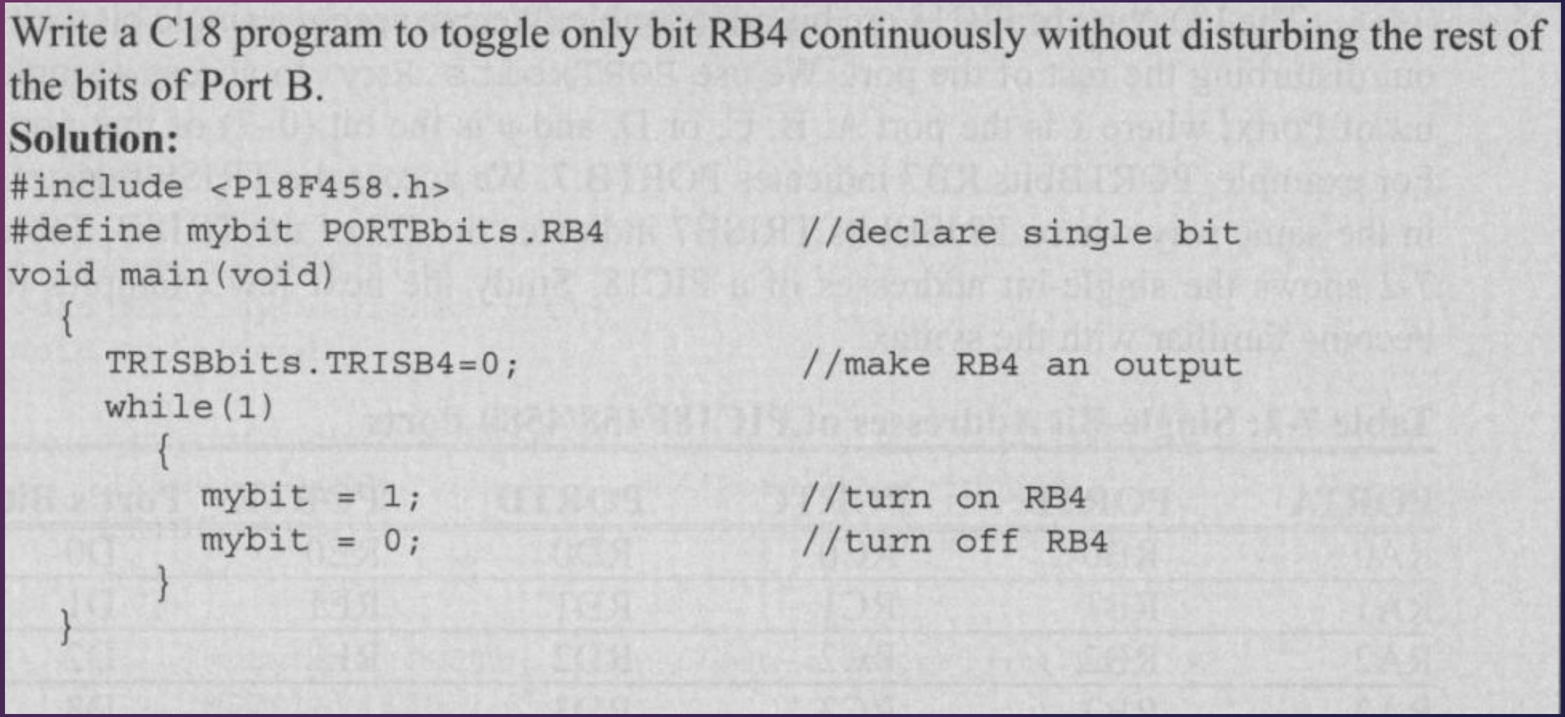
Write a C18 program to get a byte of data from Port C. If it is less than 100, send it to Port B; otherwise, send it to Port D.

**Solution:**

```
#include <P18F458.h>
void main(void)
  {
    unsigned char mybyte;
    TRISC = 0xFF;               //make Port C an input
    TRISB = 0;
    TRISD = 0;                  //both Port B and D as output
    while(1)
      {
        mybyte = PORTC;         //get a byte from PORTC
        if(mybyte < 100)
          PORTB = mybyte;       //send it to PORTB if less than 100
        else
          PORTD = mybyte;       //send it to PORTD if more than 100
      }
  }
```

## Bit addressable I/O Programming

- We use PORTxbits . Rxy to access a single bit of Portx, where x is the port A.,B,C or D and y is the (0-7) bit of that port.
- Example: PORTBbits . RB7 indicates PORTB.7.
- We access the TRIS registers in the same way where TRISBbits. TRISB7 indicates the D7 of the TRISB.

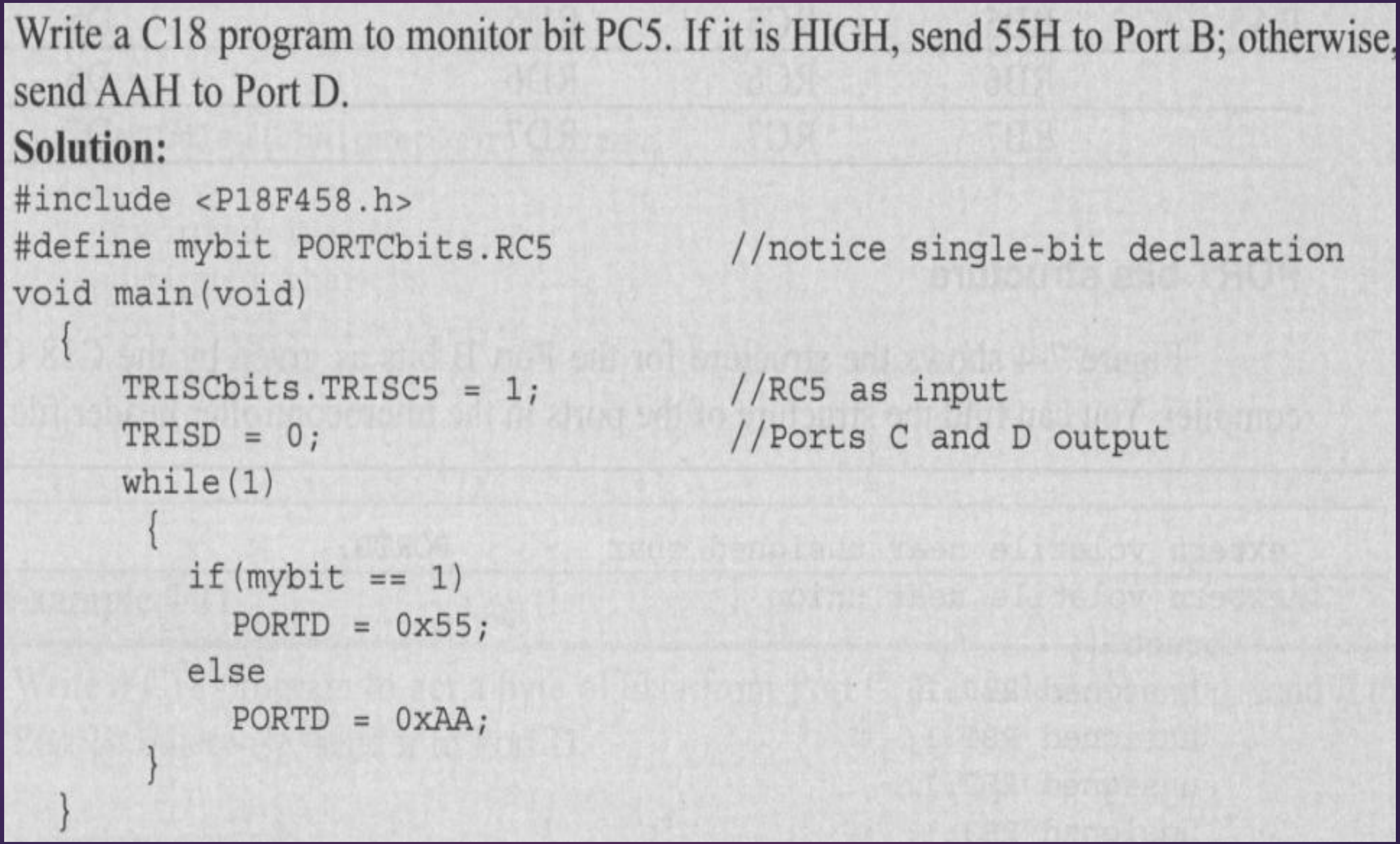


Write a C18 program to toggle only bit RB4 continuously without disturbing the rest of the bits of Port B.

