CSE-112 Quiz 1 Rubric

Q1: **[7 marks]**

Consider the following operation: **-511 - 505**.

(a) Write the 1's complement representation of both the operands. **[1+1 marks]**

(b) Also, state the rule for getting 2's complement representation from 1's complement representation. **[1 mark]**

(c) Use this rule to get 2's complement representation of the two numbers. **[1 mark]**

(d) Perform the operation in 2's complement form itself. Show your computation. **[2 marks]**

(e) Report the minimum bits required for the output of the computation correctly in 2's complement notation. **[1 mark]**

Solution:

a) **[3 marks]**

Legend:

- ***Sign bit (signed representation)***
- ***1's complement representation***
- ***2's complement representation***

1's complement representation:

$-(511)_{10} = -(111\ 111\ 111) = (1\ 111\ 111\ 111)_2$
$\sim (1\ 000\ 000\ 000)_2$

$-(505)_{10} = -(111\ 111\ 001) = (1\ 111\ 111\ 111)_2$
$\sim (1\ 000\ 000\ 110)_2$

b) ***2's complement = 1's complement + 1***

c) $-(511)_{10} \sim (1\ 000\ 000\ 000)_2 \sim (1\ 000\ 000\ 001)_2$

$-(505)_{10} \sim (1\ 000\ 000\ 110)_2 \sim (1\ 000\ 000\ 111)_2$

d)

The rule for overflow detection used here: $c_9 \oplus c_{10} \sim (1 \oplus 0) \sim 1$

```
   1 000 000 001
+  1 000 000 111
---------------
1 0 000 001 000
```

Therefore, an overflow has occurred.
(so our answer here is inaccurate)

Ignoring the msb ~ 0 000 001 000 ~ $+(000\ 001\ 000)_2 \sim (8)_{10}$

e) **11 bits are required in 2's complement representation**

i.e.

$-(511)_{10}$ ~ $(1\ 000\ 000\ 001)_2$
~ sign extension ~ $(11\ 000\ 000\ 001)_2$

$-(505)_{10}$ ~ $(1\ 000\ 000\ 111)_2$
~ sign extension ~ $(11\ 000\ 000\ 111)_2$

Q2: [1 + 2 + 2 + 3 = 8 marks]

Consider the following IEE754 Double Precision Floating Point Number:
1100_0010_0001_0010_1010_0100_0000_0000_0000_0000_0000_0000_0000_0000_0000_
0000. Here, '_' are added for clarity.
Show your computation.

a. What is the sign of this number (+ve or -ve)?

Since MSB is 1 then sign should be -ve

b. What is the exponent part (in decimal)?

In Double precision format, we have bias of 11 bits therefore, exponent bits will be
10000100001 represents 1024+32+1 = 1057-1023(bias) = 34

c. What is the mantissa part (in binary)?

In Double precision, we have to assign 52 bits i.e.
0010101001000000000000000000000000000000000000000000

d. Convert the number to decimal notation.

For the given binary number, corresponding decimal number will be $(1+0.1650390625) * 2^{34}$
= **$-2.0015218688*10^{10}$**
(any answer close to this is acceptable{within error margin of 0.001})

Q3: [14 marks]

Consider an ISA which has **32-bit** long instructions. There are two types of instruction with the following syntaxes:

1. Type A: **<Q bit opcode> <P-bit address> <5 bit register>**

2. Type B: **<Q bit opcode> <R bits filler> <5 bit register> <5 bit register>**

Type A instructions have 3 fields: the opcode, a memory address, and a register. Type B instructions have three fields: the opcode and two registers. The filler bits are useless bits to ensure that both types of instructions are of the same length of 32 bits. Suppose the ISA supports the address space of **8 MegaBytes** with byte-addressable memory. Given this, answer the following questions:

Solution(a,b,c,d,e):

a. How many bits are used to represent an address in this ISA?
Since ISA supports address space of **8 MB = 2^{23}.** Hence the number of bits used to represent an address in ISA is $\log_2(2^{23}) = 23$.

b. Determine **P**?
23

c. Determine **Q**?
4

d. Determine **R**?
18

e. What is the maximum number of instructions that this ISA can support? (Hint: think opcode…)
$2^4 = 16$

f.Consider a new type of instruction for multiply operation (neither of type A nor type B). The input for this instruction is the registers containing the numbers. Consider the case to be of the following type.

