There are two methods for doing binary to BCD conversion. The first method makes use of the equation given by
\[ b_n b_{n-1} \ldots b_2 b_1 b_0 = 2^n b_n + 2^{n-1} b_{n-1} \ldots 2^2 b_2 + 2 b_1 + b_0 \]
or,
\[ b_n b_{n-1} \ldots b_2 b_1 b_0 = b_0 + 2(b_1 + 2(b_2 + \ldots + 2(b_{n-1} + 2b_n))) \]
If all of the operations in these equations are done using decimal arithmetic then the result will be in BCD.

The second method relies on the mod operator (%) and division by 10. If the digits of a binary number are \( b_n b_{n-1} \ldots b_2 b_1 b_0 \) and the digits of the equivalent BCD number are given by \( d_N d_{N-1} \ldots d_2 d_1 d_0 \) then \( d_0 = b_n b_{n-1} \ldots b_2 b_1 b_0 \mod 10 \). Likewise, \( d_1 = (b_n b_{n-1} \ldots b_2 b_1 b_0)/10 \mod 10 \) etc.

Both of these methods can be used to do the conversion and either can be written in assembly language or in C or in a mix of the two.

The following examples illustrate how this is done for a 12 bit binary number which converts to a 4 digit BCD number. All of the examples are done on an 8051 processor which has four I/O ports. The binary number is stored in P0:P1 and the BCD result is returned in P2:P3.
; This program converts a 12 bit binary number into BCD. All of the is done in
; assembly code. Algorithm is: binary number = b0 + 2b1 + 4 b2 + ... + 2^11*b11
; or
; binary number = b0 + 2*(b1 + 2*(b2 + 2*(b3 ... + 2*(b10 + 2*b11)))))))
; where all arithmetic is done in BCD.

MainSeg SEGMENT CODE
SubPgms SEGMENT CODE

CSEG at 0h
ljmp Start

RSEG MainSeg
Start: mov P0, #0FH  ;Test data in P0:P1
       mov P1, #0FFH
       mov r4, #0    ;r4:r5 holds the bcd number
       mov r5, #0
       mov a, P0
       anl a, #0FH   ;Limit P0:P1 to 12 bits
       mov r6, a     ;r6:r7 holds the number to convert
       mov r7, P1
       mov r0, #5
ShLp:   call ShiftLeft ;Shift R6:R7 left 5 times
       djnz r0, ShLp
       call AddCyDecimal
       mov r0, #11
MLp:    call Times2Decimal
       call ShiftLeft
       call AddCyDecimal
       djnz r0, MLp
       mov P3, r5
       mov P2, r4
Last: sjmp Last

RSEG SubPgms
ShiftLeft:clr c         ;Shifts r6:r7 one place left.
            mov a, r7     ; shifts zeros in from right.
            rlc a
            mov r7, a
            mov a, r6
            rlc a
            mov r6, a
            ret

AddCyDecimal:         ;Add carry to R4:R5 and adjusts decimal
            mov a, r5     ;First argument in R4:R5 (R5 is LSByte)
            addc a, #0    ;Add in carry
            da a          ;fix decimal
            mov r5, a
            mov a, r4     ;Repeat for upper byte
            addc a, #0
            da a
            mov r4, a     ;Answer returned in R4:R5
            ret

Times2Decimal:      ;Doubles R4:R5 and decimal adjusts
            mov a, r5    ;Argument in R4:R5 with R5 as LSByte
            add a, r5    ;Add R7 to itself to double it
            da a        ;fix decimal
            mov r5, a
            mov a, r4    ;Repeat for upper byte. Tuck in carry
            addc a, r4
            da a
            mov r4, a    ;Answer returned in R4:R5
            ret

END
/* BCDConversion.c */
This program converts a 12 bit binary number into BCD
All of the bit manipulation and BCD adjustment is done in C
Algorithm is
   binary number = b0 + 2b1 + 4 b2 + ... + 2^11*b11
   or
   binary number = b0 + 2*(b1 + 2*(b2 + 2*(b3 ... + 2*(b10 + 2*b11))))))),
where all arithmetic is done in BCD.
*/
#include<reg51.h>
unsigned int DecimalAdjust(unsigned int num);
void main()
{
    unsigned int num, bcd;
    unsigned char i;
    bcd = 0;
    P1 = 0xff; P0 = 0x0f;               // MSB byte in P0, LSB byte in P1
    num = (P1 + 256*P0) & 0x0FFF;      // Form a 12 bit number
    num = num << 5;                    // msb to far left
    bcd = bcd + (unsigned int) CY;      // msb to bcd
    for(i=0; i<11; i++)
    {
        bcd = bcd*2;                    // multiply by 2 eleven times
        bcd = DecimalAdjust(bcd);       // Always decimal adjust
        num = num << 1;                 // Add in next bit
        bcd = bcd + (unsigned int) CY;   // decimal adjust
    }
    P3 = bcd;
    P2 = bcd/256;
}