**Solution:**

```
#include <P18F458.h>
#define mybit PORTBbits.RB4             //declare single bit
void main(void)
  {
    TRISBbits.TRISB4=0;                 //make RB4 an output
    while(1)
      {
        mybit = 1;                      //turn on RB4
        mybit = 0;                      //turn off RB4
      }
  }
```

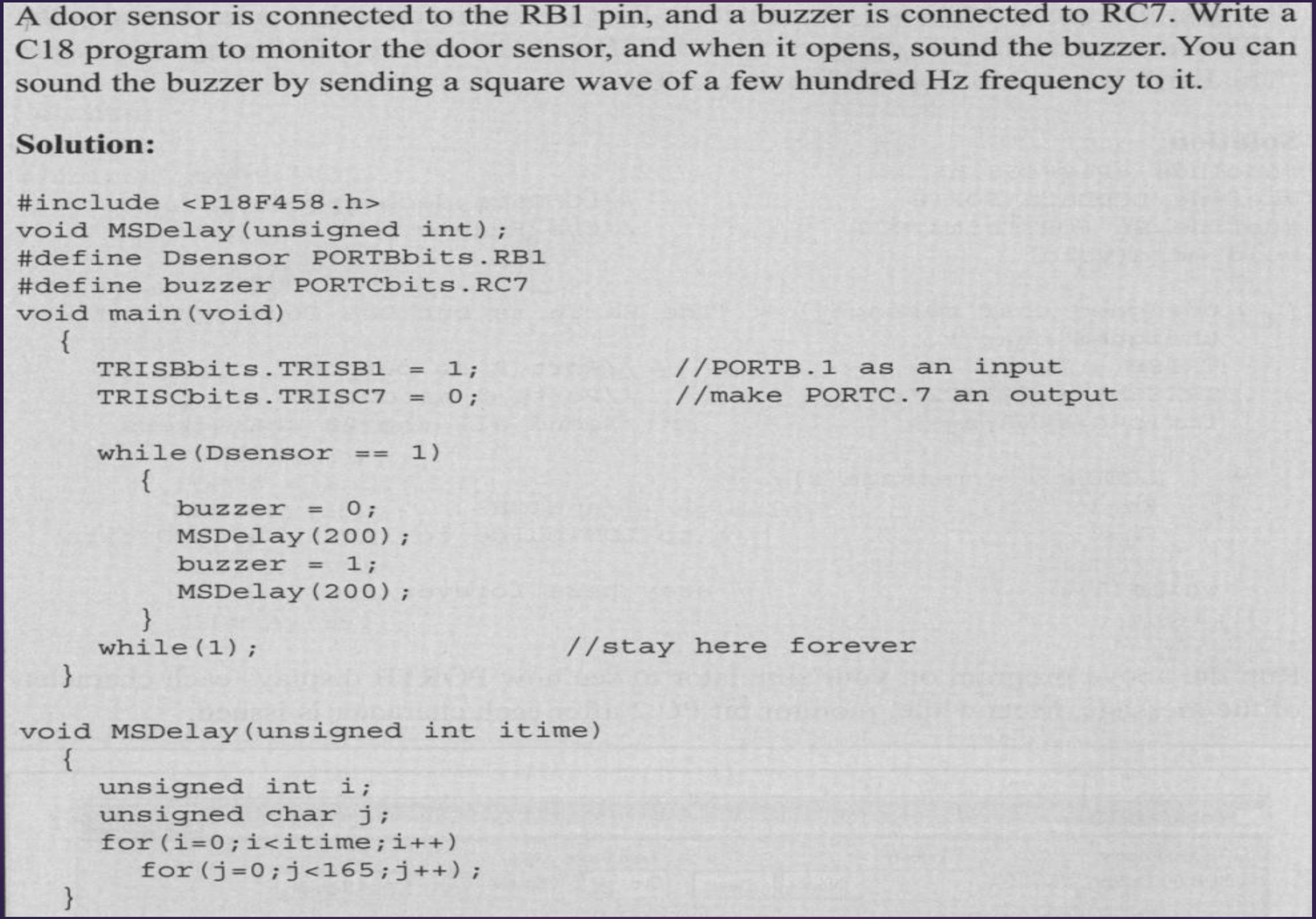
## Examples



Write a C18 program to monitor bit PC5. If it is HIGH, send 55H to Port B; otherwise, send AAH to Port D.

**Solution:**

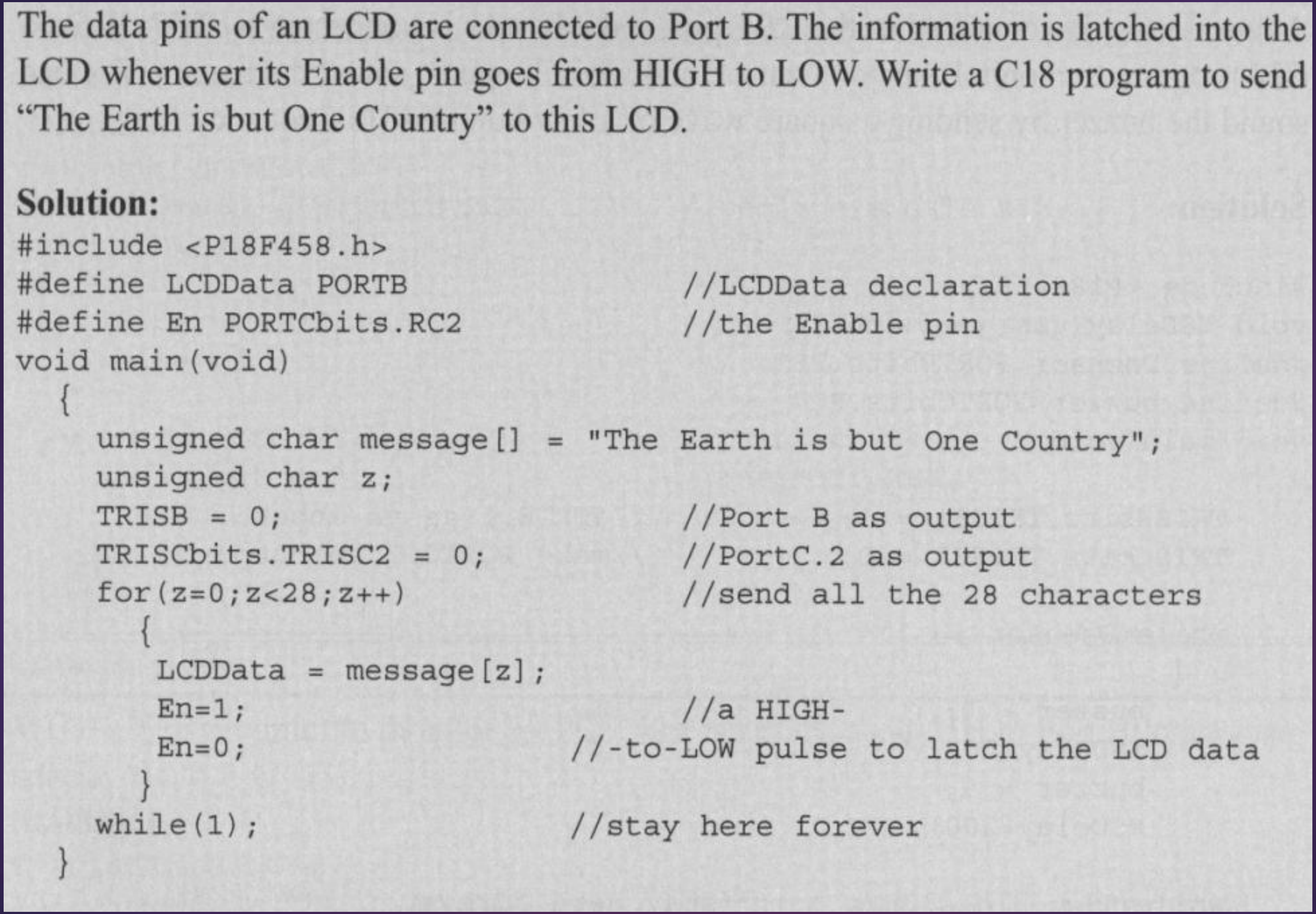
```
#include <P18F458.h>
#define mybit PORTCbits.RC5          //notice single-bit declaration
void main(void)
  {
     TRISCbits.TRISC5 = 1;           //RC5 as input
     TRISD = 0;                      //Ports C and D output
     while(1)
      {
        if(mybit == 1)
          PORTD = 0x55;
        else
          PORTD = 0xAA;
      }
  }
```



A door sensor is connected to the RB1 pin, and a buzzer is connected to RC7. Write a C18 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 frequency to it.

