Chapter 9 – Embedded Systems

9.1. Connect character input to the serial port and the 7-segment display unit to parallel port A. Connect bits $PAOUT_6$ to $PAOUT_0$ to the display segments a to g, respectively. Use the segment encoding shown in Figure A.37. For example, the decimal digit 0 sets the segments a, b, ..., g to the hex pattern 7E.

A suitable program may use a table to convert the ASCII characters into the hex patterns for the display. The ASCII-encoded digits (see Table E.2) are represented by the pattern 111 in bit positions b_{6-4} and the corresponding BCD value (see Table E.1) in bit positions b_{3-0}. Hence, extracting the bits b_{3-0} from the ASCII code provides an index, j, which can be used to access the required entry in the conversion table (list). A possible program is is obtained by modifying the program in Figure 9.11 as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SSTAT (volatile char *) 0xFFFFFFE2
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2


void main()
{
    /* Define the conversion table */
    char seg7[10] = {0x7E, 0x30, 0x6C, 0x79, 0x33, 0x5B, 0x5F, 0x30, 0x3F, 0x3B};
    char j;
    /* Initialize the parallel port */
    *PADIR = 0xFF;                      /* Configure Port A as output */

    /* Transfer the characters */
    while (1) {                         /* Infinite loop */
        while ((*SSTAT & 0x1) == 0);    /* Wait for a new character */
        j = *RBUF & 0xF;                /* Extract the BCD value */
        *PAOUT = seg7[j];               /* Send the 7-segment code to Port A */
    }
}
```

9.2. The arrangement explained in the solution for Problem 9.1 can be used. The entries in the conversion table can be accessed using the indexed addressing mode. Let the table occupy ten bytes starting at address SEG7. Then, using register R0 as the index register, the table is accessed using the mode SEG7(R0). The desired program may be obtained by modifying the program in Figure 9.10 as follows:

```
RBUF    EQU         $FFFFFFE0       Receive buffer.
SSTAT   EQU         $FFFFFFE2       Status register for serial interface.
PAOUT   EQU         $FFFFFFF1       Port A output data.
PADIR   EQU         $FFFFFFF2       Port A direction register.

* Define the conversion table
        ORIGIN      $200
SEG7    DataByte    $7E, $30, $6C, $79, $33, $5B, $5F, $30, $3F, $3B

* Initialization
        ORIGIN      $1000
        MoveByte    #$FF,PADIR      Configure Port A as output.

* Transfer the characters
LOOP    Testbit     #0,SSTAT        Check if new character is ready.
        Branch=0    LOOP
        MoveByte    RBUF,R0         Transfer a character to R0.
        And         #$F,R0          Extract the BCD value.
        MoveByte    SEG7(R0),PAOUT  Send the 7-segment code to Port A.
        Branch      LOOP
```

9.3. The arrangement explained in the solution for Problem 9.1 can be used. The desired program may be obtained by modifying the program in Figure 9.16 as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SCONT (char *) 0xFFFFFFE3
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2
#define int_addr (int *) (0x24)


void intserv();

void main()
{
    /* Define the conversion table */
    char seg7[10] = {0x7E, 0x30, 0x6C, 0x79, 0x33, 0x5B, 0x5F, 0x30, 0x3F, 0x3B};
    char j;
    /* Initialize the parallel port */
    *PADIR = 0xFF;                  /* Configure Port A as output */

    /* Initialize the interrupt mechanism */
    int_addr = &intserv;            /* Set interrupt vector */
    __asm__("Move  #0x40,%PSR");    /* Processor responds to IRQ interrupts */
    *SCONT = 0x10;                  /* Enable receiver interrupts */

    /* Transfer the characters */
    while (1);                      /* Infinite loop */
}

/* Interrupt service routine */
void intserv()
{
    j = *RBUF & 0xF;                /* Extract the BCD value */
    *PAOUT = seg7[j];               /* Send the 7-segment code to Port A */
    __asm__("ReturnI");             /* Return from interrupt */
}
```

9.4. The arrangement explained in the solutions for Problems 9.1 and 9.2 can be used. The desired program may be obtained by modifying the program in Figure 9.14 as follows:

```
RBUF     EQU       $FFFFFFE0          Receive buffer.
SCONT    EQU       $FFFFFFE3          Control register for serial interface.
PAOUT    EQU       $FFFFFFF1          Port A output data.
PADIR    EQU       $FFFFFFF2          Port A direction register.

* Define the conversion table
         ORIGIN    $200
SEG7     DataByte  $7E, $30, $6C, $79, $33, $5B, $5F, $30 , $3F, $3B

* Initialization
         ORIGIN    $1000
         MoveByte  #$FF,PADIR         Configure Port A as output.
         Move      #INTSERV,$24       Set the interrupt vector.
         Move      #$40,PSR           Processor responds to IRQ interrupts.
         MoveByte  #$10,SCONT         Enable receiver interrupts.

* Transfer loop
LOOP     Branch    LOOP               Infinite wait loop.

* Interrupt service routine
INTSERV  MoveByte  RBUF,R0            Transfer a character to R0.
         And       #$F,R0             Extract the BCD value.
         MoveByte  SEG7(R0),PAOUT     Send the 7-segment code to Port A.
         ReturnI                      Return from interrupt.
```

