CS2100 Computer Organization
2021/22 Semester I
Make-up Midterm Assessment

Question 0. (3 MARKS)

Write your name, student number (AxxxxxxY) and tutorial group on your answer script. (1 mark each)

Question 1 (10 MARKS)

We are given the following MIPS assembly program that processes and array A, whose base address is in $2. The variable "len" is mapped to $3 and contains the number of elements in A, while variable "x" is mapped to $4.

```
        addi $4, $0, 0
        sll $5, $3, 2
        add $6, $2, $5
        addi $7, $2, 0
a:      lw $8, 0($7)
        andi $9, $8, 4
        bne $9, $0, b
        addi $4, $4, 1
b:      addi $7, $7, 4
        slt $8, $7, $6
        bne $8, $0, a
```

a. If A has 5 elements, in the best case, how many instructions are executed by the program above? (3 marks)

Before the loop: 4 instructions. (1 mark)
Inside loop: 6 instructions x 5 (2 marks)
Total = 4 + 5 x 6 = 34 instructions

b. If A has 5 elements, in the worst case, how many instructions are executed by the program above? (3 marks)

Before the loop: 4 instructions (1 mark)
Inside loop: 7 instructions x 5 (2 marks)
Total = 4 + 5 x 7 = 39 instructions

c. Rewrite this program in C, following the code above closely. You may declare additional variables like loop indices (indexes) to complete your program, but assume that A and count have been properly declared and initialized, and that x has been declared.

Note that if your program declares unnecessary variables, is unnecessarily complicated or deviates unnecessarily from the structure of the code above, marks will be deducted. (4 marks)

```
x = 0
do{
        if(A[i] & 0b100)
                x = x + 1;
} while(i<len);
```

Note: The original code is executed at least once even if count=0. Using anything other than do..while is incorrect.

Question 2 (7 MARKS)

We now explore the idea of a "column parity" of an array of words. The idea of a column parity for five words is illustrated below, in which a 6th word containing the parity for each column of bits is produced.

Our actual MIPS processor uses 32-bit words, but here we show 8-bit words for simplicity. Here we use odd parity. For ease of reading a space has been inserted between each column.

Word 0:	0 1 1 0 1 1 1 0
Word 1:	0 1 0 1 1 1 0 1
Word 2:	1 1 0 1 1 0 1 0
Word 3:	1 0 1 0 1 0 1 0
Word 4:	1 0 1 1 0 1 1 0
Parity:	0 0 0 0 1 0 1 0

The "Parity" word consists of parity bits for each column of bits. So bit 7 of "Parity" is the odd parity for all the bit 7 of Word 0 to 4, bit 6 of "Parity" is the odd parity for all the bit 6 of Word 0 to 4, etc. Thus the "Parity" for each (vertical) column is set to 1 or 0 to make the number of 1's in that column odd.

We are given an array B. The data occupies the first to second last element of B. We will put the parity in the last element of B. This idea is shown below for n=6, using the example above:

n = 6

Array B:

Word 0 0110 1110	Word 1 0101 1101	Word 2 1101 1010	Word 3 1010 1010	Word 4 1011 0110	Parity 0000 1010

Given that the base address of B is in $s0 and *n* is mapped to $s1, write a MIPS assembly language program to compute the **ODD** column parity of the data in array B and write it to the last element, as described above.

```
        addi $t0, $s1, -2       ; Find address of last data element
        sll $t0, $t0, 2         ;
        add $t0, $s0, $t0       ;
        add $t1, $s0, $zero     ; Address of first element
        lui $v0, 0xFFFF         ; Set our initial parity value.
        ori $v0, 0xFFFF         ; Can also just NOT the parity at the end
loop:   slt $t2, $t1, $t0       ; Have we reached the end of the data?
        beq $t2, $0, end        ; Yes
        lw $t2, 0($t1)          ; No, load the data
        xor $v0, $v0, $t2       ; XOR to get the parity.
        addi $t1, $t1, 4
        j loop
        sw $v0, 0($t1)          ; Store the parity
```

Question 3 (10 MARKS)

The table below shows the execution times of various units in the MIPS Datapath, which is identical to Question 2 in Tutorial 5. See the Appendix for a diagram of the datapath.