**Solution:**

```
#include <P18F458.h>
void MSDelay(unsigned int);
#define Dsensor PORTBbits.RB1
#define buzzer PORTCbits.RC7
void main(void)
  {
    TRISBbits.TRISB1 = 1;          //PORTB.1 as an input
    TRISCbits.TRISC7 = 0;          //make PORTC.7 an output

    while(Dsensor == 1)
      {
        buzzer = 0;
        MSDelay(200);
        buzzer = 1;
        MSDelay(200);
      }
    while(1);                    //stay here forever
  }

void MSDelay(unsigned int itime)
  {
    unsigned int i;
    unsigned char j;
    for(i=0;i<itime;i++)
      for(j=0;j<165;j++);
  }
```



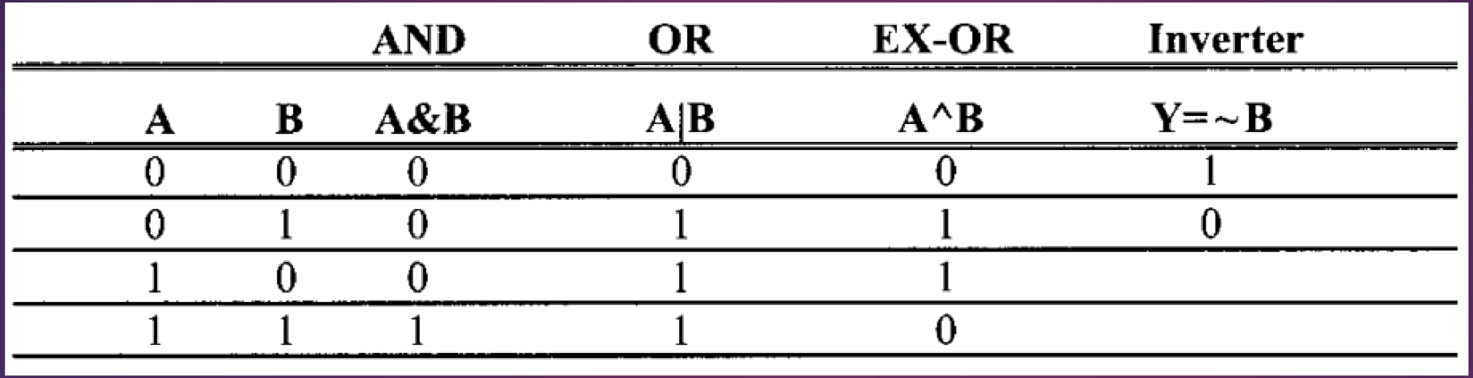
The data pins of an LCD are connected to Port B. The information is latched into the LCD whenever its Enable pin goes from HIGH to LOW. Write a C18 program to send "The Earth is but One Country" to this LCD.

**Solution:**

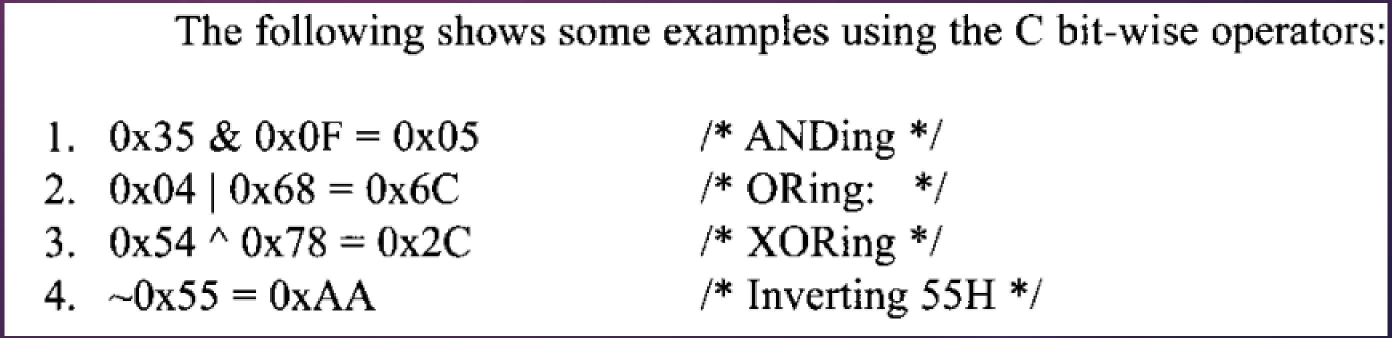
```
#include <P18F458.h>
#define LCDData PORTB                    //LCDData declaration
#define En PORTCbits.RC2                 //the Enable pin
void main(void)
  {
    unsigned char message[] = "The Earth is but One Country";
    unsigned char z;
    TRISB = 0;                           //Port B as output
    TRISCbits.TRISC2 = 0;                //PortC.2 as output
    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
      }
    while(1);                      //stay here forever
  }
```

# Logic Operations in C

**Bit wise operators are widely used in software engineering for embedded systems and control.**



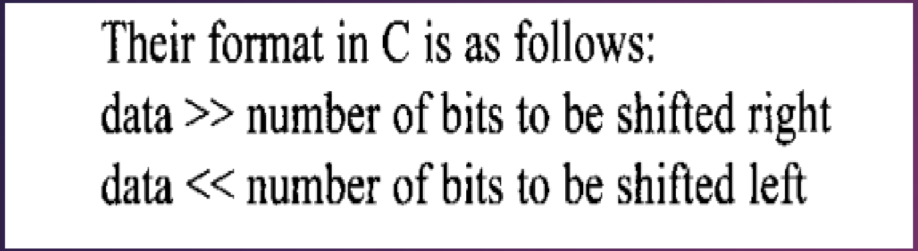
| | | AND | OR | EX-OR | Inverter |
|---|---|---|---|---|---|
| A | B | A&B | A\|B | A^B | Y=~B |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 1 | 0 | |



The following shows some examples using the C bit-wise operators:

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

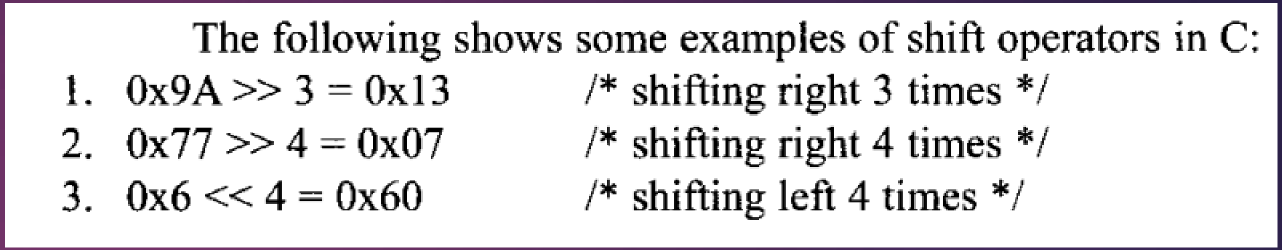
# Bit-wise shift operation in C



Their format in C is as follows:

