Introduction to Computer Science-101
Homework 2_solution

1. Show the result of the following operations. (10%)
   a. NOT $(99)_{16}$ $(66)_{16}$
   b. NOT $(FF)_{16}$ $(00)_{16}$
   c. NOT $(00)_{16}$ $(FF)_{16}$
   d. NOT $(01)_{16}$ $(FE)_{16}$

2. Show the result of the following operations. (10%)
   a. $(99)_{16}$ AND $(99)_{16}$ $(99)_{16}$
   b. $(99)_{16}$ AND $(00)_{16}$ $(00)_{16}$
   c. $(99)_{16}$ OR $(FF)_{16}$ $(FF)_{16}$
   d. $(FF)_{16}$ OR $(FF)_{16}$ $(FF)_{16}$

3. Using an 8-bit allocation, first convert each of the following numbers to sign-and-magnitude representation, do the operation, and then convert the result to decimal. (10%)
   a. 19 + 23 42

$19 + 23 \rightarrow A = 19 = (00010011)_2$ and $B = 23 = (00010111)_2$.
Operation is addition; sign of B is not changed. $S = A_S$ XOR $B_S = 0$, $R_M = A_M + B_M$ and $R_S = A_S$

No overflow

			1		1	1	1		Carry
A_S	0		0	0	1	0	0	1	1	A_M
B_S	0	+	0	0	1	0	1	1	1	B_M
R_S	0		0	1	0	1	0	1	0	R_M

The result is $(00101010)_2 = 42$ as expected.

   b. 19 – 23 -4

$19 - 23 \rightarrow A = 19 = (00010011)_2$ and $B = 23 = (00010111)_2$. Operation is subtraction, sign of B is changed. $B_S = \overline{B}_S$. $S = A_S$ XOR $B_S = 1$, $R_M = A_M + (\overline{B}_M + 1)$. Since there is no overflow $R_M = (\overline{R}_M + 1)$ and $R_S = B_S$

No overflow

						1	1		Carry
A_S	0		0	0	1	0	0	1	1	A_M
B_S	1	+	1	1	0	1	0	0	1	$(\overline{B}_M + 1)$
			1	1	1	1	1	0	0	R_M
R_S	1		0	0	0	0	1	0	0	$R_M = (\overline{R}_M + 1)$

The result is $(10000100)_2 = -4$ as expected.

c. -19 + 23 4

$-19 + 23 \rightarrow A = -19 = (10010011)_2$ and $B = 23 = (00010111)_2$. Operation is addition, sign of B is not changed. $S = A_S$ XOR $B_S = 1$, $R_M = A_M + (\overline{B}_M + 1)$. Since there is no overflow $R_M = (\overline{R}_M + 1)$ and $R_S = B_S$

	No overflow					1	1			Carry
A_S	1		0	0	1	0	0	1	1	A_M
B_S	0	+	1	1	0	1	0	0	1	$(\overline{B}_M + 1)$
			1	1	1	1	1	0	0	R_M
R_S	0		0	0	0	0	1	0	0	$R_M = (\overline{R}_M + 1)$

The result is $(00000100)_2 = 4$ as expected.

d. -19 – 23 -42

$-19 - 23 \rightarrow A = -19 = (10010011)_2$ and $B = 23 = (00010111)_2$. Operation is subtraction, sign of B is changed. $S = A_S$ XOR $B_S = 0$, $R_M = A_M + B_M$ and $R_S = A_S$

	No overflow			1		1	1	1		Carry
A_S	1		0	0	1	0	0	1	1	A_M
B_S	1	+	0	0	1	0	1	1	1	B_M
R_S	1		0	1	0	1	0	1	0	R_M

The result is $(10101010)_2 = -42$ as expected.

4. Show the result of the following floating-point operations using IEEE_127—see Chapter 3. (10%)

   a. 34.75 + 23.125

$34.75 + 23.125 = (100010.11)_2 + (10111.001)_2 = 2^5 \times (1.0001011)_2 + 2^4 \times (1.0111001)_2$. These two numbers are stored in floating-point format as shown, but we need to remember that each number has a hidden 1 (which is not stored, but assumed). $E_1 = 127 + 5 = 132 = (10000100)_2$ and $E_2 = 127 + 4 = 131 = (10000011)_2$. The first few steps in UML diagram is not needed. We move to denormalization. We denormalize the numbers by adding the hidden 1's to the mantissa and incrementing the exponent.

	S	E	M
A	0	10000100	00010110000000000000000
B	0	10000011	01110010000000000000000

Now both denormalized mantissas are 24 bits and include the hidden 1's. They should store in a location to hold all 24 bits. Each exponent is incremented.

	S	E	Denormalized M
A	0	10000101	100010110000000000000000
B	0	10000100	101110010000000000000000

We align the mantissas. We increment the second exponent by 1 and shift its mantissa to the right once.

	S	E	Denormalized M
A	0	10000101	100010110000000000000000
B	0	10000101	010111001000000000000000

Now we do sign-and-magnitude addition treating the sign and the mantissa of each number as one integer stored in sign-and-magnitude representation.