Inst-Mem	Adder	MUX	ALU	Reg-File	Data-Mem	Control/ ALUControl	Left-shift/ Sign-Extend/ AND
400ps	100ps	30ps	120ps	200ps	350ps	100ps	20ps

The "bne" operation takes the same amount of time as the "beq" operation, and the "sll" operation does the actual shift operation in the ALU. Just for this question alone, assume that the "andi" operation sign-extends the immediate value, and for the "sll" operation, the "shamt" portion of the instruction is connected to the ALU, and incurs the regular ALU timing to execute.

a. Complete the latencies for the following instructions in picoseconds. Where applicable you may use latencies from Question 2 of Tutorial 5. (7 marks)

Instruction	Latency
add	**980**
addi	**950**
andi	**950**
bne	**800**
lw	**1300**
sll	**980**
slt	**980**

Note:
andi: R[rt] = R[rs] AND signextend(immed)
Timing = IF + regFile + ALU + MUX + regFile
= 400 + 200 + 120 + 30 + 200 = 950

sll does R[rd] = R[rt] << shamt, NOT R[rd] = R[rs] << shamt.
Timing = IF + regFile + MUX + ALU + MUX + regFile
= 400 + 200 + 30 + 120 + 30 + 200 = 980ps

b. Compute the worst case running time in picoseconds for the program in Question 1, for an array of 5 elements. You do not need to show your working, but if you do you MAY get partial credit even if your answer is incorrect. (3 marks)

Instruction	Timing
addi $4, $0, 0	950
sll $5, $3, 2	980
add $6, $2, $3	980
addi $7, $2, 0	950

Total time: 3860 ps

Instruction	Timing
lw $8, 0($7)	1300
andi $9, $8, 4	950
bne $9, $0, b	800
addi $4, $4, 1	950
addi $7, $7, 4	950
slt $8, $7, $6	980
bne $8, $0, a	800

Time for 1 iteration: 6730 ps
Time for 5 iterations: 33650 ps
Total time: 33650 + 3860 = 37510 ps

Question 4 (10 MARKS)

In a revolutionary new computer technology invented by two scientists, Yucan and Dueet, they are able to represent reliably 4-value digit they call a "qubit". That is, a single qubit can represent the values 0, 1, 2 and 3. We denote a number x in qubits as x_q.

In their very first computer called the Yucan Dueet ONE (otherwise simply called the ONE), they have created an 8-qubit fixed-point format with three qubits for the integer portion, and five qubits for the fraction portion. For the moment the ONE can only represent positive numbers.

a. What is the largest number that the ONE can represent in this number system? Write your answer in decimal, to 5 decimal places. (2 marks)

Largest number is 333.33333
= 3 x 4^2 + 3 x 4^1 + 3 + 3 x 4^(-1) + 3 x 4^(-2) + 3 x 4^(-3) + 3 x 4^(-4) + 3 x 4^(-5)
= 63.99902

b. What is the smallest non-zero number the ONE can represent in this number system? Write your answer in decimal, to 5 decimal places. (3 marks)

