How to convert a 24 bits number to decimal

Question: *In some of your previous exams, it is indicated to display a result which is on 24 bits. What is the easiest way to print integers? I know how to print an 16bit integer into the display screen (by convert it to BCD or dividing by 10 and then converting chars to ASCII and printing char by char) but on 24 bits, it could be very hard, how should I do in these cases?*

First Answer

Let's call A our number, which is represented on 24 bits; A (on 24 bits) is at most about 4 * 10^6, i.e. 7 decimal digits. Why not considering to split A into two “decimal” halves, each one fitting 16 bits?

For example: $A = B*10^3 + C$

Certainly:

$B= int(A/10^3) <= 4000$ (it fits on 16 bits for sure)
C is the residual of the previous division, and therefore it is < 1000 (fitting 16 bits as well)

You store A on 32 bits (DX:AX) and then divide by CX with CX storing 1000; The quotient (in AX) will be B and the residual (in DX) will be C. Now you have two 16 bits numbers B and C, to be converted in decimal (4 digits B and 3 digits for C); this is trivial; you convert B first and continue appending the digits coming out from the conversion of C.

Second answer (general for any number of bits)

Search Google for the “Shift and Add-3 Algorithm”

http://people.ee.duke.edu/~dwyer/courses/ece52/Binary_to_BCD_Converter.pdf

Binary to BCD Converter

Shift and Add-3 Algorithm

1. Shift the binary number left one bit.
2. If 8 shifts have taken place, the BCD number is in the *Hundreds*, *Tens*, and *Units* column.
3. If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
4. Go to 1.

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1

Example 1: Convert hex `E` to BCD

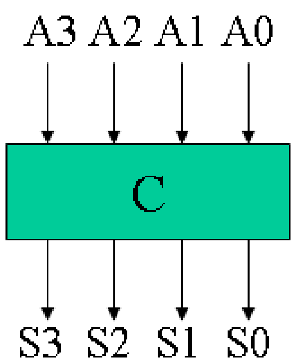
Operation	Tens	Units	Binary
HEX			**E**
Start			1 1 1 0
Shift 1		1	1 1 0
Shift 2		1 1	1 0
Shift 3		1 1 1	0
Add 3		1 0 1 0	0
Shift 4	1	0 1 0 0	
BCD	**1**	**4**	

Example 2: Convert hex `FF` to BCD

Operation	Hundreds	Tens	Units	Binary	
HEX				**F**	**F**
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	

Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

Truth table for Add-3 Module



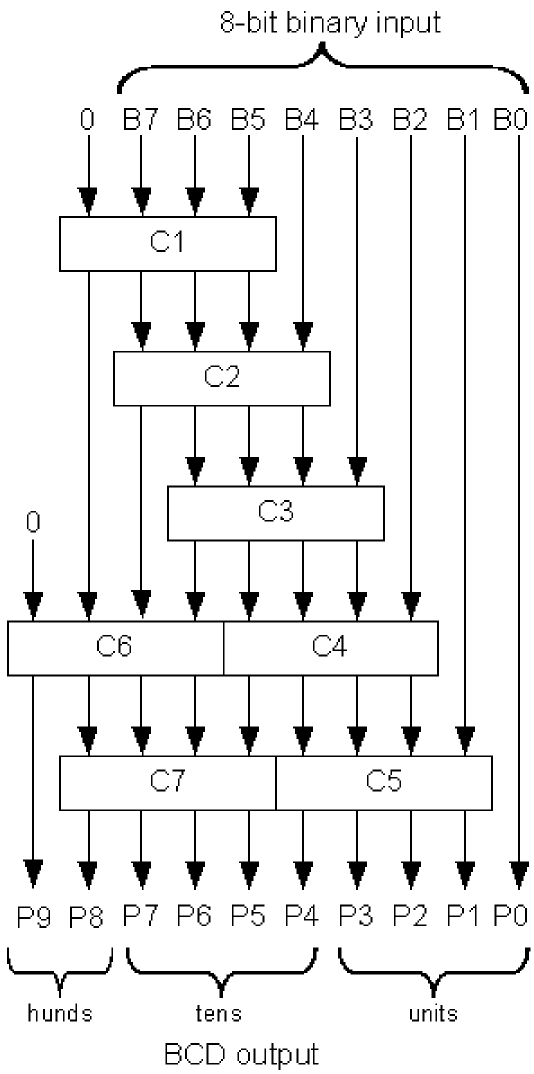
A3	A2	A1	A0	S3	S2	S1	S0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Here is a Verilog module for this truth table.

```
module add3(in,out);
input [3:0] in;
output [3:0] out;
reg [3:0] out;

always @ (in)
        case (in)
        4'b0000: out <= 4'b0000;
        4'b0001: out <= 4'b0001;
        4'b0010: out <= 4'b0010;
        4'b0011: out <= 4'b0011;
        4'b0100: out <= 4'b0100;
        4'b0101: out <= 4'b1000;
        4'b0110: out <= 4'b1001;
        4'b0111: out <= 4'b1010;
        4'b1000: out <= 4'b1011;
        4'b1001: out <= 4'b1100;
        default: out <= 4'b0000;
        endcase
endmodule
```

