Introduction to Computer Science-103

Quiz_2

Part A. Choice questions (20%)

1. In two's complement addition, if there is a final carry after the leftmost column addition, ___a___.
   a. discard it
   b. increase the bit length
   c. add it to the leftmost column
   d. add it to the rightmost column
2. ____a____ is a type of memory in which the user, not the manufacturer, stores programs that cannot be overwritten.
   a. PROM
   b. EEPROM
   c. ROM
   d. EPROM
3. In the ___a____ method for synchronizing the operation of the CPU with an I/O device, the CPU is idle until the I/O operation is finished.
   a. programmed I/O
   b. interrupt-driven I/O
   c. DMA
   d. isolated I/O
4. To flip all the bits of a bit pattern, make a mask of all 1s and then ___a____ the bit pattern and the mask.
   a. XOR
   b. AND
   c. OR
   d. NOT
5. In two's complement representation with a 4-bit allocation, we get ___a____when we add 5 to 5.
   a. -6
   b. 10
   c. -7
   d. -5

Part B. Short answer questions

1. Show the result of the following operations. (10%)

   a. NOT $[(99)_{16}$ OR $(00)_{16}]$

   NOT $[(99)_{16}$ OR $(99)_{16}]$ = NOT $[(10011001)_2$ OR $(10011001)_2]$

   = $(01100110)_2 = (66)_{16}$

   b. $(99)_{16}$ XOR $(2E)_{16}$

   $(10011001)_2$ XOR $(00101110)_2 = (10110111)_2 = (B7)_{16}$

2. Show the result of the following operations assuming that the numbers are stored in 16-bit two's complement representation. Show the result in hexadecimal notation. (10%)

   a. $(8011)_{16} + (0001)_{16}$

															Carry	Hexadecimal
												1			**Carry**	Hexadecimal
	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1		8011
+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		0001
	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0		8012

   b. $(E12A)_{16} + (9E27)_{16}$

																	Carry	Hexadecimal
1									**1**		**1**	**1**	**1**				**Carry**	Hexadecimal
	1	1	1	0	0	0	0	1	0	0	1	0	1	0	1	0		E12A
+	1	0	0	1	1	1	1	0	0	0	1	0	0	1	1	1		9E27
	0	1	1	1	1	1	1	1	0	1	0	1	0	0	0	1		**1**7F51

3. Show the result of the floating-point operations using Excess_127
   451.00 − 12.625 = 438.375. (10%)

   451 − 12.625 = $(111000011)_2$ − $(1100.101)_2$

   = $2^8 \times (1.11000011)_2 - 2^3 \times (1.100101)_2$.

   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).

$E_1 = 127 + 8 = 135 = (10000111)_2$ and $E_2 = 127 + 3 = 130 = (10000010)_2$

	S	E	M
A	0	10000111	11000011000000000000000
B	0	10000010	10010100000000000000000

The first two steps in UML diagram is not needed. Since the operation is subtraction, we change the sing of the second number.

	S	E	M
A	0	10000111	11000011000000000000000
B	1	10000010	10010100000000000000000

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	0	10001000	**1**11000011000000000000000
B	1	10000011	**1**10010100000000000000000

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

	S	E	Denormalized M
A	0	10001000	**1**11000011000000000000000
B	1	10001000	00000**1**10010100000000000

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	11011011001100000000000

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

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

4. Using an 8-bit allocation, first convert each of the following integers to two's complement, do the operation, and then convert the result to decimal. (10%)

   a. –19 – 23

(–00010011) – 00010111 = (–00010011) + (–00010111) = 11101101 + 11101001 =

	1	**1**	**1**		**1**			**1**		**Carry**	Decimal
		1	1	1	0	1	1	0	1		–19
+		1	1	1	0	1	0	0	1		–23
		1	1	0	1	0	1	1	0		–42

   b. 19 + 23

00010011 + 00010111

				1		**1**	**1**	**1**		**Carry**	Decimal
		0	0	0	1	0	0	1	1		19
+		0	0	0	1	0	1	1	1		23
		0	0	1	0	1	0	1	0		42

