Assignment # 2
(CLO2 -> PLO2)
Computer System Architecture

Submission Deadline: 23rd Nov 2015

Note: Students should score 40% in OBE specific questions to ensure their accumulated scores towards respective PLOs are above 40%

1. This problem covers 4-bit binary multiplication. Fill in the table for the Product, Multiplier and Multiplicand for each step. You need to provide the DESCRIPTION of the step being performed (shift left, shift right, add, no add). The value of M (Multiplicand) is 1011, Q (Multiplier) is initially 1010.

Product	Multiplicand	Multiplier	Description	Step
0000 0000	0000 1011	1010	Initial Values	Step 0
				Step 1
				Step 2
				Step 3
				Step 4
				Step 5
				Step 6
				Step 7
				Step 8
				Step 9
				Step 10
				Step 11
				Step 12
				Step 13
				Step 14
				Step 15

Product	Multiplicand	Multiplier	Description	Step
0000 0000	0000 1011	1010	Initial Values	Step 0
0000 0000	0000 1011	1010	0 => No add	Step 1
0000 0000	0001 0110	1010	Shift left M	Step 2
0000 0000	0001 0110	0101	Shit right Q	Step 3
0001 0110	0001 0110	0101	1 => Add M to Product	Step 4
0001 0110	0010 1100	0101	Shift left M	Step 5
0001 0110	0010 1100	0010	Shift right Q	Step 6
0001 0110	0010 1100	0010	0 => No add	Step 7
0001 0110	0101 1000	0010	Shift left M	Step 8
0001 0110	0101 1000	0001	Shift right Q	Step 9
0110 1110	0101 1000	0001	1 => Add M to Product	Step 10
0110 1110	1011 0000	0001	Shift left M	Step 11
0110 1110	1011 0000	0000	Shift right Q	Step 12
0110 1110	1011 0000	0000	0 => No add	Step 13
0110 1110	0110 0000	0000	Shift left M	Step 14
0110 1110	0110 0000	0000	Shift Right Q	Step 15

2. Consider 2's complement 4-bit signed integer addition and subtraction. Since the operands can be negative or positive and the operator can be subtraction or addition, there are 8 possible combinations of inputs. For example, a positive number could be added to a negative number, or a negative number could be subtracted from a negative number, etc. For each of them, describe how the overflow can be computed from the sign of the input operands and the carry out and sign of the output. Fill in the table below:

Sign (Input 1)	Sign (Input 2)	Operation	Sign (Output)	Overflow (Y/N)
+	+	+	+	
+	+	+	-	
+	+	-	+	
+	+	-	-	
+	-	+	+	
+	-	+	-	
+	-	-	+	
+	-	-	-	
-	+	+	+	
-	+	+	-	
-	+	-	+	
-	+	-	-	
-	-	+	+	
-	-	+	-	
-	-	-	+	
-	-	-	-	

Sign (Input 1)	Sign (Input 2)	Operation	Sign (Output)	Overflow (Y/N)
+	+	+	+	N
+	+	+	-	Y
+	+	-	+	N
+	+	-	-	N
+	-	+	+	N
+	-	+	-	N
+	-	-	+	N
+	-	-	-	Y
-	+	+	+	N
-	+	+	-	N
-	+	-	+	N
-	+	-	-	N
-	-	+	+	Y
-	-	+	-	N
-	-	-	+	N
-	-	-	-	N

3. This problem covers 4-bit binary unsigned division (similar to Fig. 3.11 in the text). Fill in the table for the Quotient, Divisor and Dividend for each step. You need to provide the DESCRIPTION of the step being performed (shift left, shift right, sub). The value of Divisor is 4 (0100, with additional 0000 bits shown for right shift), Dividend is 6 (initially loaded into the Remainder).