Binary-to-BCD Converter Module



Here is a structural Verilog module corresponding to the logic diagram.

```
module binary_to_BCD(A,ONES,TENS,HUNDREDS);
input [7:0] A;
output [3:0] ONES, TENS;
output [1:0] HUNDREDS;
wire [3:0] c1,c2,c3,c4,c5,c6,c7;
wire [3:0] d1,d2,d3,d4,d5,d6,d7;

assign d1 = {1'b0,A[7:5]};
assign d2 = {c1[2:0],A[4]};
assign d3 = {c2[2:0],A[3]};
assign d4 = {c3[2:0],A[2]};
assign d5 = {c4[2:0],A[1]};
assign d6 = {1'b0,c1[3],c2[3],c3[3]};
assign d7 = {c6[2:0],c4[3]};
add3 m1(d1,c1);
add3 m2(d2,c2);
add3 m3(d3,c3);
add3 m4(d4,c4);
add3 m5(d5,c5);
add3 m6(d6,c6);
add3 m7(d7,c7);
assign ONES = {c5[2:0],A[0]};
assign TENS = {c7[2:0],c5[3]};
assign HUNDREDS = {c6[3],c7[3]};
```

```
endmodule
```

General Binary-to-BCD Converter

The linked code is a general binary-to-BCD Verilog module, but I have not personally tested the code.

Reference: course materials from Prof. Richard E. Haskell

Maintained by John Loomis, last updated *4 Jan 2004*

Binary-to-BCD Converter

Double-Dabble Binary-to-BCD Conversion Algorithm

Basic Idea

- Y←X, X is a 4-bit binary number
  - Y is a 4-bit binary number (Binary to binary)
    ⇒ can be done by only shifting
    ex: 1011 ← 1011 (shift left 4 times)
  - Y is a BCD number (Binary to BCD)
    ∵X: 0000~1111,
    ∴Y: 00~15 (two BCD digits, at least 5 bits)
    ex: 01000 ← 1000
    ? ← 1011

Y				X			
				1	0	1	1
			1	0	1	1	
		1	0	1	1		
	1	0	1	1			
1	0	1	1				

```
if (U > 4)
   then U=U+3;
Shift left;
```

Shift left

U ← U*2+X[3]

U				X			
				1	0	1	1
			1	0	1	1	
		1	0	1	1		
	1	0	1	1			
1	0	1	1				

Out of range
U=U+6;

if (U > 4) then U will be
Out of range after "shift left"

Double-Dabble Binary-to-BCD Conversion Algorithm

Shift and Add-3 Algorithm (consider 8-bit binary)

1. Shift the binary number left one bit.
2. If 8 shifts have taken place, the BCD number is in the *Hundreds*, *Tens*, and *Units* column.
3. If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
4. Go to 1.

