Please write your name and Kerberos on your solutions.

The only material you may access for this exam is this pdf, the RISC-V ISA handout (https://6004.mit.edu/web/_static/spring21/resources/references/6004_isa_reference.pdf) and the quiz 1 answer template if you would like to use it (https://6004.mit.edu/web/_static/spring21/resources/quizzes/q1_answer_template.pdf).

Each subproblem is labelled with (Label: X), (e.g., Label: 2A_1). If you are writing your solutions on blank paper, please make sure you write down the label of each subproblem together with its solution and circle each answer. So, for example, if your computed value for problem 2A_1 is 5, you would write:

\[ \text{2A}_1: 5 \]

NOTE: You must show your work to get credit for your answers.

Problem 1. Honor Code Agreement (1 point)

We are using the honor system during this exam, and ask that you accept the following terms of this honor system:

1. You will not share the exam with anyone
2. You will not discuss the material on the exam with anyone until after solutions are released
3. You understand that the exam is closed book and that you may only use reference material provided with the exam
4. You understand that you must show your work to get credit for your answers.

(label: 1) I will abide by the above terms (circle one): Agree / Do Not Agree
Problem 2. Binary Arithmetic (11 points)

(A) (4 points) Express the following decimal values as hexadecimal numbers encoded in 8-bit 2’s complement. **If a value cannot be expressed in 8-bit 2’s complement, write “Not Possible”**.

(label: 2A_1) -47 in 8-bit 2’s complement hexadecimal (0x): ________________

(label: 2A_2) 128 in 8-bit 2’s complement hexadecimal (0x): ________________

(B) (2 points) Calculate the 8-bit 2’s complement sum of the numbers 0xA9 and 0x70 using 8-bit 2’s complement arithmetic. Provide your answer in 8-bit 2’s complement notation. **If the result cannot be expressed in 8-bit 2’s complement, write “Not Possible”. You must show your work using 2’s complement arithmetic**.

(label: 2B) 0xA9 + 0x70 in 8-bit 2’s complement binary notation (0b): ________________

(C) (1 point) What is the minimum number of bits required to represent the number 47 in 2’s complement notation?

(label: 2C) Minimum number of bits to represent 47 in 2’s complement notation: _______
(D) (4 points) Perform an arithmetic right-shift by 2 on the 12-bit 2’s complement number 0xD18. What is the result in hexadecimal, and what is the equivalent mathematical operation of this shift operation?

(label: 2D_1) Result of shift in hexadecimal (0x): __________________________

(label: 2D_2) Equivalent mathematical operation of right-shift by 2: __________________________
Problem 3. RISC-V Assembly (16 points)

(A) (3 points) Fill in the blanks in the following C code given its translated RISC-V assembly code.

// arr = an array of integers
// n = length of array
int fn(int arr[], int n) {
    int count = 0;
    for (int i = 0; i < n; i = i + 1) {
        if (____________________) {
            count = count + 1;
        }
    }
    return ________________;
}

RISC-V:
fn:   li a2, 0
      li a3, 0

L:     slli a6, a3, 2
       add a6, a6, a0
       lw a4, 0(a6)  // (label: 3A_1)
      andi a5, a4, 1
       bne a5, zero, endL
       addi a2, a2, 1
endL:  addi a3, a3, 1
       blt a3, a1, L
       mv a0, a2
       ret

       (label: 3A_1)  if (____________________)

       ...  

       (label: 3A_2)  return ________________;


(B) (1 point) What is the hexadecimal encoding of the highlighted instruction: \texttt{lw a4, 0(a6)} in the assembly code snippet from part (A)? You may use the template below to help with the encoding. \textbf{Recall that a4 translates to x14 and a6 translates to x16.}

<table>
<thead>
<tr>
<th>[31: 20]</th>
<th>[19:15]</th>
<th>[14:12]</th>
<th>[11:7]</th>
<th>[6:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>imm[11:0]</td>
<td>rs1</td>
<td>funct3</td>
<td>rd</td>
<td>opcode</td>
</tr>
</tbody>
</table>

\textbf{(label: 3B) Instruction encoding \texttt{(0x)}: } ________________________________

For the following code snippets, provide the value left in each register \textbf{just prior to executing the “nop” operation} (i.e., when the processor reaches the instruction at the end label), or specify \textbf{CAN’T TELL} if it is impossible to tell the value of a particular register. The code snippets are independent of both each other and of the first two parts of this question. Assume that the \texttt{nop} operation is an operation that does nothing.