9.5. The arrangement explained in the solution for Problem 9.1 can be used, having the 7-segment displays connected to Ports A and B. Upon detecting the character H, the first digit has to be saved and displayed only when the second digit arrives. The desired program may be obtained by modifying the program in Figure 9.11 as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SSTAT (volatile char *) 0xFFFFFFE2
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2
#define PBOUT (char *) 0xFFFFFFF4
#define PBDIR (char *) 0xFFFFFFF5


void main()
{
    /* Define the conversion table */
    char seg7[10] = {0x7E, 0x30, 0x6C, 0x79, 0x33, 0x5B, 0x5F, 0x30, 0x3F, 0x3B};
    char j, temp;

    /* Initialize the parallel ports */
    *PADIR = 0xFF;                      /* Configure Port A as output */
    *PBDIR = 0xFF;                      /* Configure Port B as output */

    /* Transfer the characters */
    while (1) {                         /* Infinite loop */
        while ((*SSTAT & 0x1) == 0);    /* Wait for a new character */
        if (*RBUF == 'H') {
          while ((*SSTAT & 0x1) == 0);  /* Wait for the first digit */
          j = *RBUF & 0xF;              /* Extract the BCD value */
          temp = seg7[j];               /* Prepare 7-segment code for Port A */
          while ((*SSTAT & 0x1) == 0);  /* Wait for the second digit */
          j = *RBUF & 0xF;              /* Extract the BCD value */
          *PBOUT = seg7[j];             /* Send the 7-segment code to Port B */
          *PAOUT = temp;                /* Send the 7-segment code to Port A */
        }
    }
}
```

9.6. The arrangement explained in the solutions for Problems 9.1 and 9.2 can be used, having the 7-segment displays connected to Ports A and B. Upon detecting the character H, the first digit has to be saved and displayed only when the second digit arrives. The desired program may be obtained by modifying the program in Figure 9.10 as follows:

```
RBUF    EQU         $FFFFFFE0       Receive buffer.
SSTAT   EQU         $FFFFFFE2       Status register for serial interface.
PAOUT   EQU         $FFFFFFF1       Port A output data.
PADIR   EQU         $FFFFFFF2       Port A direction register.
PBOUT   EQU         $FFFFFFF4       Port B output data.
PBDIR   EQU         $FFFFFFF5       Port B direction register.

* Define the conversion table
        ORIGIN      $200
SEG7    DataByte    $7E, $30, $6C, $79, $33, $5B, $5F, $30, $3F, $3B

* Initialization
        ORIGIN      $1000
        MoveByte    #$FF,PADIR      Configure Port A as output.
        MoveByte    #$FF,PBDIR      Configure Port B as output.

* Transfer the characters
LOOP    Testbit     #0,SSTAT        Check if new character is ready.
        Branch=0    LOOP
        MoveByte    RBUF,R0         Read the character.
        Compare     #$48,R0         Check if H.
        Branch≠0    LOOP
LOOP2   Testbit     #0,SSTAT        Check if first digit is ready.
        Branch=0    LOOP2
        MoveByte    RBUF,R0         Read the first digit.
        And         #$F,R0          Extract the BCD value.
LOOP3   Testbit     #0,SSTAT        Check if second digit is ready.
        Branch=0    LOOP3
        MoveByte    RBUF,R1         Read the second digit.
        And         #$F,R1          Extract the BCD value.
        MoveByte    SEG7(R1),PBOUT  Send the 7-segment code to Port B.
        MoveByte    SEG7(R0),PAOUT  Send the 7-segment code to Port A.
        Branch      LOOP
```

9.7. The arrangement explained in the solution for Problem 9.1 can be used, having the 7-segment displays connected to Ports A and B. Upon detecting the character H, the first digit has to be stored and displayed only when the second digit arrives. Interrupts are used to detect the arrival of both H and the subsequent pair of digits. Therefore, the interrupt service routine has to keep track of the received characters. Variable k is set to 2 when an H is detected, and it is decremented as the subsequent digits arrive. The desired program may be obtained by modifying the program in Figure 9.16 as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SCONT (char *) 0xFFFFFFE3
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2
#define PBOUT (char *) 0xFFFFFFF4
#define PBDIR (char *) 0xFFFFFFF5
#define int_addr (int *) (0x24)


void intserv();

void main()
{
    /* Define the conversion table */
    char seg7[10] = {0x7E, 0x30, 0x6C, 0x79, 0x33, 0x5B, 0x5F, 0x30, 0x3F, 0x3B};
    char j;
    char digits[2];                    /* Buffer for received BCD digits */
    int k = 0;                         /* Set up to detect the first H */
    /* Initialize the parallel ports */
    *PADIR = 0xFF;                     /* Configure Port A as output */
    *PBDIR = 0xFF;                     /* Configure Port B as output */

    /* Initialize the interrupt mechanism */
    int_addr = &intserv;               /* Set interrupt vector */
    __asm__("Move  #0x40,%PSR");       /* Processor responds to IRQ interrupts */
    *SCONT = 0x10;                     /* Enable receiver interrupts */

    /* Transfer the characters */
    while (1);                         /* Infinite loop */
}
```

```
/* Interrupt service routine */
void intserv()
{
        *SCONT = 0;             /* Disable interrupts */
        if (k > 0) {
          j = *RBUF & 0xF;      /* Extract the BCD value */
          k = k − 1;
          digits[k] = seg7[j];  /* Save 7-segment code for new digit */
          if (k == 0) {
            *PAOUT = digits[1]; /* Send first digit to Port A */
            *PBOUT = digits[0]; /* Send second digit to Port B */
           }
        else if (*RBUF == 'H') k = 2;
        }
        *SCONT = 0x10;          /* Enable receiver interrupts */
        _asm_("ReturnI");       /* Return from interrupt */
}
```