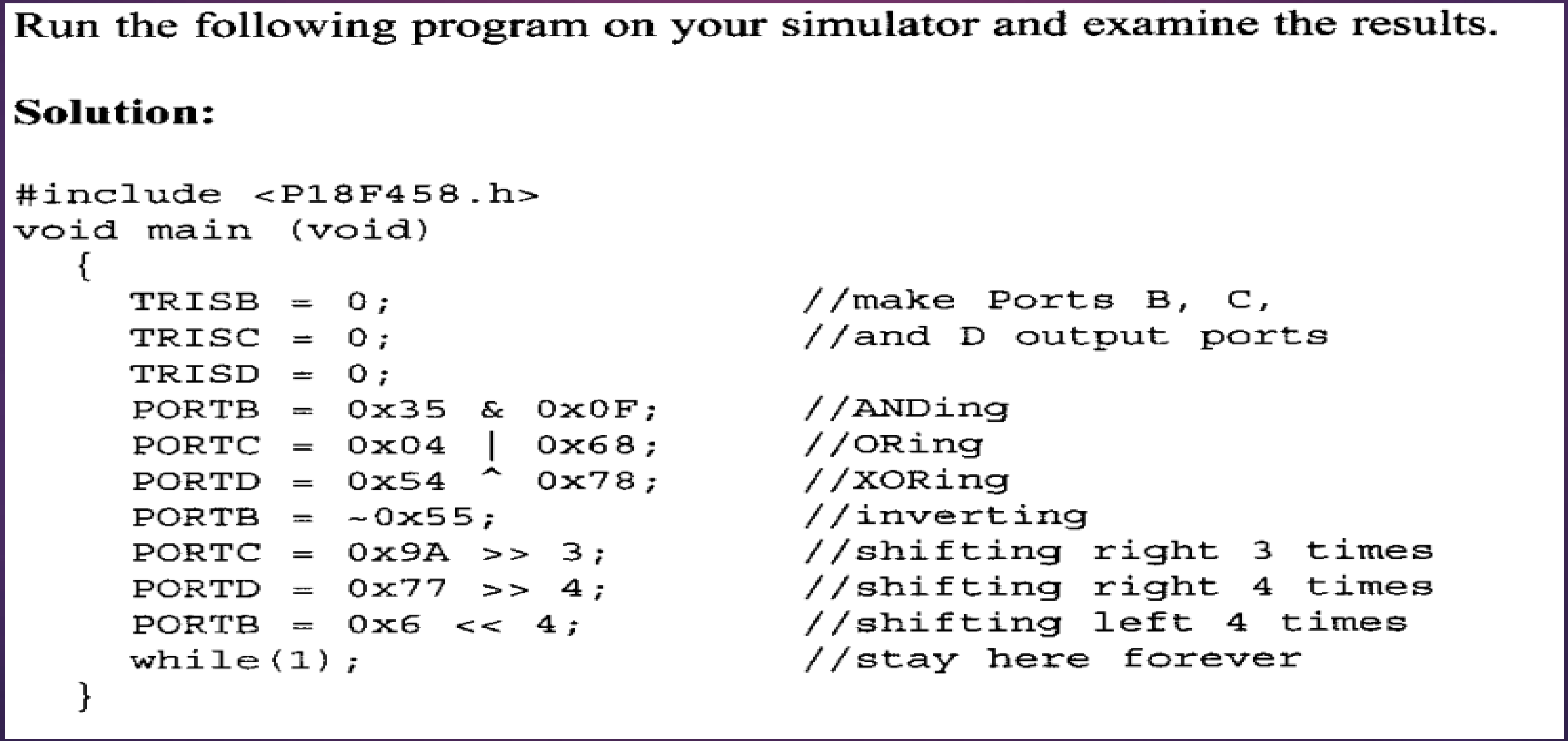
data >> number of bits to be shifted right

data << number of bits to be shifted left



The following shows some examples of shift operators in C:

1. 0x9A >> 3 = 0x13 /* shifting right 3 times */
2. 0x77 >> 4 = 0x07 /* shifting right 4 times */
3. 0x6 << 4 = 0x60 /* shifting left 4 times */

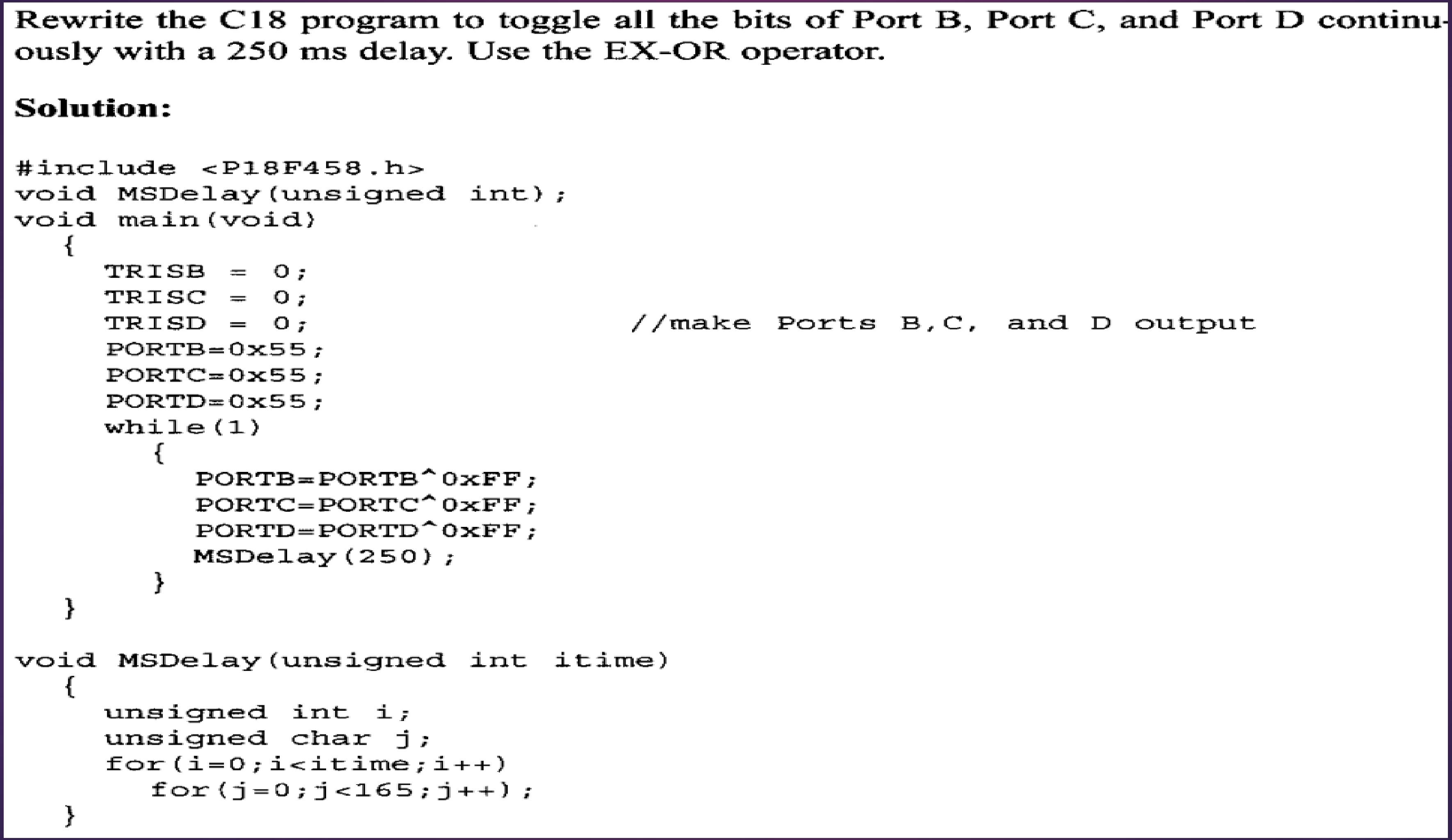


**Run the following program on your simulator and examine the results.**

**Solution:**

```
#include <P18F458.h>
void main (void)
  {
    TRISB = 0;                        //make Ports B, C,
    TRISC = 0;                        //and D output ports
    TRISD = 0;
    PORTB = 0x35 & 0x0F;              //ANDing
    PORTC = 0x04 | 0x68;              //ORing
    PORTD = 0x54 ^ 0x78;              //XORing
    PORTB = ~0x55;                    //inverting
    PORTC = 0x9A >> 3;                //shifting right 3 times
    PORTD = 0x77 >> 4;                //shifting right 4 times
    PORTB = 0x6 << 4;                 //shifting left 4 times
    while(1);                         //stay here forever
  }
```