	S	E	Denormalized M
R	0	10000101	11100111100000000000000

There is no overflow in mantissa, so we normalized.

	S	E	M
R	0	10000100	11001111000000000000000

The mantissa is only 23 bits because there is no overflow, no rounding is needed.

$$E = (10000100)_2 = 132, M = 11001111$$

In other words, the result is

$$(1.11001111)_2 \times 2^{132-127} = (111001.111)_2 = 57.875$$

b. -12.625 + 451.00

$-12.625 + 451 = -(1100.101)_2 + (111000011)_2 = -2^3 \times (1.100101)_2 + 2^8 \times (1.11000011)_2$. These two numbers are stored in floating-point format as shown, but we need to remember that each number has a hidden 1 (which is not stored, but assumed). $E_1 = 127 + 3 = 130 = (10000010)_2$ and $E_2 = 127 + 8 = 135 = (10000111)_2$

	S	E	M
A	1	10000010	10010100000000000000000
B	0	10000111	11000011000000000000000

The first few steps in UML diagram is not needed. We move to denormalization. We denormalize the numbers by adding the hidden 1's to the mantissa and incrementing the exponent. Now both denormalized mantissas are 24 bits and include the hidden 1's. They should store in a location to hold all 24 bits. Each exponent is incremented.

	S	E	Denormalized M
A	1	10000011	110010100000000000000000
B	0	10001000	111000011000000000000000

We align the mantissas. We increment the first exponent by 5 and shift its mantissa to the right five times.

	S	E	Denormalized M
A	1	10001000	000001100101000000000000
B	0	10001000	111000011000000000000000

Now we do sign-and-magnitude addition treating the sign and the mantissa of each number as one integer stored in sign-and-magnitude representation.

	S	E	Denormalized M
R	0	10001000	110110110011000000000000

There is no overflow in mantissa, so we normalized.

	S	E	M
R	0	10000111	10110110011000000000000

The mantissa is only 23 bits because there is no overflow, no rounding is needed.

$$E = (10000111)_2 = 135, M = 10110110011$$

In other words, the result is

$$(1.10110110011)_2 \times 2^{135-127} = (110110110.011)_2 = 438.375$$

5. Using an 16-bit allocation, first convert each of the following numbers to two's complement, do the operation, and the convert the result to decimal. (10%)
   a. 161 + 1023 1184

																Carry	Decimal
						1	1	1									
	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	161
+	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1023
	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	1184

b. 161 – 1023 -862

																Carry	Decimal
								1									
	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	161
+	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	−1023
	1	1	1	1	1	1	0	0	1	0	1	0	0	0	1	0	−862

c. -161 + 1023 862

																Carry	Decimal
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	−161
+	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1023
	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	0	862

d. -161 – 1023 -1184

																Carry	Decimal
1	1	1	1	1	1				1	1	1	1	1				
	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	−161
+	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	−1023
	1	1	1	1	1	0	1	1	0	1	1	0	0	0	0	0	−1184

6. Compare and contrast the three methods for handling the synchronization of the CPU with I/O devices. (10%)
   In the programmed I/O method, the CPU waits for the I/O device. A lot of CPU time is wasted by checking for the status of an I/O operation.
   In the interrupt driven I/O method, the I/O device informs the CPU of its status via an interrupt.
   In direct memory access (DMA), the CPU sends its I/O requests to the DMA controller which manages the entire transaction.

7. A computer has 64 MB of memory. Each word is 4 bytes. How many bits are needed to address each single word in memory? (10%)
   We have 64 MB / (4 bytes per word) = 16 Mega words = $16 \times 2^{20} = 2^4 \times 2^{20} = 2^{24}$ words. Therefore, we need 24 bits to access memory words.

8. An imaginary computer has sixteen data register (R0 to R15), 1024 words in memory, and 16 different instructions (add, subtract, and so on). What is the minimum size of an instruction in bits if a typical instruction uses the following format: instruction M R2 (10%)
   We need 4 bits to determine the instruction ($2^4 = 16$). We need 4 bits to address a register ($2^4 = 16$). We need 10 bits to address a word in memory ($2^{10} = 1024$). The size of the instruction is therefore (4 + 4 + 10) or 18 bits.

9. What is the minimum size of the control bus in the computer in question 8 ? (10%)
   The control bus should handle all instructions. The minimum size of the control bus is therefore 4 bits ($\log_2 16$)

10. A computer uses memory-mapped I/O addressing. The address bus uses 10 lines (10 bits). If memory made up of 1,000 words, how many four-register controllers can be accessed by the computer. (10%)
    The address bus uses 10 lines which means that it can address 2^{10} = 1024 words. Since the memory is made of 1000 words and the system uses shared (memory-mapped I/O) addressing, 1024 − 1000 = 24 words are available for I/O controllers. If each controller has 4 registers, then 24/4 = 6 controllers can be accessed in this system.