9.8. The arrangement explained in the solutions for Problems 9.1 and 9.2 can be used, having the 7-segment displays connected to Ports A and B. Upon detecting the character H, the first digit has to be stored and displayed only when the second digit arrives. Interrupts are used to detect the arrival of both H and the subsequent pair of digits. Therefore, the interrupt service routine has to keep track of the received characters. Variable K is set to 2 when an H is detected, and it is decremented as the subsequent digits arrive. The desired program may be obtained by modifying the program in Figure 9.14 as follows:

```
RBUF     EQU       $FFFFFFE0      Receive buffer.
SCONT    EQU       $FFFFFFE3      Control reg for serial interface.
PAOUT    EQU       $FFFFFFF1      Port A output data.
PADIR    EQU       $FFFFFFF2      Port A direction register.
PBOUT    EQU       $FFFFFFF4      Port B output data.
PBDIR    EQU       $FFFFFFF5      Port B direction register.

* Define the conversion table and buffer for first digit
         ORIGIN    $200
SEG7     DataByte  $7E, $30, $6C, $79, $33, $5B, $5F, $30, $3F, $3B
DIG      ReserveByte 1            Buffer for first digit.
K        Data      0              Set up to detect first H.
* Initialization
         ORIGIN    $1000
         MoveByte  #$FF,PADIR     Configure Port A as output.
         MoveByte  #$FF,PBDIR     Configure Port B as output.
         Move      #INTSERV,$24   Set the interrupt vector.
         Move      #$40,PSR       Processor responds to IRQ.
         MoveByte  #$10,SCONT     Enable receiver interrupts.

* Transfer loop
LOOP     Branch    LOOP           Infinite wait loop.

* Interrupt service routine
INTSERV  MoveByte  #0, SCONT      Disable interrupts.
         MoveByte  RBUF,R0        Read the character.
         Move      K,R1           See if a new digit
         Branch>0  NEWDIG           is expected.
         Compare   #$48,R0        Check if H.
         Branch≠0  DONE
         Move      #2,K           Detected an H.
         Branch    DONE
NEWDIG   And       #$F,R0         Extract the BCD value.
         Subtract  #1,R1          Decrement K.
         Move      R1,K
         Branch=0  DISP           Second digit received.
         MoveByte  SEG7(R0),DIG   Save the first digit.
         Branch    DONE
DISP     MoveByte  DIG,PAOUT      Send 7-segment code to Port A.
         MoveByte  SEG7(R0),PBOUT Send 7-segment code to Port B.
DONE     MoveByte  #$10,SCONT     Enable receiver interrupts.
         ReturnI                  Return from interrupt.
```

9.9. Connect the parallel ports A and B to the four BCD to 7-segment decoders. Choose that PA_{7-4}, PA_{3-0}, PB_{7-4} and PB_{3-0} display the first, second, third and fourth received digits, respectively. Assume that all four digits arrive immediately after the character H has been received. The task can be achieved by modifying the program in Figure 9.11 as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SSTAT (volatile char *) 0xFFFFFFE2
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2
#define PBOUT (char *) 0xFFFFFFF4
#define PBDIR (char *) 0xFFFFFFF5

void main()
{
    char temp;
    char digits[4];                          /* Buffer for received digits */
    int i;
    /* Initialize the parallel ports */
    *PADIR = 0xFF;                           /* Configure Port A as output */
    *PBDIR = 0xFF;                           /* Configure Port B as output */