(C) (6 points)

\begin{verbatim}
addi x1, x0, 0xC55
slli x1, x1, 8
li x2, 0x1
blt x2, x1, L1
addi x2, x2, 1
L1:
  addi x2, x2, 1
end:
  nop
\end{verbatim}

\textbf{(label: 3C_1) x1: } (0x)_______________________

\textbf{(label: 3C_2) x2: } (0x)_______________________

\textbf{(label: 3C_3) pc: } (0x)_______________________
(D) (6 points)

\[
\begin{align*}
&. = 0x100 \\
&\text{addi } x3, \text{ zero, } 0x304 \\
&\text{lw } x4, \ -4(x3) \\
&\text{slli } x4, \ x4, \ 4 \\
&\text{sw } x4, \ 4(x3) \\
\end{align*}
\]

end:

\[
\begin{align*}
&\text{nop} \\
&. = 0x300 \\
&.\text{word } 0x12345678 \\
&.\text{word } 0x60046004 \\
&.\text{word } 0xAAAAAAAA
\end{align*}
\]

(label: 3D_1) Which address in memory is written to: (0x)_______________________

(label: 3D_2) What value is written to memory: (0x)_______________________

(label: 3D_3) pc: (0x)_______________________
Problem 4. Crops and Calling Conventions (19 points)

You are writing a function in RISC-V assembly called do_harvest that will be used as part of a special new RISC-V processor for farmers. The do_harvest function, which is intended to compute the expected profit from harvesting wheat on a certain area of land, calls the function wheat_yield, which takes in the land area over which to pick wheat and returns the expected yield of wheat in units of weight.

```
#define GRAIN_PRICE 2
#define LABOR_COST 8
unsigned do_harvest(unsigned acres) {
    unsigned yield = wheat_yield(acres);
    unsigned income = yield * GRAIN_PRICE;
    unsigned expenses = acres * LABOR_COST;
    return income - expenses;
}
```

(A) (6 points) An implementation of the desired behavior of do_harvest is shown below in C syntax.

```
do_harvest:
    ...
    call wheat_yield
    ...
    ret
```

Implement the do_harvest function in RISC-V assembly such that its behavior matches that of the C implementation, while adhering to the calling convention. Assume that wheat_yield also follows the calling convention. You can use any mapping of variables to registers as you wish, but you will receive more credit for a shorter, correct solution. Assume that the processor is a 32-bit RISC-V so all the registers are 32-bits in length, and all the values involved fit within 32 bits (so do NOT handle over/under-flow). We have intentionally provided the entire next page for your do_harvest implementation to give everyone plenty of space to write their function.
(B) Below is one possible implementation of \texttt{wheat\_yield} (although there are many other ways this could be written) shown in C syntax. The function \texttt{pick\_wheat} returns the wheat in units of weight yielded from a single acre of land, and it follows the calling convention.

```c
unsigned wheat_yield(unsigned acres) {
    unsigned yield = 0;
    for (unsigned j = 0; j < acres; j++) {
        yield += pick_wheat();
    }
    return yield;
}
```

This is the RISC-V assembly implementation that your friend wrote for \texttt{wheat\_yield}. It computes the correct yield and follows the calling convention, but you noticed that its execution is slow due to excessive memory accesses and is throttling the new farming processor, so you offered to take a look at it.

```
wheat_yield:
    addi sp, sp, -16
    sw ra, 0(sp)
    sw s1, 4(sp)
    mv a1, a0
    li s1, 0
    li t1, 0

wy_for:
    sw a1, 8(sp)
    sw t1, 12(sp)
    call pick_wheat
    add s1, s1, a0
    lw t1, 12(sp)
    lw a1, 8(sp)

    addi t1, t1, 1
    bltu t1, a1, wy_for

wy_forend:
    mv a0, s1
    lw s1, 4(sp)
    lw ra, 0(sp)
    addi sp, sp, 16
    ret
```
1. (3 points) During the for loop between the labels `wy_for` and `wy_forend` in the code above, what registers in the assembly implementation do the variables in the C implementation correspond to?

   (label: 4B_1_a) acres := Reg[ ]
   (label: 4B_1_b) yield := Reg[ ]
   (label: 4B_1_c) j := Reg[ ]

2. (4 points) If you could rewrite this code using a different set of registers, what selection of registers would result in the fewest total pushes and pops from the stack during execution of this code, while still following the calling convention (assuming `acres` is a large number, on the order of 100 or more)? Briefly explain your choices.

   (label: 4B_2_a) acres := Reg[ ]
   (label: 4B_2_b) yield := Reg[ ]
   (label: 4B_2_c) j := Reg[ ]
   (label: 4B_2_d) Explanation: ___________________________________________________________

3. (6 points) Using your new mapping in the skeleton below, show all pushes, pops, and updates to the stack pointer required in order to correctly implement `wheat_yield` while following the calling convention.