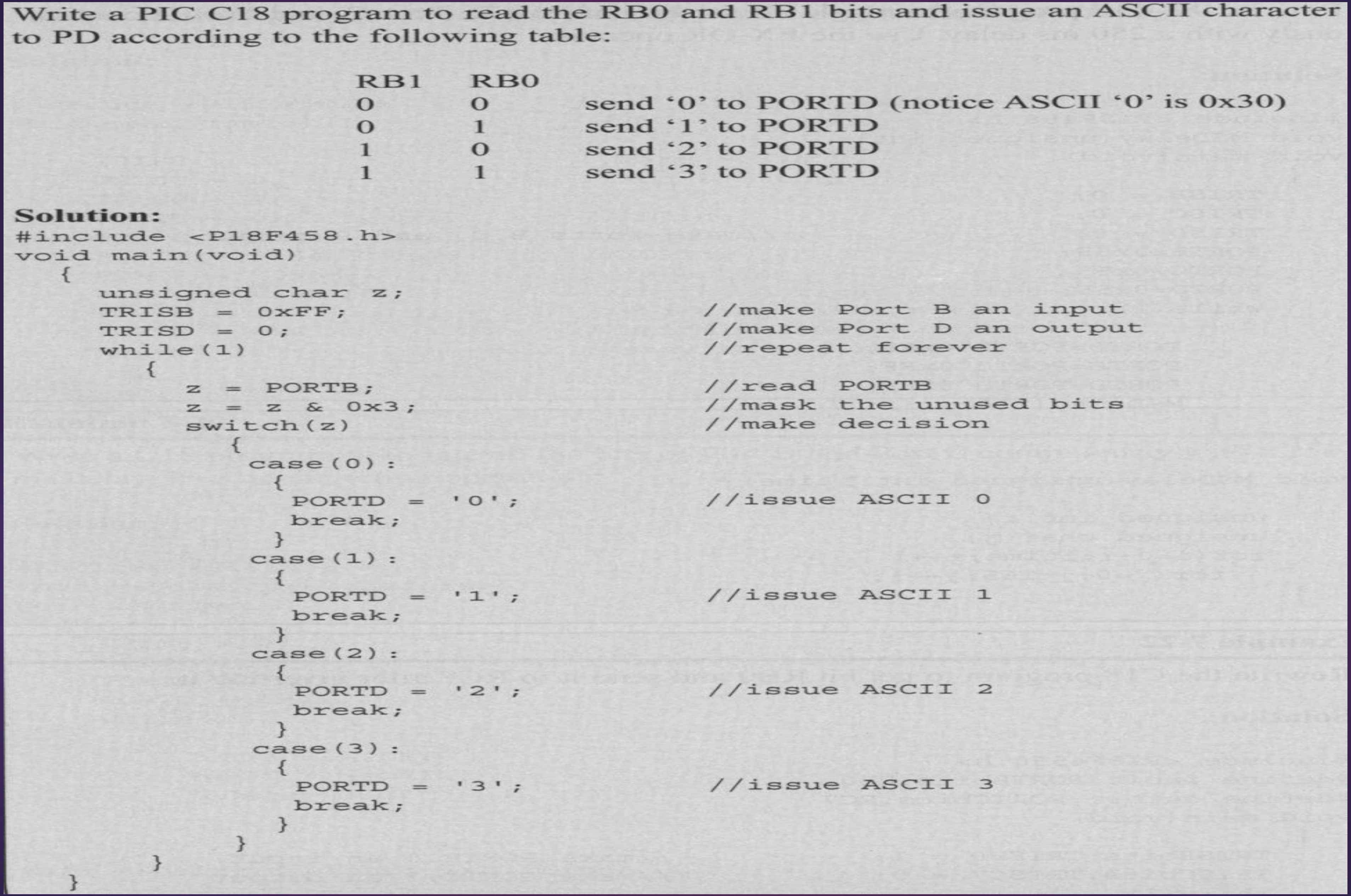
# Example



Rewrite the C18 program to toggle all the bits of Port B, Port C, and Port D continuously with a 250 ms delay. Use the EX-OR operator.

**Solution:**

```
#include <P18F458.h>
void MSDelay(unsigned int);
void main(void)
  {
     TRISB = 0;
     TRISC = 0;
     TRISD = 0;                     //make Ports B,C, and D output
     PORTB=0x55;
     PORTC=0x55;
     PORTD=0x55;
     while(1)
       {
         PORTB=PORTB^0xFF;
         PORTC=PORTC^0xFF;
         PORTD=PORTD^0xFF;
         MSDelay(250);
       }
  }

void MSDelay(unsigned int itime)
  {
     unsigned int i;
     unsigned char j;
     for(i=0;i<itime;i++)
       for(j=0;j<165;j++);
  }
```



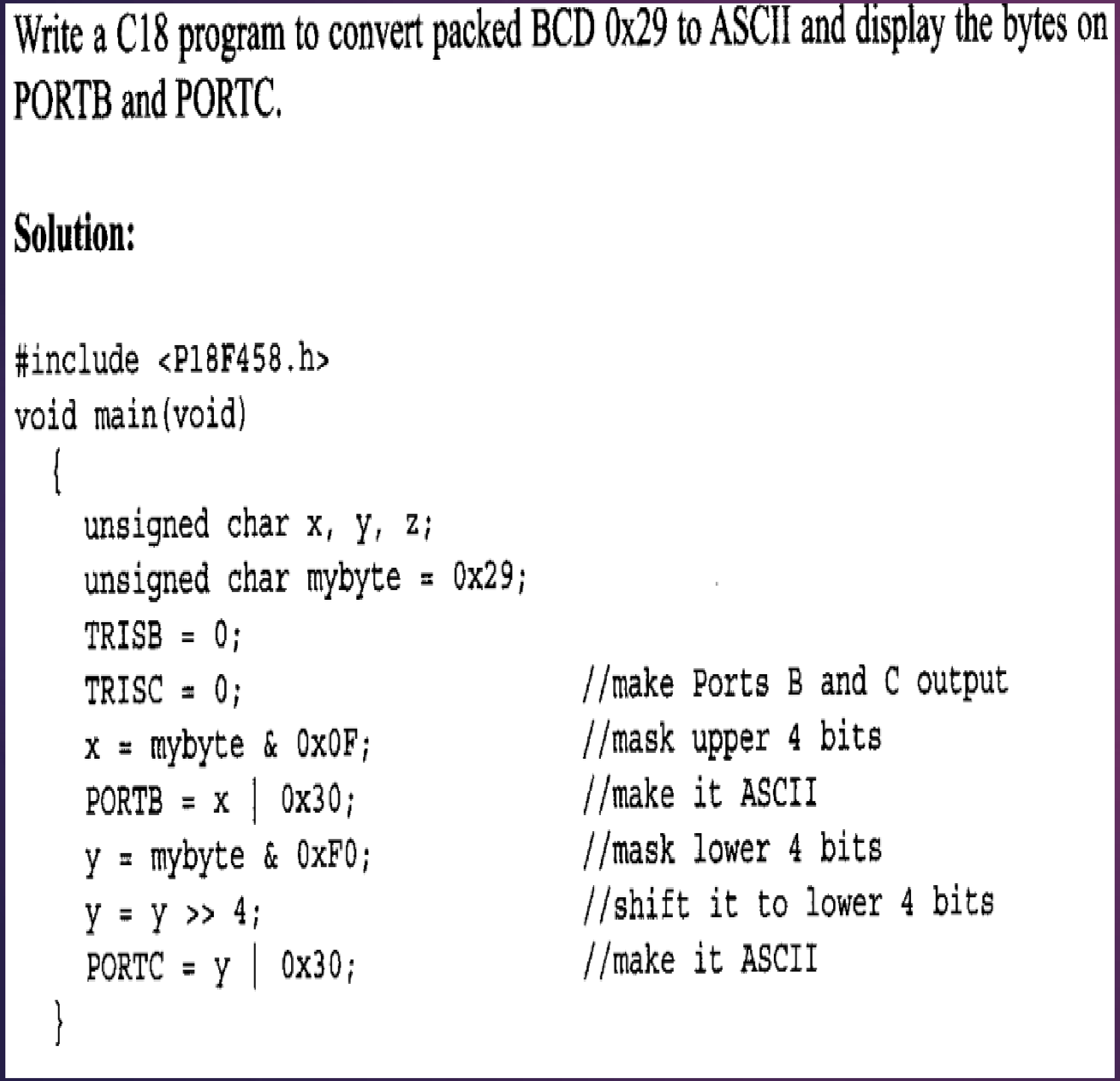
Write a PIC C18 program to read the RB0 and RB1 bits and issue an ASCII character to PD according to the following table:

| RB1 | RB0 | |
|---|---|---|
| 0 | 0 | send '0' to PORTD (notice ASCII '0' is 0x30) |
| 0 | 1 | send '1' to PORTD |
| 1 | 0 | send '2' to PORTD |
| 1 | 1 | send '3' to PORTD |