    /* Transfer the characters */
    while (1) {                              /* Infinite loop */
        while ((*SSTAT & 0x1) == 0);         /* Wait for a new character */
        if (*RBUF == 'H') {
          for (i = 3; i >= 0; i--) {
            while ((*SSTAT & 0x1) == 0);     /* Wait for the next digit */
            digits[i] = *RBUF;               /* Save the new digit (ASCII) */
          }
          temp = digits[3] << 4;             /* Shift left first digit by 4 bits, */
          *PAOUT = temp | (digits[2] & 0xF); /*   append second and send to A */
          temp = digits[1] << 4;             /* Shift left third digit by 4 bits, */
          *PBOUT = temp | (digits[0] & 0xF); /*   append fourth and send to B */
        }
    }
}
```

9.10. Connect the parallel ports A and B to the four BCD to 7-segment decoders. Choose that PA_{7-4}, PA_{3-0}, PB_{7-4} and PB_{3-0} display the first, second, third and fourth received digits, respectively. Assume that all four digits arrive immediately after the character H has been received. Then, the desired program may be obtained by modifying the program in Figure 9.10 as follows:

```
RBUF    EQU       $FFFFFFE0    Receive buffer.
SSTAT   EQU       $FFFFFFE2    Status register for serial interface.
PAOUT   EQU       $FFFFFFF1    Port A output data.
PADIR   EQU       $FFFFFFF2    Port A direction register.
PBOUT   EQU       $FFFFFFF4    Port B output data.
PBDIR   EQU       $FFFFFFF5    Port B direction register.

* Initialization
        ORIGIN    $1000
        MoveByte  #$FF,PADIR   Configure Port A as output.
        MoveByte  #$FF,PBDIR   Configure Port B as output.

* Transfer the characters
LOOP    Testbit   #0,SSTAT     Check if new character is ready.
        Branch=0  LOOP
        MoveByte  RBUF,R0      Read the character.
        Compare   #$48,R0      Check if H.
        Branch≠0  LOOP
LOOP2   Testbit   #0,SSTAT     Check if first digit is ready.
        Branch=0  LOOP2
        MoveByte  RBUF,R0      Read the first digit.
        LShiftL   #4,R0        Shift left 4 bit positions.
LOOP3   Testbit   #0,SSTAT     Check if second digit is ready.
        Branch=0  LOOP3
        MoveByte  RBUF,R1      Read the second digit.
        And       #$F,R1       Extract the BCD value.
        Or        R1,R0        Concatenate digits for Port A.
LOOP4   Testbit   #0,SSTAT     Check if third digit is ready.
        Branch=0  LOOP4
        MoveByte  RBUF,R1      Read the third digit.
        LShiftL   #4,R1        Shift left 4 bit positions.
LOOP5   Testbit   #0,SSTAT     Check if fourth digit is ready.
        Branch=0  LOOP5
        MoveByte  RBUF,R2      Read the fourth digit.
        And       #$F,R2       Extract the BCD value.
        Or        R2,R1        Concatenate digits for Port B.
        MoveByte  R0,PAOUT     Send digits to Port A.
        MoveByte  R1,PBOUT     Send digits to Port B.
        Branch    LOOP
```

9.11. Connect the parallel ports A and B to the four BCD to 7-segment decoders. Choose that PA_{7-4}, PA_{3-0}, PB_{7-4} and PB_{3-0} display the first, second, third and fourth received digits, respectively. Upon detecting the character H, the subsequent four digits have to be saved and displayed only when the fourth digit arrives. Interrupts are used to detect the arrival of both H and the four digits. Therefore, the interrupt service routine has to keep track of the received characters. Variable k is set to 4 when an H is detected, and it is decremented as the subsequent digits arrive. The desired program may be obtained by modifying the program in Figure 9.16 as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SCONT (char *) 0xFFFFFFE3
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2
#define PBOUT (char *) 0xFFFFFFF4
#define PBDIR (char *) 0xFFFFFFF5
#define int_addr (int *) (0x24)

void intserv();

void main()
{
    /* Define the conversion table */
    char seg7[10] = {0x7E, 0x30, 0x6C, 0x79, 0x33, 0x5B, 0x5F, 0x30, 0x3F, 0x3B};
    char temp;
    char digits[4];                 /* Buffer for received BCD digits */
    int k = 0;                      /* Set up to detect the first H */
    /* Initialize the parallel ports */
    *PADIR = 0xFF;                  /* Configure Port A as output */
    *PBDIR = 0xFF;                  /* Configure Port B as output */

    /* Initialize the interrupt mechanism */
    int_addr = &intserv;            /* Set interrupt vector */
    __asm__("Move  #0x40,%PSR");    /* Processor responds to IRQ interrupts */
    *SCONT = 0x10;                  /* Enable receiver interrupts */

    /* Transfer the characters */
    while (1);                      /* Infinite loop */
}
```

```
/* Interrupt service routine */
void intserv()
{
        *SCONT = 0;                                 /* Disable interrupts */
        if (k > 0) {
          k = k − 1;
          digits[k] = *RBUF;                        /* Save the new digit (ASCII) */
          if (k == 0) {
            temp = digits[3] << 4;                  /* Shift left first digit by 4 bits, */
            *PAOUT = temp | (digits[2] & 0xF);      /*   append second and send to A */
            temp = digits[1] << 4;                  /* Shift left third digit by 4 bits */
            *PBOUT = temp | (digits[0] & 0xF);      /*   append fourth and send to B */
           }
        else if (*RBUF == 'H') k = 4;
        }
        *SCONT = 0x10;                              /* Enable receiver interrupts */
        _asm_("ReturnI");                           /* Return from interrupt */
}
```