   (label: 4B_3) wheat_yield:

   _______________________________________
   _______________________________________
   _______________________________________
   _______________________________________
   _______________________________________
mv _______, a0
li _____, 0
li _____, 0

wy_for:
_____________________________
_____________________________
_____________________________
call pick_wheat
add _____, _____, a0
_____________________________
_____________________________
_____________________________
addi _____, _____, 1
bltu _____, _____, wy_for

wy_forend:

mv a0, _____
_____________________________
_____________________________
_____________________________
_____________________________
_____________________________

ret
Problem 5. Stack Detective (16 points)

A mad scientist created a special rabbit (which he named “RAD-BIT”) that can reproduce without a mate and live forever! Every RAD-BIT gives birth to a new RAD-BIT every day.

The software engineer that works for him wrote the `radbit` function in C below to calculate how many RAD-BITs exist after x days from creation of the first RAD-BIT.

```c
/* recursively count the number radbit after x days */
int radbit(unsigned x) {
    if (x == 1) {
        return 2;
    } else {
        return 2 * radbit(x - 1);
    }
}
```

Two RAD-BITs are created. One RAD-BIT (and its offspring) were able to reproduce for n*2 days, and the other RAD-BIT (and its offspring) were able to reproduce for n*4 days.

The following function calculates the total number of RAD-BITs.

```c
int math(unsigned n) {
    unsigned math_x = radbit(n * 2);
    unsigned math_y = radbit(n * 4);
    return math_x + math_y;
}
```

```c
math: addi sp, sp -12
    sw ra, 0(sp)
    sw a0, 4(sp)
    sw s0, 8(sp)

first:
    call radbit
    mv s0, a0
    lw a0, 4(sp)
    slli a0, a0, 2

second:
    call radbit
    add a0, a0, s0
    lw ra, 0(sp)
    lw s0, 8(sp)
    addi sp, sp, 12
    jr ra
```
The engineer is paid by the hour, so they wrote the programs in assembly as well to kill time (shown above on the right).

(A) (2 points) Fill in the missing line in the assembly code for the math function to match the C implementation.

(label: 5A) Missing assembly line: _________________

The mad scientist calls the math function. The program is halted during the second (i.e., after the call radbit at label second) call to radbit, right before executing the line labeled stop.

The stack pointer was located at SP_start when the math assembly program first started running. At the time the program halted, it was located at SP_stop.

The radbit and the math functions share the same stack.

| SP_stop → | 0x10 |
|           | 0xF0 |
|           | (label: 5B_1) |
|           | (label: 5B_2) |
|           | (label: 5B_3) |
|           | 0x40 |
|           | 0x08 |
|           | 0x02 |
|           | 0x01 |

| SP_start → | 0x14 |
| Larger addresses | 0x10 |

(B) (3 points) Fill out the three missing spaces in the stack.

(C) (1 point) What is the value of unsigned n passed into the original math function call?

(label: 5C) Value of n passed into original math function call: _________________
(D) (2 points) What is the value inside unsigned math_x after the first call to radbit returns?

(label: 5D) Value of math_x after the first call to radbit (0x): ____________________

(E) (2 points) What is the address of the original math function call?

(label: 5E) Address of original math function call (0x): ____________________

(F) (2 points) What is the address of the label first in the math function?

(label: 5F) Address of label first in math function (0x): ____________________

(G) (2 points) What is the address of the label stop in the radbit function?

(label: 5G) Address of label stop in radbit function (0x): ____________________

(H) (2 points) The mad scientist noticed that one of the registers the engineer saved into the stack doesn’t need to be saved. Which register is this and why?

(label: 5H_1) Unnecessarily saved register: ____________________

(label: 5H_2) Why doesn’t this register need to be saved on the stack? ____________________
Problem 6. Static Discipline (14 points)

Ben Bitdiddle and Alyssa P. Hacker are given Device R which has the following VTC that is dependent on the supply voltage of the device, $V_{DD}$. All numbers on the curve are a factor of $V_{DD}$ (e.g. 0.3 means $0.3 \times V_{DD}$). For the entirety of this problem, assume that all circuits only contain Device R.

(A) (1 point) Assuming the proper signaling specifications are applied to obey the static discipline, what logic gate(s) can a single instance of Device R implement?