unsigned int DecimalAdjust(unsigned int bcd)
{
    unsigned char nibble;
    nibble = bcd & 0x0f;              // add 6 to any nibble > 1001
    if(nibble > 9)                    // carry propagates to next nibble
        bcd = bcd + 6;
    nibble = (bcd >> 4) & 0x0f;
    if(nibble > 9)
        bcd = bcd + 0x60;
    nibble = (bcd/256) & 0x0f;        // 8-bit shift
    if(nibble > 9)
        bcd = bcd + 0x600;
    nibble = ((bcd/256) >> 4) & 0x0f;
    if(nibble > 9)
        bcd = bcd + 0x6000;
    return bcd;
}
/ *BCDConversion2.c
 * This program converts a 12 bit binary number into BCD. All of the bit manipulation
 * and BCD adjustment is done in assembly code.
 * Algorithm is
 * binary number = b0 + 2b1 + 4 b2 + ... + 2^11*b11
 * or
 * binary number = b0 + 2*(b1 + 2*(b2 + 2*(b3 ... + 2*(b10 + 2*b11)))))))
 * where all arithmetic is done in BCD.
 */
#include<reg51.h>
extern unsigned int Times2Decimal(unsigned int bcd);
extern unsigned int AddCyDecimal(unsigned int bcd);

void main()
{
  unsigned int num, bcd;
  unsigned char i;
  bcd = 0;
  P1 = 0xff;P0 = 0x0f;           //MSBbyte in P0, LSByte in P1
  num = (P1 + 256*P0) & 0x0FFF;  //Form a 12 bit number
  num = num << 5;                //msb to far left
  bcd = bcd + (unsigned int)CY;  //msb to bcd
  for(i=0;i<11;i++)
  {
    bcd = Times2Decimal(bcd);
    num = num << 1;
    bcd = AddCyDecimal(bcd);
  }
  P3 = bcd;
  P2 = bcd/256;
}

/****************** EXTERNAL ASSEMBLY MODULE 1 *******************/
;AddCyDecimal.a51
;Adds carry to R6:R7 and decimal adjusts the result
public _AddCyDecimal
AddCy SEGMENT CODE
RSEG AddCy
_AddCyDecimal:
  mov a, r7        ;First argument in R6:R7 (R7 is LSByte)
  addc a, #0      ;Add in carry
  da a             ;fix decimal
  mov r7, a
  mov a, r6        ;Repeat for upper byte
  addc a, #0
  da a
  mov r6, a        ;Answer returned in R6:R7
  ret

END

/****************** EXTERNAL ASSEMBLY MODULE 2 *******************/
;Times2Decimal.a51
;Multiplies the argument in R6:R7 by 2 and decimal adjusts result
public _Times2Decimal
Times2 SEGMENT CODE
RSEG Times2
_Times2Decimal:
  mov a, r7        ;Argument in R6:R7 with R7 as LSByte
  add a, r7        ;Add R7 to itself to double it
  da a             ;fix decimal
  mov r7, a
  mov a, r6        ;Repeat for upper byte. Tuck in carry
  addc a, r6
  da a
  mov r6, a        ;Answer returned in R6:R7
  ret

END
```c
#include<reg51.h>
//Convert to BCD by using mod function. Converts a 12 bit number // in P0:P1 into a 4 digit BCD number in P2:P3
void main(void)
{
    unsigned int num;
    unsigned char d0, d1, d2, d3;
    P1 = 0xff; // P0 = 0x0f;  // 0xFFF is max 12 bit number
    num = P1 + (P0 << 8);
    d0 = num % 10;
    num = num/10;
    d1 = num%10;
    num = num/10;
    d2 = num%10;
    num = num/10;
    d3 = num%10;
    P3 = d0 + (d1 << 4);
    P2 = d2 + (d3 << 4);
}
```
Summary of Results:

<table>
<thead>
<tr>
<th></th>
<th>Using Bits algorithm</th>
<th>Size in bytes</th>
<th>Lines of code</th>
<th>Total states executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Assembly</td>
<td></td>
<td>75</td>
<td>53</td>
<td>503</td>
</tr>
<tr>
<td>All C Code</td>
<td>Note 1</td>
<td>217</td>
<td>37</td>
<td>2130</td>
</tr>
<tr>
<td>Mixed C and Assembly</td>
<td>Note 1</td>
<td>131</td>
<td>47</td>
<td>995</td>
</tr>
</tbody>
</table>

Note 1: Includes 389 states and 15 bytes of startup code.

<table>
<thead>
<tr>
<th></th>
<th>Using mod 10 algorithm</th>
<th>Size in bytes</th>
<th>Lines of code</th>
<th>Total states executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>All C Code</td>
<td>Note 1</td>
<td>217</td>
<td>15</td>
<td>1050</td>
</tr>
</tbody>
</table>

Note 1: Includes 389 states and 15 bytes of startup code.