9.12. Connect the parallel ports A and B to the four BCD to 7-segment decoders. Choose that PA_{7-4}, PA_{3-0}, PB_{7-4} and PB_{3-0} display the first, second, third and fourth received digits, respectively. Upon detecting the character H, the subsequent four digits have to be saved and displayed only when the fourth digit arrives. Interrupts are used to detect the arrival of both H and the four digits. Therefore, the interrupt service routine has to keep track of the received characters. Variable K is set to 4 when an H is detected, and it is decremented as the subsequent digits arrive. The desired program may be obtained by modifying the program in Figure 9.14 as follows:

```
RBUF        EQU         $FFFFFFE0   Receive buffer.
SCONT       EQU         $FFFFFFE3   Control reg for serial interface.
PAOUT       EQU         $FFFFFFF1   Port A output data.
PADIR       EQU         $FFFFFFF2   Port A direction register.
PBOUT       EQU         $FFFFFFF4   Port B output data.
PBDIR       EQU         $FFFFFFF5   Port B direction register.
            ORIGIN      $200
DIG         ReserveByte 4           Buffer for received digits.
K           Data        0           Set up to detect first H.
* Initialization
            ORIGIN      $1000
            MoveByte    #$FF,PADIR  Configure Port A as output.
            MoveByte    #$FF,PBDIR  Configure Port B as output.
            Move        #INTSERV,$24 Set the interrupt vector.
            Move        #$40,PSR    Processor responds to IRQ.
            MoveByte    #$10,SCONT  Enable receiver interrupts.
* Transfer loop
LOOP        Branch      LOOP        Infinite wait loop.

* Interrupt service routine
INTSERV     MoveByte    #0, SCONT   Disable interrupts.
            MoveByte    RBUF,R0     Read the character.
            Move        K,R1        See if a new digit
            Branch>0    NEWDIG       is expected.
            Compare     #$48,R0     Check if H.
            Branch≠0    DONE
            Move        #4,K        Detected an H.
            Branch      DONE
NEWDIG      And         #$F,R0      Extract the BCD value.
            Subtract    #1,R1       Decrement K.
            MoveByte    R0,DIG(R1)  Save the digit.
            Move        R1,K
            Branch>0    DONE        Expect more digits.
            Move        #DIG,R0     Pointer to buffer for digits.
DISP        MoveByte    (R0)+,R1    Get fourth digit.
            MoveByte    (R0)+,R2    Get third digit and
            LShiftL     #4,R2        shift it left.
            Or          R1,R2       Concatenate digits for Port B.
            MoveByte    R2,PBOUT    Send digits to Port B.
            MoveByte    (R0)+,R1    Get second digit.
            MoveByte    (R0)+,R2    Get first digit and
            LShiftL     #4,R2        shift it left.
            Or          R1,R2       Concatenate digits for Port A.
            MoveByte    R2,PAOUT    Send digits to Port A.
DONE        MoveByte    #$10,SCONT  Enable receiver interrupts.
            ReturnI                 Return from interrupt.
```

9.13. Use a table to convert a received ASCII digit into a 7-segment code as explained in the solution for Problem 9.1. Connect the bits S_a to S_g of all four registers to bits PA_{6-0} of Port A. Use bits PB_3 to PB_0 of Port B as Load signals for the registers displaying the first, second, third and fourth received digits, recpectively. Then, the required task can be achieved by modifying the program in Figure 9.11 as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SSTAT (volatile char *) 0xFFFFFFE2
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2
#define PBOUT (char *) 0xFFFFFFF4
#define PBDIR (char *) 0xFFFFFFF5

void main()
{
    /* Define the conversion table */
    char seg7[10] = {0x7E, 0x30, 0x6C, 0x79, 0x33, 0x5B, 0x5F, 0x30, 0x3F, 0x3B};
    char j;
    char digits[4];                          /* Buffer for received BCD digits */
    int k = 0;                               /* Set up to detect the first H */
    int i;
    /* Initialize the parallel ports */
    *PADIR = 0xFF;                           /* Configure Port A as output */
    *PBDIR = 0xFF;                           /* Configure Port B as output */