**Solution:**

```
#include <P18F458.h>
void main(void)
  {
    unsigned char z;
    TRISB = 0xFF;                       //make Port B an input
    TRISD = 0;                          //make Port D an output
    while(1)                            //repeat forever
      {
        z = PORTB;                      //read PORTB
        z = z & 0x3;                    //mask the unused bits
        switch(z)                       //make decision
          {
           case(0):
             {
              PORTD = '0';              //issue ASCII 0
              break;
             }
           case(1):
             {
              PORTD = '1';              //issue ASCII 1
              break;
             }
           case(2):
             {
              PORTD = '2';              //issue ASCII 2
              break;
             }
           case(3):
             {
              PORTD = '3';              //issue ASCII 3
              break;
             }
          }
      }
  }
```

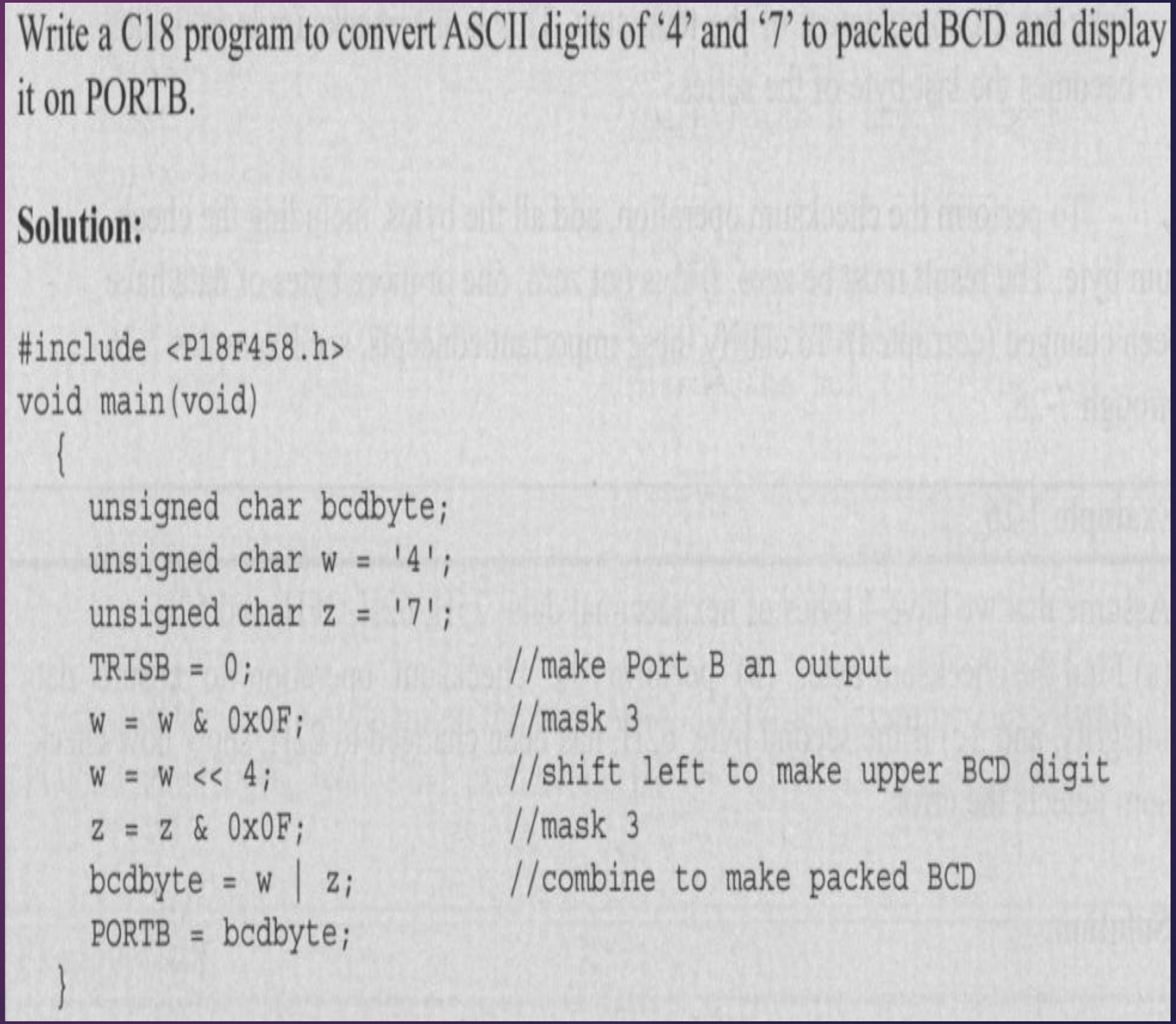
# Data Conversion Programs in C



Write a C18 program to convert packed BCD 0x29 to ASCII and display the bytes on PORTB and PORTC.

**Solution:**

```
#include <P18F458.h>
void main(void)
  {
    unsigned char x, y, z;
    unsigned char mybyte = 0x29;
    TRISB = 0;
    TRISC = 0;                      //make Ports B and C output
    x = mybyte & 0x0F;              //mask upper 4 bits
    PORTB = x | 0x30;               //make it ASCII
    y = mybyte & 0xF0;              //mask lower 4 bits
    y = y >> 4;                     //shift it to lower 4 bits
    PORTC = y | 0x30;               //make it ASCII
  }
```

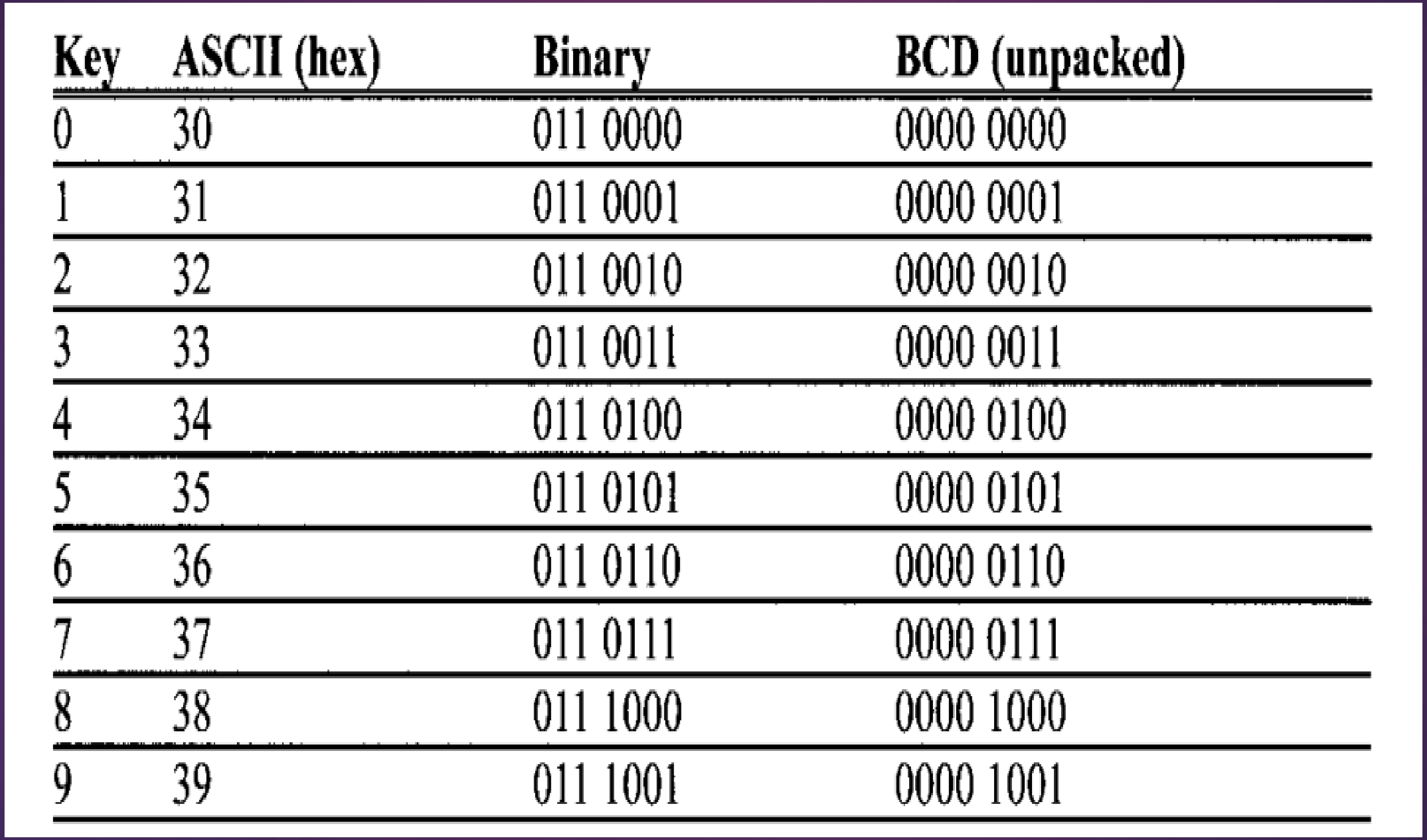


Write a C18 program to convert ASCII digits of '4' and '7' to packed BCD and display it on PORTB.