Example:

8 bits

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1

Steps to convert an 8-bit binary number to BCD

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

Example of converting hex E to BCD

Operation	Tens	Units	Binary
HEX			E
Start			1 1 1 0
Shift 1		1	1 1 0
Shift 2		1 1	1 0
Shift 3		1 1 1	0
Shift 4		1 1 1 0	
6		0 1 1 0	
Add 6	1	0 1 0 0	
BCD	1	4	

Steps to convert a 6-bit binary number to BCD

1. Clear all bits of z to zero
2. Shift B left 3 bits

   $z[8:3] = B[5:0]$;
3. Do 3 times

   if $Units > 4$

   then add 3 to $Units$

   (note: $Units = z[9:6]$)

   Shift z left 1 bit
4. $Tens = P[6:4] = z[12:10]$

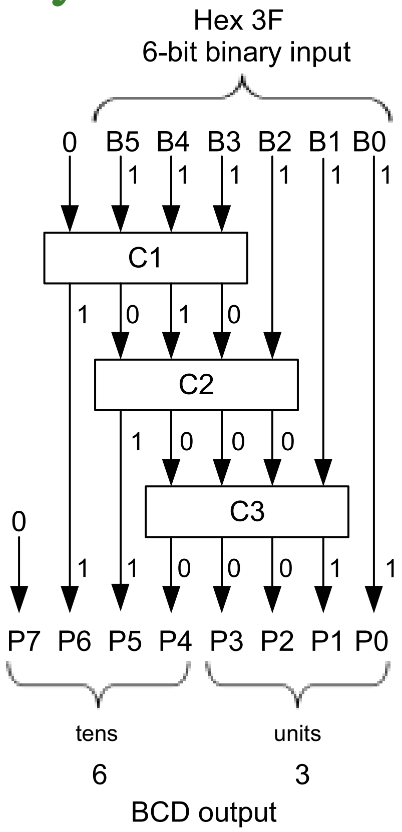
   $Units = P[3:0] = z[9:6]$

Operation	Tens	Units	Binary
B			5 4 3 2 1 0
HEX			3 F
Start			1 1 1 1 1 1
Shift 1		1	1 1 1 1 1
Shift 2		1 1	1 1 1 1
Shift 3		1 1 1	1 1 1
Add 3		1 0 1 0	1 1 1
Shift 4	1	0 1 0 1	1 1
Add 3	1	1 0 0 0	1 1
Shift 5	1 1	0 0 0 1	1
Shift 6	1 1 0	0 0 1 1	
BCD	6	3	
P	7 4	3 0	
z	13 10	9 6	5 0

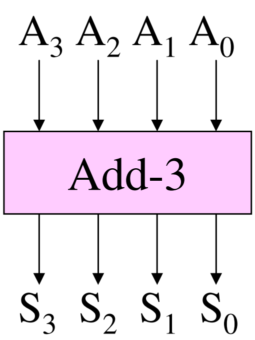
How to implement?

Steps to convert a 6-bit binary number to BCD (Cont'd)

Operation	Tens	Units	Binary
B			5 4 3 2 1 0
HEX			3 F
Start			1 1 1 1 1 1
Shift 1		1	1 1 1 1 1
Shift 2		1 1	1 1 1 1
Shift 3		1 1 1	1 1 1
Add 3		1 0 1 0	1 1 1
Shift 4	1	0 1 0 1	1 1
Add 3	1	1 0 0 0	1 1
Shift 5	1 1	0 0 0 1	1
Shift 6	1 1 0	0 0 1 1	
BCD	6	3	
P	7 4	3 0	
z	13 10	9 6	5 0



Truth table for Add-3 Module



A_3	A_2	A_1	A_0	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

K-Map for S_3

A_3	A_2	A_1	A_0	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

A_3A_2 \ A_1A_0	00	01	11	10
00				
01		1	1	1
11	x	x	x	x
10	1	1	x	x

$S_3 = A_3+A_2A_0+A_2A_1$

$S_2 = A_3A_0+A_2A_1'A_0'$

$S_1 = A_3A_0'+A_2'A_1+A_1A_0$

$S_0 = A_3A_0'+A_3'A_2'A_0+A_2A_1A_0'$

exercise

- Design a Verilog module to convert an 8-bit binary number to the BCD form.

```
module Binary_to_BCD_8(P,B);
output [9:0] P; //BCD form of B
input [7:0] B;
. . .
endmodule
```