    /* Transfer the characters */
    while (1) {                              /* Infinite loop */
        while ((*SSTAT & 0x1) == 0);         /* Wait for a new character */
        if (*RBUF == 'H') {
          for (i = 3; i >= 0; i--) {
            while ((*SSTAT & 0x1) == 0);     /* Wait for the next digit */
            j = *RBUF & 0xF;                 /* Extract the BCD value */
            digits[i] = seg7[j];             /* Save 7-segment code for the digit */
          }
          for (i = 0; i <= 3; i++) {
            *PAOUT = digits[i];              /* Send a digit to Port A */
            *PBOUT = 1 << i;                 /* Load the digit into its register */
            *PBOUT = 0;                      /* Clear the Load signal */
          }
        }
    }
}
```

9.14. Use a table to convert a received ASCII digit into a 7-segment code as explained in the solution for Problem 9.1. Connect the bits S_a to S_g of all four registers to bits PA_{6-0} of Port A. Use bits PB_3 to PB_0 of Port B as Load signals for the registers displaying the first, second, third and fourth received digits, recpectively. Then, the required task can be achieved by modifying the program in Figure 9.10 as follows:

```
RBUF    EQU     $FFFFFFE0   Receive buffer.
SSTAT   EQU     $FFFFFFE2   Status register for serial interface.
PAOUT   EQU     $FFFFFFF1   Port A output data.
PADIR   EQU     $FFFFFFF2   Port A direction register.
PBOUT   EQU     $FFFFFFF4   Port B output data.
PBDIR   EQU     $FFFFFFF5   Port B direction register.

* Define the conversion table and buffer for received digi ts
        ORIGIN      $200
SEG7    DataByte    $7E, $30, $6C, $79, $33, $5B, $5F, $30, $3F, $3B
DIG     ReserveByte 4               Buffer for received digits.
* Initialization
        ORIGIN      $1000
        MoveByte    #$FF,PADIR      Configure Port A as output.
        MoveByte    #$FF,PBDIR      Configure Port B as output.

* Transfer the characters
LOOP    Testbit     #0,SSTAT        Check if new character is ready.
        Branch=0    LOOP
        MoveByte    RBUF,R0         Read the character.
        Compare     #$48,R0         Check if H.
        Branch≠0    LOOP
        Move        #3,R1           Set up a counter.
LOOP2   Testbit     #0,SSTAT        Check if next digit is available.
        Branch=0    LOOP2
        MoveByte    RBUF,R0         Read the digit.
        And         #4,R0           Extract the BCD value.
        MoveByte    SEG7(R0),DIG(R1) Save 7-seg code for the digit.
        Subtract    #1,R1           Check if more digits
        Branch>=0   LOOP2             are expected.
        Move        #DIG,R0         Pointer to buffer for digits.
        Move        #8,R1           Set up Load signal for d3.
DISP    MoveByte    (R0)+,PAOUT     Send 7-segment code to Port A.
        MoveByte    R1,PBOUT        Load the digit into its register.
        MoveByte    #0,PBOUT        Clear the Load signal.
        LShiftR     #1,R1           Set Load for the next digit.
        Branch>0    DISP            There are more digits to send.
        Branch      LOOP
```

9.15. Use a table to convert a received ASCII digit into a 7-segment code as explained in the solutions for Problems 9.1. and 9.2. Connect the bits S_a to S_g of all four registers to bits PA_{6-0} of Port A. Use bits PB_3 to PB_0 of Port B as Load signals for the registers displaying the first, second, third and fourth received digits, recpectively. Upon detecting the character H, the subsequent four digits have to be saved and displayed only when the fourth digit arrives. Interrupts are used to detect the arrival of both H and the four digits. Therefore, the interrupt service routine has to keep track of the received characters. Variable k is set to 4 when an H is detected, and it is decremented as the subsequent digits arrive. The desired program may be obtained by modifying the program in Figure 9.16 as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SCONT (char *) 0xFFFFFFE3
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2
#define PBOUT (char *) 0xFFFFFFF4
#define PBDIR (char *) 0xFFFFFFF5
#define int_addr (int *) (0x24)

void intserv();

void main()
{
    /* Define the conversion table */
    char seg7[10] = {0x7E, 0x30, 0x6C, 0x79, 0x33, 0x5B, 0x5F, 0x30, 0x3F, 0x3B};
    char j;
    char digits[4];                 /* Buffer for received BCD digits */
    int k = 0;                      /* Set up to detect the first H */
    int i;
    /* Initialize the parallel ports */
    *PADIR = 0xFF;                  /* Configure Port A as output */
    *PBDIR = 0xFF;                  /* Configure Port B as output */

    /* Initialize the interrupt mechanism */
    int_addr = &intserv;            /* Set interrupt vector */
    __asm__("Move  #0x40,%PSR");    /* Processor responds to IRQ interrupts */
    *SCONT = 0x10;                  /* Enable receiver interrupts */

    /* Transfer the characters */
    while (1);                      /* Infinite loop */
}
```

```
/* Interrupt service routine */
void intserv()
{
        *SCONT = 0;                 /* Disable interrupts */
        if (k > 0) {
          j = *RBUF & 0xF;          /* Extract the BCD value */
          k = k - 1;
          digits[k] = seg7[j];      /* Save 7-segment code for new digit */
          if (k == 0) {
            for (i = 0; i <= 3; i++) {
              *PAOUT = digits[i];   /* Send a digit to Port A */
              *PBOUT = 1 << i;      /* Load the digit into its register */
              *PBOUT = 0;           /* Clear the Load signal */
            }
          }
        else if (*RBUF == 'H') k = 4;
        }
        *SCONT = 0x10;              /* Enable receiver interrupts */
        _asm_("ReturnI");           /* Return from interrupt */
}
```