5. We need to unset the three leftmost bits and set the two rightmost bits of a pattern.

Show the masks and the operation. (5%)

$$\text{Mask1} = (00011111)_2 \quad \text{Mask2} = (00000011)_2$$
$$\textbf{Operation: } [\text{Mask1 } \textbf{AND } (xxxxxxxx)_2] \textbf{ OR } \text{Mask2} = (000xxx11)_2$$

6. What are the two methods for handling the addressing of I/O devices? What is the difference between them? (10%)

   The only difference is the instruction. If the instruction refers to a word in main memory, data transfer is between main memory and the CPU. If the instruction identifies an I/O device, data transfer is between the I/O device and the CPU. There are two methods for handling the addressing of I/O devices: isolated I/O and memory-mapped I/O.

7. How many bytes of memory are needed to store a full screen of data if the screen is made of 30 lines with 70 characters in each line? The system uses ASCII code, with each ASCII character stored as a byte. (10%)

   We need 30 × 70 = 2100 bytes.

8. An imaginary computer has sixteen data register (R0 to R15), 1024 words in memory, and 16 different instructions (add, subtract, and so on). If a typical instruction uses the following format: **instruction M R2.** If the computer uses the same size of word for data and instructions, what is the size of each data register? (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.

   Since the size of the instruction is 18 bits (See Solution to Exercise 43), we must have 18-bit data registers.

9. Using the instruction set of the simple computer in the following table, write the code for a program that performs the following calculation:

   $$B \leftarrow A - 2$$

   A and 2 are integers in two's complement format. The user types the value of A and the value of B is displayed on the monitor. The keyboard is assumed to be memory location $(FE)_{16}$, and the monitor is assumed to be $(FF)_{16}$.(10%)

Instruction	Code	Operands			Action
	d_1	d_2	d_3	d_4	
HALT	0				Stops the execution of the program
LOAD	1	R_D	M_S		$R_D \leftarrow M_S$
STORE	2	M_D		R_S	$M_D \leftarrow R_S$
ADDI	3	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
ADDF	4	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
MOVE	5	R_D	R_S		$R_D \leftarrow R_S$
NOT	6	R_D	R_S		$R_D \leftarrow \overline{R_S}$
AND	7	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ AND R_{S2}
OR	8	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ OR R_{S2}
XOR	9	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ XOR R_{S2}
INC	A	R			R $\leftarrow$ R + 1
DEC	B	R			R $\leftarrow$ R – 1
ROTATE	C	R	n	0 or 1	Rot_n R
JUMP	D	R	n		IF $R_0 \neq$ R then PC = *n*, otherwise continue

Key: R_S, R_{S1}, R_{S2}: Hexadecimal address of source registers
R_D: Hexadecimal address of destination register
M_S: Hexadecimal address of source memory location
M_D: Hexadecimal address of destination memory location
n: hexadecimal number
d_1, d_2, d_3, d_4: First, second, third, and fourth hexadecimal digits

The first column is not part of the code; it contains the instruction addresses for reference. We type A on the keyboard. The program reads and stores it as we press the ENTER key. Code for B ← A – 2

Step	Code(hexadecimal)	Description
1	1FFE	// RF ← MFE , Input A from keyboard to RF
2	240F	// M40 ← RF , Store A in M40
3	1040	// M40 ← R0 , Load A from M40 to R0
4	B000	// R0 ← R0 − 1, Decrement A
5	B000	// R0 ← R0 − 1, Decrement A
6	2410	// M41 ← R0 , Store The result in M41
7	1F41	// RF ← M41 , Load the result to RF
8	2FFF	// MFF ← RF , Send the result to the monitor
9	0000	// Halt