104 - Introduction to Computer Science - Quiz two

Name：_____ Student ID：_____

1. Show the result in hexadecimal notation of the following operations, showing your work. (10%)
   a. NOT[$(99)_{16}$ OR $(99)_{16}$] OR [NOT$(00)_{16}$]
   b. [$(99)_{16}$ AND $(33)_{16}$] OR [$(00)_{16}$ AND $(FF)_{16}$]
   a. NOT[$(10011001)_2$ OR $(10011001)_2$] OR [NOT$(00000000)_2$]
      = $(01100110)_2$ OR $(11111111)_2$ = $(11111111)_2$ = $(FF)_{16}$
   b. [$(10011001)_2$ AND $(00110011)_2$] OR [$(00000000)_2$ AND $(11111111)_2$]
      = $(00010001)_2$ OR $(00000000)_2$ = $(00010001)_2$ = $(11)_{16}$
2. Using an 8-bit allocation to do the operation, the former convert the integers to two's complement, the latter convert the integers to sign-and-magnitude representation, and then show the operated result in decimal notation, showing your work. (10%)
   a. –19 + 23
   b. 17 – 32
   a. $(-00010011)_2$ + $(00010111)_2$
      = $(11101101)_2$ + $(00010111)_2$ = $(00000100)_2$ = $(4)_{10}$
   b. 17 = $(00010001)_2$ and 32 = $(00100000)_2$.
      Operation is subtraction, sign of B is changed. S = AS XOR BS = 1. The two's complement of B is $(11100000)_2$.
      There is no overflow … so $(10001111)_2$ = $(-15)_{10}$

A_S	0		0	0	1	0	0	0	1
B_S	1	+	1	1	0	0	0	0	0
			1	1	1	0	0	0	1
R_S	1		0	0	0	1	1	1	1

3. 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, showing your work. (10%)
   a. $(712A)_{16}$ + $(9E00)_{16}$ The result is not valid because of overflow.
   b. $(E12A)_{16}$ + $(9E27)_{16}$ The result is not valid because of overflow.
   a. $(712A)_{16}$ = $(0111\ 0001\ 0010\ 1010)_2$ and $(9E00)_{16}$ = $(1001\ 1110\ 0000\ 0000)_2$

1	1	1	1													Carry
	0	1	1	1	0	0	0	1	0	0	1	0	1	0	1	0
+	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0

   b. $(E12A)_{16}$ = $(1110\ 0001\ 0010\ 1010)_2$ and $(9E27)_{16}$ = $(1001\ 1110\ 0010\ 0111)_2$

1									1		1	1	1			Carry
	1	1	1	0	0	0	0	1	0	0	1	0	1	0	1	0

+	1	0	0	1	1	1	1	0	0	0	0	1	0	1	1	1
	0	1	1	1	1	1	1	1	0	1	0	1	0	0	0	1

4. Show the result of the floating-point operations 33.1875 – 0.4375 using 32-bit IEEE format, showing your work. (10%)

Step.1 These two numbers are stored in floating-point format. We need to remember that each number has a hidden 1.

A ➔ 33.1875 = 0 10000100 00001001100000000000000

B ➔ 0.4375 = 0 01111101 11000000000000000000000

Step.2 The first two steps in UML diagram is not needed. Since the operation is subtraction, we change the sing of the second number.

A ➔ 0 10000100 00001001100000000000000

B ➔ 1 01111101 11000000000000000000000

Step.3 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.

A ➔ 0 10000101 10000100110000000000000

B ➔ 1 01111110 11100000000000000000000

Step.4 We align the mantissas. We increment the second exponent by 7 and shift its mantissa to the right seven times.

A ➔ 0 10000101 10000100110000000000000

B ➔ 1 10000101 00000001110000000000000

Step.5 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.

R ➔ 0 10000101 10000011000000000000000

Step.6 There is no overflow in mantissa, so we normalized.

R ➔ 0 10000100 00000110000000000000000

Step.6 E = $(10000100)_2$ = 132, M = 0000011

$(1.0000011)_2 \times 2^{132-127} = (100000.11)_2 = \underline{32.75}$

5. Describe the applications of four logic operations and show the mask should be used (using examples or variables to represent). (5%)

a. Complementing ➔ NOT

Applying this operator to a pattern changes every 0 to 1 and every 1 to 0. This is sometimes referred to as a one's complement operation and not needs the mask.

b. Unsetting specific bits ➔ AND

This applications is to unset (force to 0) specific bits in a bit pattern. The 0-bits in the mask unset the corresponding bits in the first input. For example, if an image is using only one bit per pixel (a black and white image), then we can make a specific pixel black using a mask and the AND operator. Another example is that use a mask to unset the five leftmost bits of a

pattern, the pattern of the mask is 00000111.