9.16. Use a table to convert a received ASCII digit into a 7-segment code as explained in the solutions for Problems 9.1. and 9.2. Connect the bits S_a to S_g of all four registers to bits PA_{6-0} of Port A. Use bits PB_3 to PB_0 of Port B as Load signals for the registers displaying the first, second, third and fourth received digits, recpectively. Upon detecting the character H, the subsequent four digits have to be saved and displayed only when the fourth digit arrives. Interrupts are used to detect the arrival of both H and the four digits. Therefore, the interrupt service routine has to keep track of the received characters. Variable K is set to 4 when an H is detected, and it is decremented as the subsequent digits arrive. The desired program may be obtained by modifying the program in Figure 9.14 as follows:

```
RBUF    EQU   $FFFFFFE0   Receive buffer.
SCONT   EQU   $FFFFFFE3   Control reg for serial interface.
PAOUT   EQU   $FFFFFFF1   Port A output data.
PADIR   EQU   $FFFFFFF2   Port A direction register.
PBOUT   EQU   $FFFFFFF4   Port B output data.
PBDIR   EQU   $FFFFFFF5   Port B direction register.
```

```
* Define the conversion table and buffer for received digits
            ORIGIN      $200
SEG7        DataByte    $7E, $30, $6C, $79, $33, $5B, $5F, $30, $3F, $3B
DIG         ReserveByte 4           Buffer for received digits.
K           Data        0           Set up to detect first H.
* Initialization
            ORIGIN      $1000
            MoveByte    #$FF,PADIR  Configure Port A as output.
            MoveByte    #$FF,PBDIR  Configure Port B as output.
            Move        #INTSERV,$24 Set the interrupt vector.
            Move        #$40,PSR    Processor responds to IRQ.
            MoveByte    #$10,SCONT  Enable receiver interrupts.
* Transfer loop
LOOP        Branch      LOOP        Infinite wait loop.

* Interrupt service routine
INTSERV     MoveByte    #0, SCONT   Disable interrupts.
            MoveByte    RBUF,R0     Read the character.
            Move        K,R1        See if a new digit
            Branch>0    NEWDIG       is expected.
            Compare     #$48,R0     Check if H.
            Branch≠0    DONE
            Move        #4,K        Detected an H.
            Branch      DONE
NEWDIG      And         #$F,R0      Extract the BCD value.
            Subtract    #1,R1       Decrement K.
            MoveByte    SEG7(R0),DIG(R1) Save 7-seg code for the digit.
            Move        R1,K
            Branch>0    DONE        Expect more digits.
            Move        #DIG,R0     Pointer to buffer for digits.
            Move        #8,R1       Set up Load signal for d3.
DISP        MoveByte    (R0)+,PAOUT Send 7-segment code to Port A.
            MoveByte    R1,PBOUT    Load the digit into its register.
            MoveByte    #0,PBOUT    Clear the Load signal.
            LShiftR     #1,R1       Set Load for the next digit.
            Branch>0    DISP        There are more digits to send.
DONE        MoveByte    #$10,SCONT  Enable receiver interrupts.
            ReturnI                 Return from interrupt.
```

9.17. Programs in Figures 9.17 and 9.18 would not work properly if the circular buffer was filled with 80 characters. After the head pointer wraps around, it would trail the tail pointer and would catch up with it if the buffer is full. At this point it would be impossible to use the simple comparison of the two pointers to determine whether the buffer is empty or full. The simplest modification is to increase the buffer size to 81 characters.