<Q bit opcode> < Not-Known> <5 bit source register 1 > <5 bit source register 2>

In this instruction, the value from source register 1 and source register 2 is taken and multiplied. If each register can hold 32 bits each, determine the number of bits needed to represent the result. Discuss different ways in which the result can be stored in the register bank.

Solution(f):

In the typical scenario, 64 bits are needed to represent the result of the multiplication of two 32 bit numbers.

In the register bank, the result can be placed in the following manner:

1. If the processor strictly allows only 32-bit values, put only the initial 32 bits in the register bank. This lowers the usage of registers.
2. If 64-bit values are to be considered, one way can be to store the results in consecutive registers. In this case, we need to specify only one register as output, the other register being implicity declared by the hardware.
3. The last way to place the result is to put it in two different registers. In this case, as registers are not related to each other, both the registers should be declared in the instruction.

Q4: Multiplication & Division **[7 marks]**

a. Multiply unsigned numbers 1000_1100_1100 ('_' are added for clarity of representation) by $(80)_{10}$. Report the result in 32-bit unsigned format (add zeros before MSB to extend the result to 32 bits). Explain your approach. **[3 marks]**

Solution(a):

80 can be given as the sum of 64 and 16 (both power of 2). If a number 'a' is multiplied by 80, it can be written as (a X 64 + a X16) that is left shift and addition.
Students should mention that multiplication is just a left shift operation.

Students should mention the shift is by 6 bits (for 64) and 4 bits (for 16).
to mention why— (2^ 6=64) and (2^ 4 =16)

```
        0010_0011_0011_0000_0000
+       0000_1000_1100_1100_0000
_____________________________________
=       0010_1011_1111_1100_0000
_____________________________________
```

To extend the result to result to 32 bits, we append 0s after the MSB. The answer is 0000_0000_0000_0010_1011_1111_1100_0000.

b. Consider the following number represented in IEEE754 notation: 0100_1010_1101_1100_1001_0000_0000_0000. ('_' are added for clarity). What would be the result in IEEE754 notation if we divide this number by 8?
Compute the answer **without converting the given number to decimal**.
Explain your approach. [3 marks]

Solution(b):

In the case of the IEEE754 floating point number, when dividing with 8, we add -3 to the exponent part of the IEEE format

Given number = 0100_1010_1101_1100_1001_0000_0000_0000.
Sign = 0
Exponent = 1001_0101
New Exponent = 1001_0101 - 3

= 1001_0010
Mantissa = 101_1100_1001_0000_0000_0000
New Number after dividing by 8 = {Sign, New Exponent, Mantissa}
= 0100_1001_0101_1100_1001_0000_0000_0000

Q5: [5 marks]

1. False
2. True (performance is considered by default, in terms of speed)
3. False
4. False
5. True

Q6. [7 marks]

a) Explain two approaches for finding the overflow in 2's complement addition **[2 Marks]**
b) Find two expressions that result in the overflow condition in 2's complement addition of two "k" bit numbers(details given below) **[2 Marks]**
c) Draw the circuits for the expressions and compare them **[2 Marks]**
(Note: *You can ONLY use 2-input NAND gates for making the circuits*)
d) Which circuit should be preferred and why? **[1 Mark]**

The k-bit numbers are $A(a_{k-1}a_{k-2}\ldots a_1a_0)$ and $B(b_{k-1}b_{k-2}\ldots b_1b_0)$.
Assume that carryovers $C(c_kc_{k-1}\ldots c_1c_0)$ and arithmetic solution $S(s_{k-1}s_{k-2}\ldots s_1s_0)$ are given to you.
Initial carryover ($\mathbf{c}_{in}$) to the adder is c_0. For simplicity, assume $c_0 = 0$.