c. Setting specific bits ➔ OR

This application is to set (force to 1) specific bits in a bit pattern. The 1-bits in the mask set the corresponding bits in the first input, and the 0-bits in the mask leave the corresponding bits in the first input unchanged. For example, if an image is using only one bit per pixel (a black and white image), then we can make a specific pixel white using a mask and the OR operator. Another example is that use a mask to set the five leftmost bits of a pattern, the pattern of the mask is 11111000.

d. Flipping specific bits ➔ XOR

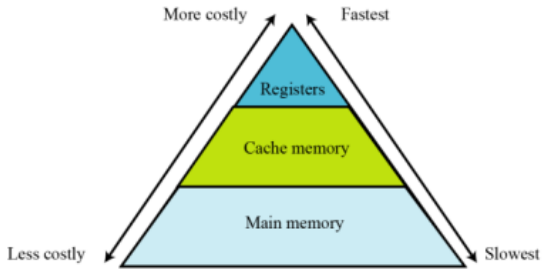
This application is to flip (complement) specific bits in a bit pattern. The 1-bits in the mask flip the corresponding bits in the first input, and the 0-bits in the mask leave the corresponding bits in the first input unchanged. For example, we use a mask to flip the five leftmost bits of a pattern, the pattern of the mask is 11111000.

6. What is the difference between logical and arithmetic shift operations? And describe that how to use the shift operation to multiply an integer by 8 and to divide an integer by 4 respectively. (6%)

① A logical shift operation is applied to a pattern that does not represent a signed number. And arithmetic shift operations assume that the bit pattern is a signed integer in two's complement format.

② Arithmetic right shift divides an integer by 2 (the result is truncated to a smaller integer). To divide an integer by 4, we apply the arithmetic right shift operation twice. And arithmetic left shift multiplies an integer by 2. To multiply an integer by 8, we apply the arithmetic left shift operation three times.

7. Define the memory hierarchy and describe that why cache memory is so efficient despite its small size. (5%)



① Use a very small amount of costly high-speed memory where speed is crucial. The registers inside the CPU are of this type. Use a moderate amount of medium-speed memory to store data that is accessed often. Cache memory is of this type. Use a large amount of low-speed memory for data that are is accessed less often. Main memory is of this type. ② The answer lies in the "80-20 rule". It has been observed that moat computers typically spend 80 percent of their time accessing only 20 percent of the data. In other words, the same data is accessed over and over again. Cache memory, with its high speed, can hold this 20 percent to make access faster at least 80 percent of the time.

8. What are the two methods for handling the addressing of I/O devices? What is the difference between them? (5%)

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.

9. How many bytes of memory are needed to store a full screen of data if the screen is made of 24 lines with 80 characters in each line? The system uses ASCII code, with each ASCII character stored as a byte. (5%)

   We need 24 × 80 = 1920 bytes.

10. We know that the architecture and organization of computers has gone through many changes in recent decades. So describe pipelining and parallel processing respectively. What is the difference between them and their purposes? (6%)

    Pipelining allows different types of phases belonging to different cycles to be done simultaneously. And a single computer can have multiple control units, multiple ALU units and multiple memory units to perform several instructions in parallel. Pipelining is running in the control unit, while the computer can have multiple control units in parallel processing. Their purposes are the same. They can increase the throughput of the computer.

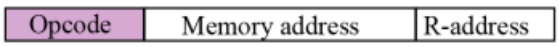
11. The CPU and memory are normally connected by three groups of connections, each called bus. But I/O devices cannot be connected directly to the buses that connect the CPU and memory, please describe the reason. And what do the I/O devices use to attach to the buses? Please give two examples. (6%)

    Because the nature of I/O devices is different from the nature of CPU and memory. I/O devices are electromechanical, magnetic, or optical devices, whereas the CPU and memory are electronic devices. And I/O devices are therefore attached to the buses through input/output controllers or interfaces. For example, small computer system interface (SCSI), Firewire, Universal Serial Bus (USB) and HDMI.

12. 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 10 bits to address a word in memory (2^{10} = 1024). We need 4 bits to address a register (2^4 = 16). The size of the instruction is therefore (4 + 10 + 4) or 18 bits.

    Since the size of the instruction is 18 bits, we must have 18-bit data registers.



13. Using the instruction set of the simple computer in the following table, write the code for a program that performs the calculation B ← A – 2. The letter 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 $(FF)_{16}$. (12%)

Table 5.4 List of instructions for the simple computer

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 $R_0 =$ 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