9.18. Using a counter variable, M, the program in Figure 9.17 can be modified as follows:

```
/* Define register addresses */
#define RBUF (volatile char *) 0xFFFFFFE0
#define SSTAT (volatile char *) 0xFFFFFFE2
#define PAOUT (char *) 0xFFFFFFF1
#define PADIR (char *) 0xFFFFFFF2
#define PSTAT (volatile char *) 0xFFFFFFF6
#define BSIZE 80


void main()
{
    unsigned char mbuffer[BSIZE];
    unsigned char fin, fout;
    unsigned char temp;
    int M = 0;

    /* Initialize Port A and circular buffer */
    *PADIR = 0xFF;                          /* Configure Port A as output */
    fin = 0;
    fout = 0;

    /* Transfer the characters */
    while (1) {                             /* Infinite loop */
        while ((*SSTAT & 0x1) == 0) {       /* Wait for a new character */
          if (M > 0) {                      /* If circular buffer is not empty */
            if (*PSTAT & 0x2) {             /*   and output device is ready */
              *PAOUT = mbuffer[fout];       /*   send a character to Port A */
              M = M - 1;                    /* Decrement the queue counter */
              if (fout < BSIZE-1)           /* Update the output index */
                fout++;
              else
                fout = 0;
            }
          }
        }
        mbuffer[fin] = *RBUF;               /* Read a character from receive buffer */
        M = M + 1;                          /* Increment the queue counter */
          if (fin < BSIZE-1)                /* Update the input index */
            fin++;
          else
            fin = 0;
    }
}
```

9.19. Using a counter variable, M, the program in Figure 9.18 can be modified as follows:

```
RBUF    EQU          $FFFFFFE0     Receive buffer.
SSTAT   EQU          $FFFFFFE2     Status reg for serial interface.
PAOUT   EQU          $FFFFFFF1     Port A output data.
PADIR   EQU          $FFFFFFF2     Port A direction register.
PSTAT   EQU          $FFFFFFF6     Status reg for parallel interface.
MBUF    ReserveByte  80            Define the circular buffer.

* Initialization
        ORIGIN       $1000
        MoveByte     #$FF,PADIR    Configure Port A as output.
        Move         #MBUF,R0      R0 points to the buffer.
        Move         #0,R1         Initialize head pointer.
        Move         #0,R2         Initialize tail pointer.
        Move         #0,R3         Initialize queue counter.

* Transfer the characters
LOOP    Testbit      #0,SSTAT      Check if new character is ready.
        Branch≠0     READ
        Compare      #0,R3         Check if queue is empty.
        Branch=0     LOOP          Queue is empty.
        Testbit      #1,PSTAT      Check if Port A is ready.
        Branch=0     LOOP
        MoveByte     (R0,R2),PAOUT Send a character to Port A.
        Subtract     #1,R3         Decrement the queue counter.
        Add          #1,R2         Increment the tail pointer.
        Compare      #80,R2        Is the pointer past queue limit?
        Branch<0     LOOP
        Move         #0,R2         Wrap around.
        Branch       LOOP
READ    MoveByte     RBUF,(R0,R1)  Place new character into queue.
        Add          #1,R3         Increment the queue counter.
        Add          #1,R1         Increment the head pointer.
        Compare      #80,R1        Is the pointer past queue limit?
        Branch<0     LOOP
        Move         #0,R1         Wrap around.
        Branch       LOOP
```

9.20. Connect the two 7-segment displays to Port A. Use the 3 bits of Port B to connect to the switches and LED as shown in Figure 9.19. It is necessary to modify the conversion and display portions of programs in Figures 9.20 and 9.21.

The end of the program in Figure 9.20 should be:

```
        /* Compute the total count */
        total_count = (0xFFFFFFFF − counter_value);

        /* Convert count to time */ ;
        actual_time = total_count / 1000000;          /* Time in hundredths of seconds */
        tenths = actual_time / 10;
        hundredths = actual_time − tenths * 10;

        *PAOUT = ((tenths << 4) | hundredths);    /* Display the elapsed time */
    }
}
```

The end of the program in Figure 9.20 should be:

```
* Convert the count to actual time in hundredths of seconds,
* and then to BCD. Put the BCD digits in R4.
        Move        #1000000,R1    Determine the count in
        Divide      R1,R2            hundredths of seconds.
        Move        #10,R1         Divide by 10 to find the digit that
        Divide      R1,R2            denotes 1/10th of a second.
        LShiftL     #4,R3          The BCD digits
        Or          R2,R3            are placed in R3.

        MoveByte    R3,PAOUT       Send digits to Port A.
        Branch      START          Ready for next test.
```