Solution:

(a) Two Conditions are :

(i) Two negative numbers are added and an answer comes positive

(ii) Two positive numbers are added and an answer comes as negative

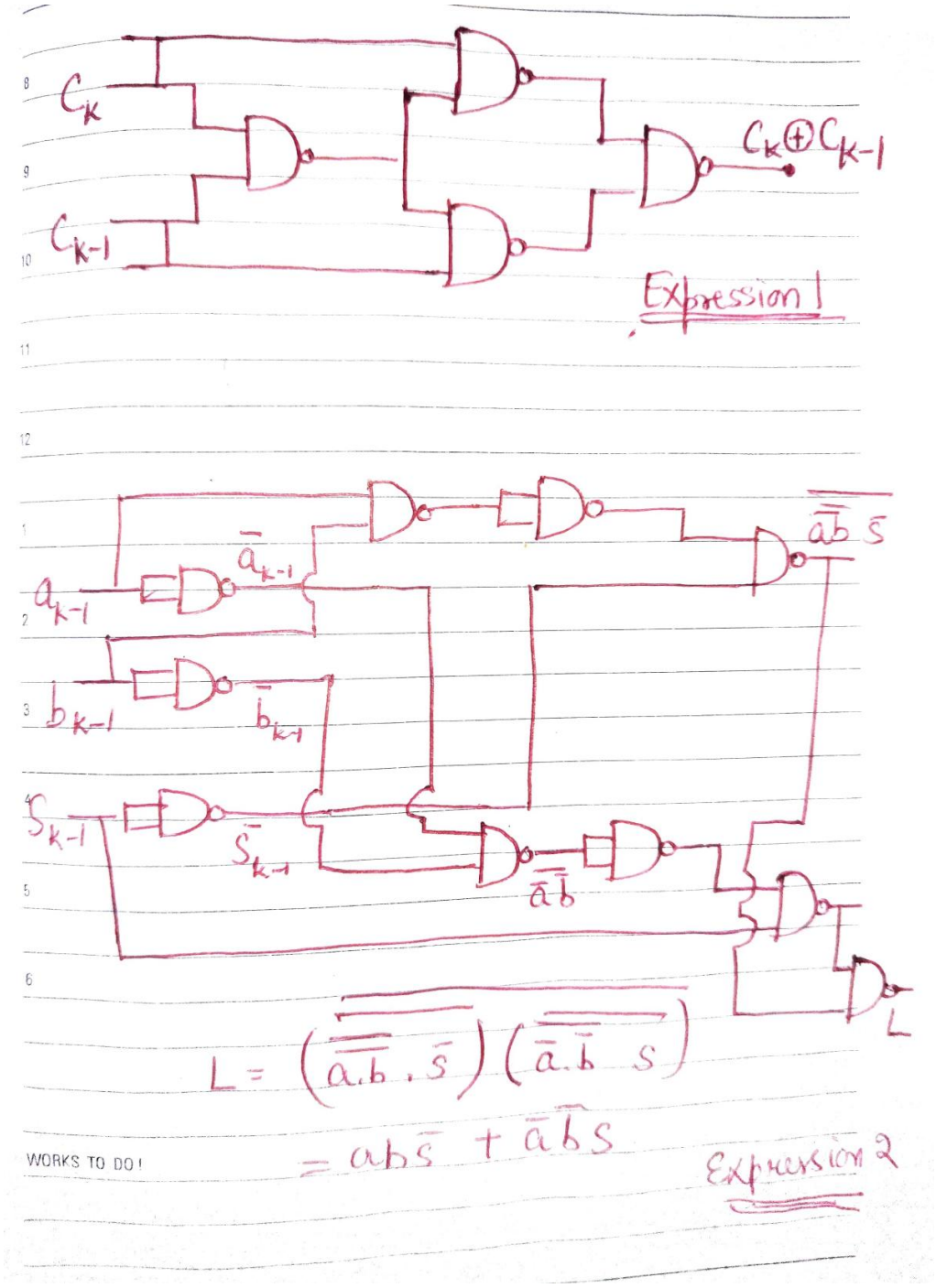
In both the above conditions, it means that the Sign bit of final answer is different from the sign bit of Summands, hence it imposes a requirement of extra bit to represent sign bit as the existing limited bits for the final SUM is not sufficient to represent the sign bit as well as the magnitude of addition