**Solution:**

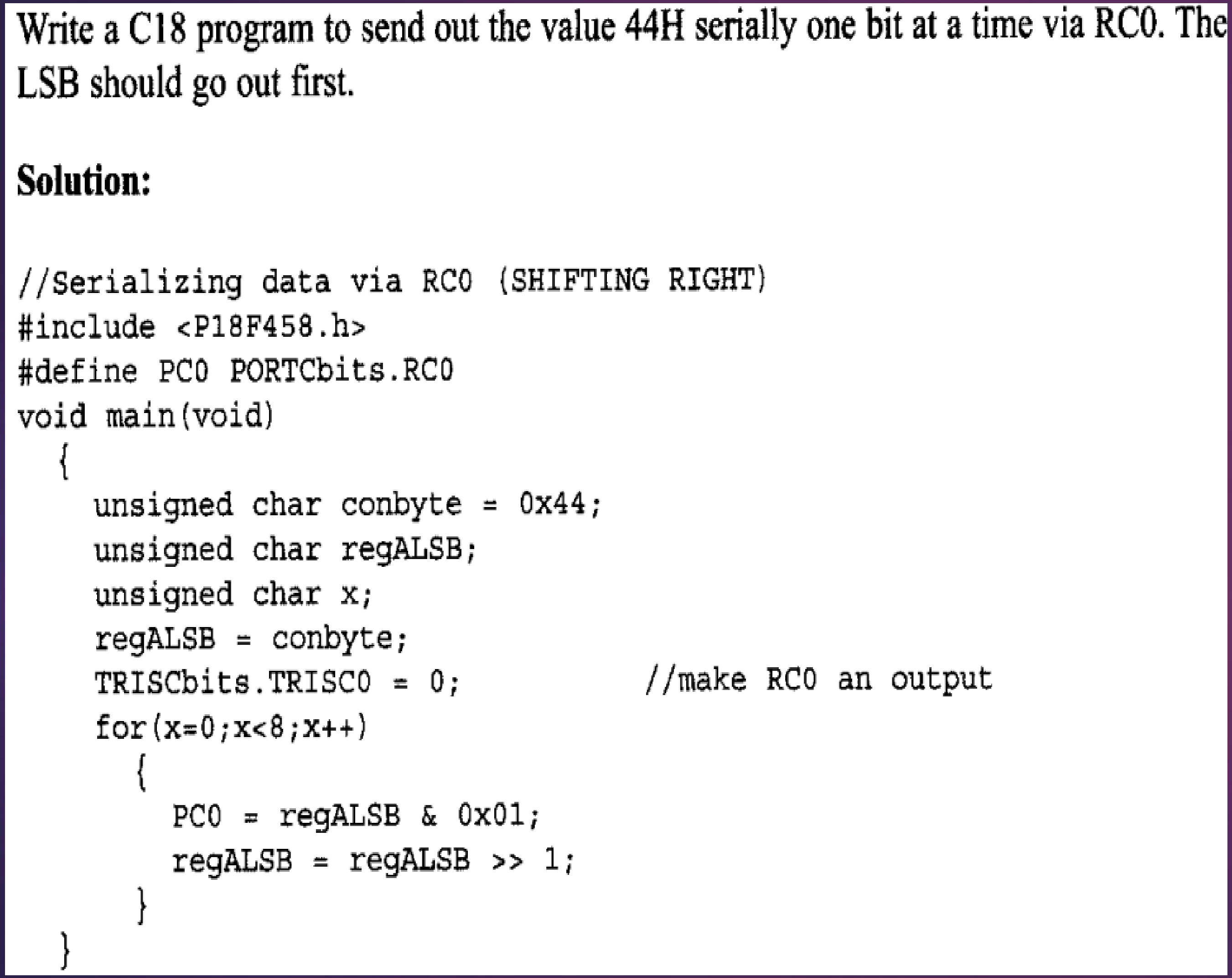
```
#include <P18F458.h>
void main(void)
  {
    unsigned char bcdbyte;
    unsigned char w = '4';
    unsigned char z = '7';
    TRISB = 0;                 //make Port B an output
    w = w & 0x0F;              //mask 3
    w = w << 4;                //shift left to make upper BCD digit
    z = z & 0x0F;              //mask 3
    bcdbyte = w | z;           //combine to make packed BCD
    PORTB = bcdbyte;
  }
```

## ASCII codes 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 |

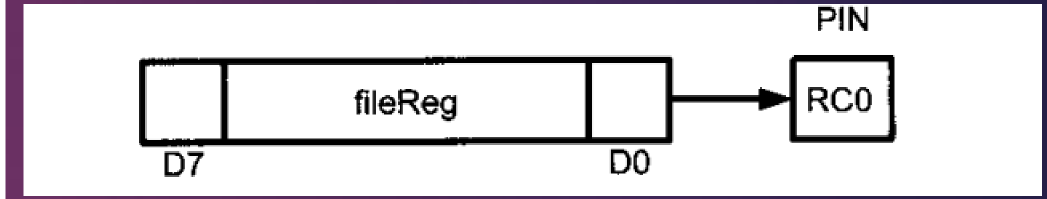
# Data Serialization in C



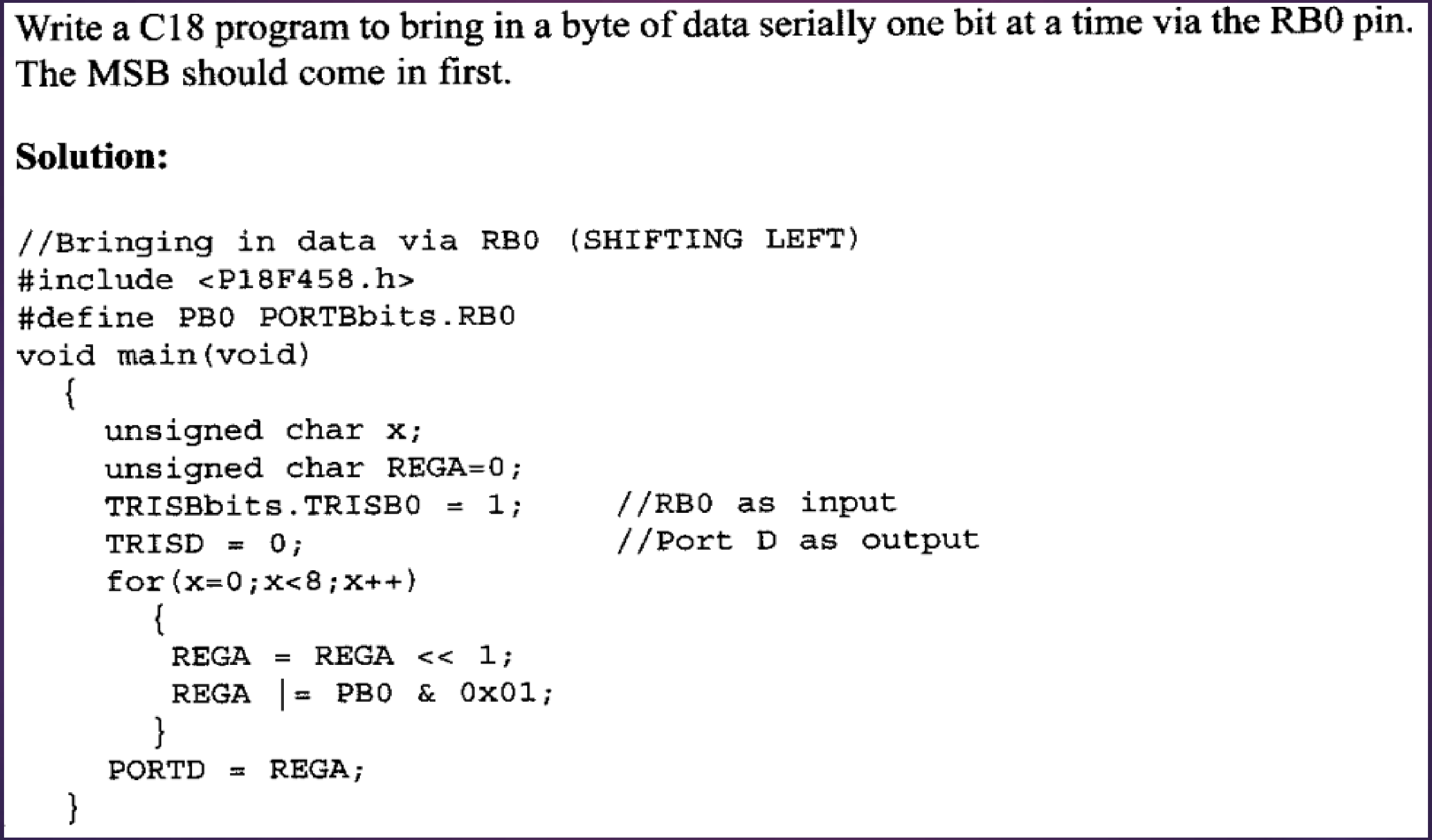
Write a C18 program to send out the value 44H serially one bit at a time via RC0. The LSB should go out first.