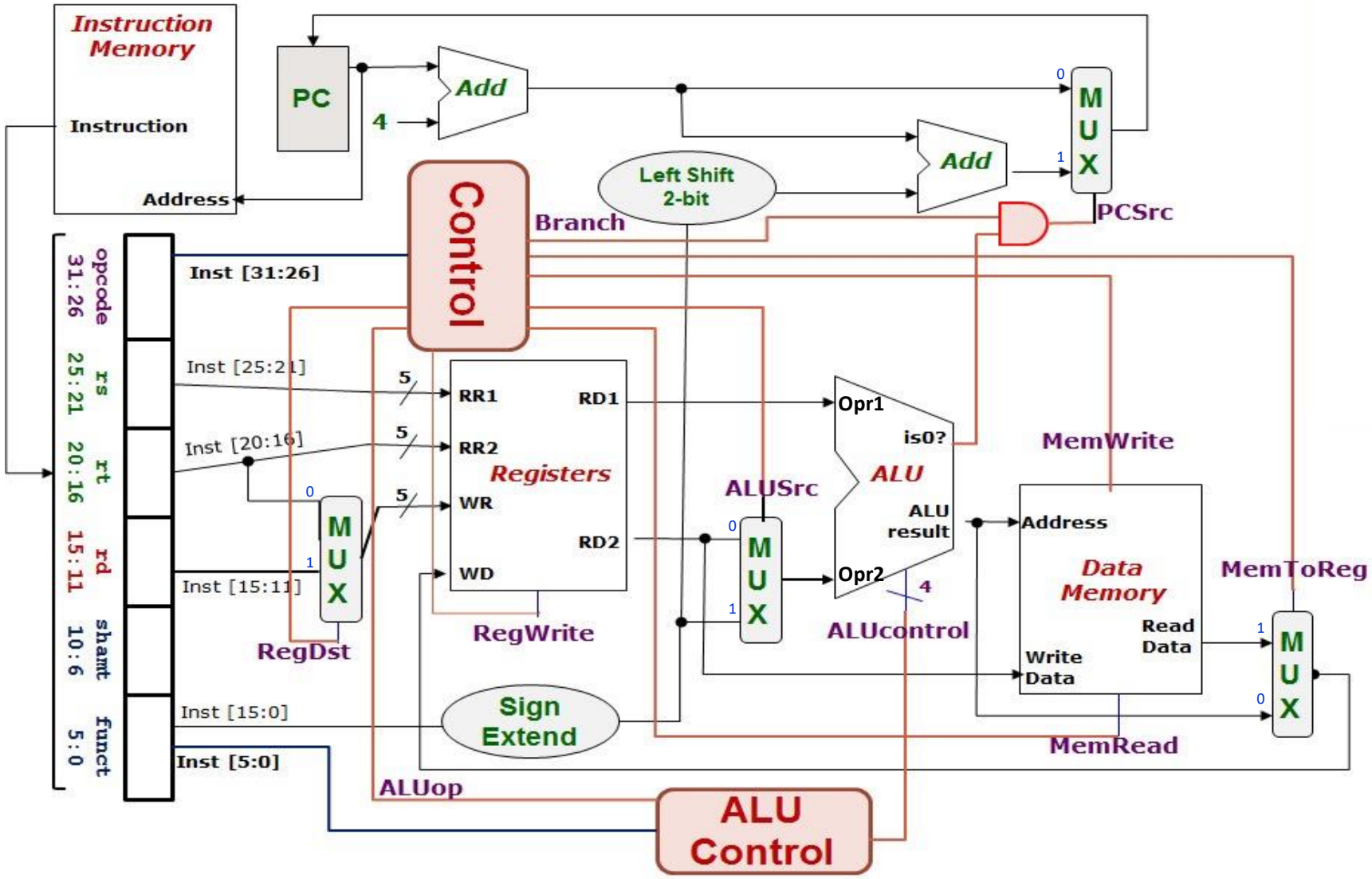
Smallest non-zero value = 1 x 4^-5 = 0.00098

c. Write the value 13.312_{10} in this number system. (3 marks)

Using repeated divide and repeated multiply, 31.10333_q

d. Convert your answer in c. back to decimal. What is the absolute error? Write your answer in decimal, to 5 decimal places. (2 marks)

31.10333 = 13.31152
13.312 – 13.31152 = 0.00048



Inst-Mem 400ps

Adder 100ps

MUX 30ps

ALU 120ps

Reg-File 200ps

Data-Mem 350ps

Control/ALU Control 100ps

Lshft/signext /AND 20ps