Quotient	Divisor	Remainder	Description	Step
0000	0100 0000	0000 0110	Initial Values	Step 0
				Step 1
				Step 2
				Step 3
				Step 4
				Step 5
				Step 6
				Step 7
				Step 8
				Step 9
				Step 10
				Step 11
				Step 12
				Step 13
				Step 14
				Step 15

Quotient	Divisor	Remainder	Description	Step
0000	0100 0000	0000 0110	Initial Values	Step 0
0000	0100 0000	1100 0110	Rem = Rem – Div	Step 1
0000	0100 0000	0000 0110	Rem < 0 => +Div, sll Q, Q0 = 0	Step 2
0000	0010 0000	0000 0110	Shift Div to right	Step 3
0000	0010 0000	1110 0110	Rem = Rem – Div	Step 4
0000	0010 0000	0000 0110	Rem < 0 => +Div, sll Q, Q0 = 0	Step 5
0000	0001 0000	0000 0110	Shift Div to right	Step 6
0000	0001 0000	1111 0110	Rem = Rem – Div	Step 7
0000	0001 0000	0000 0110	Rem < 0 => +Div, sll Q, Q0 = 0	Step 8
0000	0000 1000	0000 0110	Shift Div to right	Step 9
0000	0000 1000	1111 1110	Rem = Rem – Div	Step 10
0000	0000 1000	0000 0110	Rem < 0 => +Div, sll Q, Q0 = 0	Step 11
0000	0000 0100	0000 0110	Shift Div to right	Step 12
0000	0000 0100	0000 0010	Rem = Rem – Div	Step 13
0001	0000 0100	0000 0010	Rem > 0 => sll Q, Q0 = 1	Step 14
0001	0000 0010	0000 0010	Shift Div Right	Step 15

4. Assuming single precision IEEE 754 format, what decimal number is represent by this word:
   1 01111101 00100000000000000000000
   (Hint: remember to use the biased form of the exponent.)

   $(+1)^* (2^{(125-127)})^*(1.001)_2$
   $(+1)^*(0.25)^*(0.125)$
   0.03125

5. The floating-point format to be used in this problem is an 8-bit IEEE 754 normalized format with 1 sign bit, 4 exponent bits, and 3 mantissa bits. It is identical to the 32-bit and 64-bit formats in terms of the meaning of fields and special encodings. The exponent field employs an excess-7 coding. The bit fields in a number are (sign, exponent, mantissa). Assume that we use unbiased rounding to the nearest even specified in the IEEE floating point standard. Encode the following numbers the 8-bit IEEE format:
   (1) 0.0011011_{binary}
   (2) $16.0_{decimal}$
   (3) Decode the following 8-bit IEEE number into their decimal value: 1 1010 101
   (4) Decide which number in the following pairs are greater in value (the numbers are in 8-bit IEEE 754 format):
   0 0100 100 and 0 0100 111

(1) 0.0011011_{binary}

This number = $1.1011_2 * 2^{-3}$
= $(+1) * (1.1011)_2 * 2^{(4-7)}$
= 0 0100 110 in the 8-bit IEEE 754
(after applying unbiased rounding to the nearest even)

(2) $16.0_{decimal}$

This number = $(1000.0)_2$
= $(1.000)_2 * 2^{(10-7)}$
= 0 1010 000 in the 8-bit IEEE 754

(3)

The decimal value is $(+1)*(2)^{(10-7)}*1.625 = 13.0$

(d) Decide which number in the following pairs are greater in value (the numbers are in 8-bit IEEE 754 format):

(1) 0 0100 100 and 0 0100 111

The first number = $2^{(8-7)}*(1.1)2 = 3.0$
The second number = $2^{(8-7)}*(1.111)2 = 3.75$

The second number is greater in value

(2) 0 1100 100 and 1 1100 101

6. Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 single and double precision formats.

$63.25 \times 10^0 = 111111.01 \times 2^0$
normalize, move binary point 5 to the left
1.1111101×2^5
sign = positive, exp = 127 + 5 = 132
Final bit pattern: 0 1000 0100 1111 1010 0000 0000 0000 000
\ = 0100 0010 0111 1101 0000 0000 0000 0000 = 0x427D0000

$63.25 \times 10^0 = 111111.01 \times 2^0$
normalize, move binary point 5 to the left
1.1111101×2^5
sign = positive, exp = 1023 + 5 = 1028
Final bit pattern:
0 100 0000 0100 1111 1010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
= 0x404FA00000000000