**Solution:**

```
//Serializing data via RC0 (SHIFTING RIGHT)
#include <P18F458.h>
#define PC0 PORTCbits.RC0
void main(void)
  {
    unsigned char conbyte = 0x44;
    unsigned char regALSB;
    unsigned char x;
    regALSB = conbyte;
    TRISCbits.TRISC0 = 0;           //make RC0 an output
    for(x=0;x<8;x++)
      {
        PC0 = regALSB & 0x01;
        regALSB = regALSB >> 1;
      }
  }
```



# Example

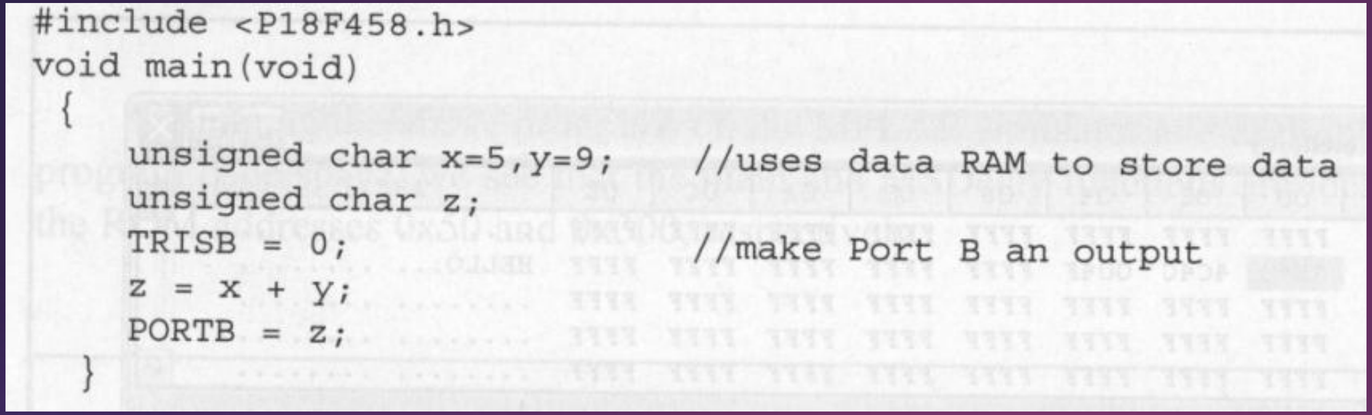


Write a C18 program to bring in a byte of data serially one bit at a time via the RB0 pin. The MSB should come in first.

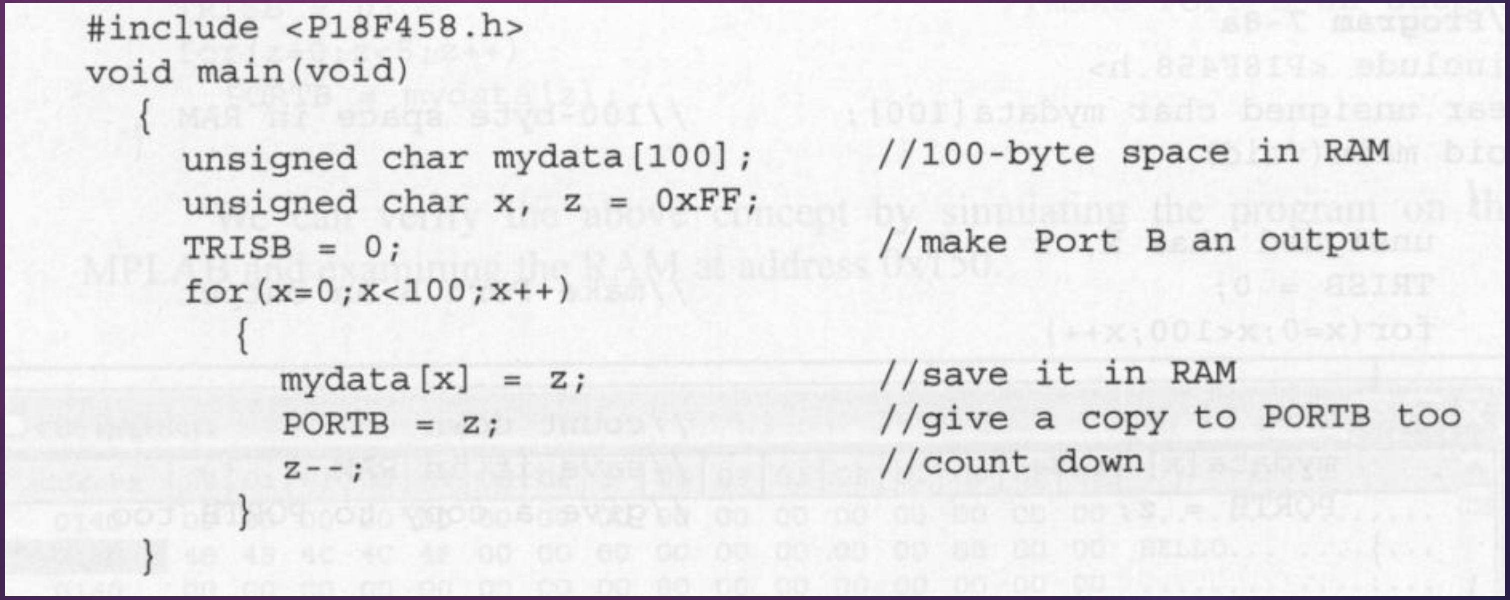
**Solution:**

```
//Bringing in data via RB0 (SHIFTING LEFT)
#include <P18F458.h>
#define PB0 PORTBbits.RB0
void main(void)
  {
     unsigned char x;
     unsigned char REGA=0;
     TRISBbits.TRISB0 = 1;     //RB0 as input
     TRISD = 0;                //Port D as output
     for(x=0;x<8;x++)
       {
        REGA = REGA << 1;
        REGA |= PB0 & 0x01;
       }
     PORTD = REGA;
  }
```

# Data RAM Allocation in C



```
#include <P18F458.h>
void main(void)
 {
    unsigned char x=5,y=9;    //uses data RAM to store data
    unsigned char z;
    TRISB = 0;                //make Port B an output
    z = x + y;
    PORTB = z;
  }
```



```
#include <P18F458.h>
void main(void)
  {
    unsigned char mydata[100];     //100-byte space in RAM
    unsigned char x, z = 0xFF;
    TRISB = 0;                     //make Port B an output
    for(x=0;x<100;x++)
      {
        mydata[x] = z;             //save it in RAM
        PORTB = z;                 //give a copy to PORTB too
        z--;                       //count down
      }
  }
```