(b) Conditions:

(i) Overflow = $c_k \oplus c_{k-1}$

(ii) Overflow = $a_{k-1}.b_{k-1}.\bar{s}_{k-1} + \bar{a}_{k-1}.\bar{b}_{k-1}.s_{k-1}$

(c) Circuit Diagram :



(d) Circuit for expression 1 is better than for expression 2
Reasoning :

1. **AREA {number of gates used in the circuit}**
   Circuit for expression 1 takes 4-5 2-input NAND gates and that for expression 2 takes 10 2-input NAND gates.
2. **Time Delay {number of layers of NAND gates}**
   Circuit for expression 1 takes 3 layers of 2-input NAND gates and that for expression 2 takes 5 layers of 2-input NAND gates.

Q7. [8 marks]

(a) Draw the circuit for a 2-stage Carry Select Adder(CSA) for adding two 8-bit unsigned binary numbers. **[4 marks]**
Note: ***Just use 2:1 MUX and 4-bit Full adders(FAs) as complete blocks ONLY for the circuit, no need for internal or gate level circuits (The MUX and FA can take multiple bits as a single input wire)***

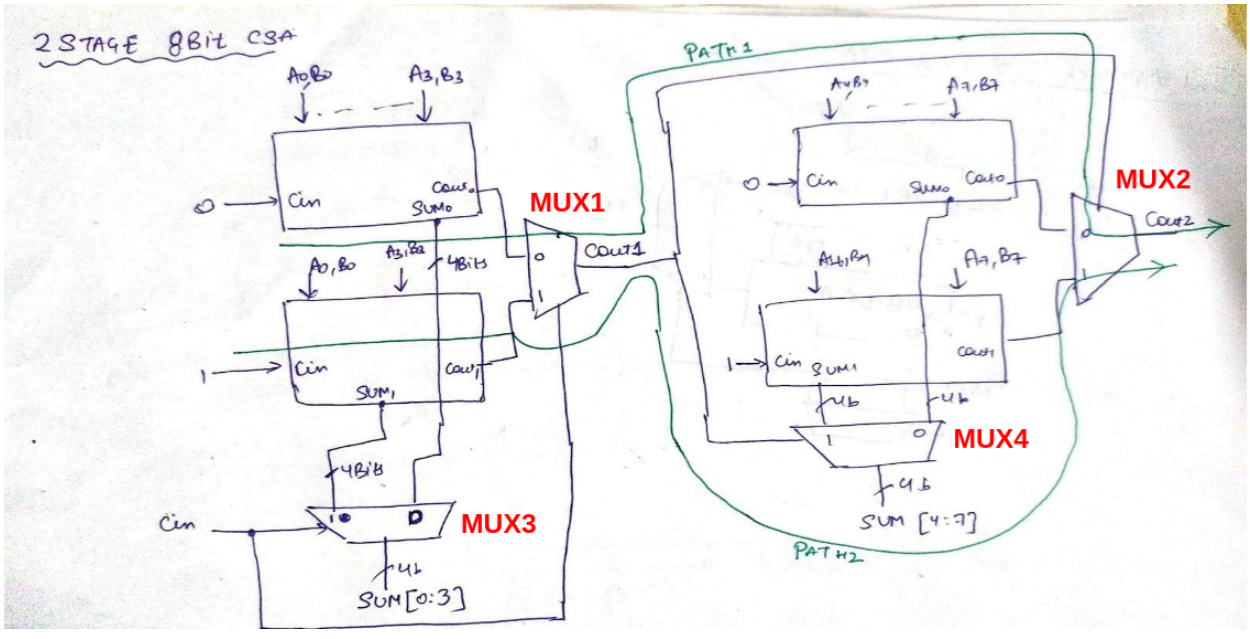
(b) Mark the **critical path** in the above circuit. (***Hint :*** **Critical path** is the **path with maximum delay** traversed by a binary signal in a circuit from source to destination. The delays can be approximated by the number elements{blocks or gates} which lie on the corresponding path) **[2 marks]**

(c) Give an expression for the time delay accumulated in the critical path **[2 marks]**
Assume these delays for the expression asked in **Part (c) :**

- time_delay(2:1 MUX) = t_m
- time_delay(4-bit Full Adder) = t_f

Solution:

(a) Circuit is:



(b) 3 Possible Paths:
(Here, the assumption is that A[7:0], B[7:0] & Cin are applied simultaneously on the circuit)

(i) Critical Path 1: 1st 4-bit FA to MUX1 to MUX2

- Here, the assumption is that the propagation delay of MUX to transfer 0th or 1st input to output(given that Select line is constant) is considered as separate from FA delay and is not negligible and hence counted in the given **"t_m"** delay
- First MUX1 gets Select line as soon as Cin comes

- First FA takes time to calculate Cout1 an Cout0 parallely and then only reaches to the input of the MUX1 after **"t_f"** time and hence becomes the part of critical path
- Now, the inputs 0th or 1st take propagation delay to travel to COUT1
- Now, after 1st FA and 1st MUX delay, COUT1 is obtained which traverses to 2nd MUX and then only the final COUT2 is obtained.
- Now, the assumption is that the Select line of MUX2 to COUT2 delay is also counted under **"t_m"** delay
- 2nd FA is not considered as its results are readily available for Cin = 1 & 0 when the results of 1st FA were being calculated
- Hence, 2nd FA was waiting for valid COUT1 to come and select appropriate Sums and

(ii) Critical Path 2: 1st 4-bit FA to MUX2

- Here, the assumption is that the Propagation delay pf MUX from either 0th or 1st input to output(given that Select line is constant) is included in the FA delay **"t_f"** and/or is negligible.
- Now, 1st FA takes **"t_f"** time to get valid output at COUT1
- The 2nd FA had already calculated the Sums and Couts corresponding to Cin = 0 and 1, hence, it was waiting for COUT1 to select which value to propagate to COUT2 as the final output

(iii) Critical Path 3: 1st 4-Bit FA to MUX4

- This is same as (ii) path , just that we consider MUX3 which outputs the SUM[4:7] instead of the MUX which outputs COUT2

(c) 3 Delays are:

(i) Path1 : $2*t_m + t_f$

(ii) Path2 : $t_m + t_f$

(iii) Path3 : $t_m + t_f$ (it is assumed that MUX3 and MUX4 with 4bit inputs are parallelly connected of 4x2-is-to1 MUXes and hence incur equal delay as soon as the COU1 comes as select line)

Q8. [4 marks]

a. Determine 'x' if $(113)_x = 23_{10}$

Solution:

$1 * x^2 + 1 * x + 3 = 23$

$x^2 + x - 20 = 0$

$x^2 + 5 * x - 4 * x - 20 = 0$

$(x + 5)(x - 4) = 0$

$x = -5, 4$

As the base cannot be negative, $x = 4$.

b. Determine 'x' if $(127)_x = 46_{16}$

Solution:

$1 * x^2 + 2 * x + 7 = 4 * 16 + 6$

$x^2 + 2x + 7 = 70$

$x^2 + 2x - 63 = 0$

$x^2 + 9x - 7x - 63 = 0$

$(x + 9)(x - 7) = 0$

$x = -9, 7$

No such value of "x" exists for the condition to be true as "x = 7" = "7" in 127 which is not possible.

THE END