(label: 6A) Device Type (circle one): Buffer Inverter Both Neither

Alyssa and Ben begin designing their circuits. To start, they decide on the following signaling specification.

\[
\begin{align*}
V_{OL} &= 0.1 \times V_{DD} \\
V_{OH} &= 0.8 \times V_{DD} \\
V_{IL} &= 0.3 \times V_{DD} \\
V_{IH} &= 0.7 \times V_{DD}
\end{align*}
\]

(B) (4 points) Alyssa decides to use a supply voltage of 10V (i.e. $V_{DD,A} = 10V$) and so designs her circuit with the following signaling specification

\[
\begin{align*}
V_{OL,A} &= 1V \\
V_{OH,A} &= 8V \\
V_{IL,A} &= 3V \\
V_{IH,A} &= 7V
\end{align*}
\]

Alyssa’s circuit must provide input to and accept output from Ben’s circuit, but Ben designed his circuit independently and used a different supply voltage. What is the full range of supply voltages ($V_{DD,B}$) that Ben could have used such that their circuits will work together correctly in the following configuration? It is recommended that you leave your answers in fractional form.
(label: 6B) **Bounds on Ben’s Supply Voltage:** \[ \_\_\_\_\_\_\_\_\_ V \leq V_{DD,B} \leq \_\_\_\_\_\_\_\_\_ V \]

(C) (3 points) Compute the high and low noise margins and the noise immunity for Alyssa’s circuit with the current signaling specification and \( V_{DD,A} = 10V \). Noise immunity is defined as the minimum of the high and low noise margins.

(label: 6C_1) **Low Noise Margin:** \[ \_\_\_\_\_\_\_\_\_ V \]

(label: 6C_2) **High Noise Margin:** \[ \_\_\_\_\_\_\_\_\_ V \]

(label: 6C_3) **Noise Immunity:** \[ \_\_\_\_\_\_\_\_\_ V \]

(D) (4 points) Alyssa and Ben’s friend, Chip D. Signer, examines their circuits and realizes that their choice of signaling specification does not maximize their noise immunity. What changes should be made to each parameter of the signaling specification to maximize noise immunity? *Pay close attention to which of the four thresholds you are providing an answer for to avoid unwanted mistakes.*

(label: 6D_1) **\( V_{OL} \) (Circle one):**
- INCREASE
- DECREASE
- NO CHANGE

(label: 6D_2) **\( V_{OH} \) (Circle one):**
- INCREASE
- DECREASE
- NO CHANGE

(label: 6D_3) **\( V_{IL} \) (Circle one):**
- INCREASE
- DECREASE
- NO CHANGE

(label: 6D_4) **\( V_{IH} \) (Circle one):**
- INCREASE
- DECREASE
- NO CHANGE

(E) (2 points) Briefly explain your choices in part D.

(label: 6E)
Problem 7. Boolean Algebra (11 points)

(A) (5 points) Simplify the following equation into its minimum sum of products form.

\[ ec(abc + \overline{a} + \overline{b} + c + \overline{d}bc + \overline{b}) \]

(label: 7A) Minimal sum of products expression: __________________________

(B) (3 points) If a gate G implements the function \( G(a, b, c, d) = ab\overline{c} + \overline{b}d \). Is that gate universal? Explain why or why not.

(label: 7B)
(C) (3 points) Show how to build a 2-input OR gate out of 2-input NAND gates.

(label: 7C)
Problem 8. CMOS (12 points)

(A) (2 points) One day at work, Bob comes running up to your desk with a seemingly random truth table, Table 1 (below). He exclaims, "This circuit will change the world!" He also claims that this circuit can be constructed using a single CMOS gate. Upon closer examination, you realize that you cannot implement the function $Q(A, B, C)$ with a single CMOS gate. Below, please explain why you cannot implement $Q$ with a single gate. In your explanation, please note specific rows that would make this impossible.

Table 1:

<table>
<thead>
<tr>
<th>Row</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(label: 8A) Explanation:
(B) (4 Points) After pointing out the mistakes, Bob rushes out quickly and returns with a new table, Table 2 (below). You see that all he did was invert all of the outputs. Unsure of his CMOS skills, Bob asks you to try to implement this new function \( R \). In the space below, please draw a CMOS implementation of \( R(A, B, C) \). **For full credit minimize the total number of transistors used in your implementation.**

Table 2:

<table>
<thead>
<tr>
<th>Row</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(label: 8B) CMOS implementation of \( R(A,B,C) \):
(C) (6 points) With your CMOS implementation, Bob proposes making a new module, \( R \) using your CMOS gate. He asserts that this new module is universal! Not wanting to make any mistakes, you set out to prove this. Using your implementation for \( R \), please show how you would build a two-input NAND gate, a two-input NOR gate, and a two-input AND gate. Please write your answers using function composition (e.g. \( R(X, Y, R(0, 0, 0)) \)).

\[
\text{(label: 8C_1) NAND(X, Y) = ____________________________}
\]

\[
\text{(label: 8C_2) NOR(X, Y) = ____________________________}
\]

\[
\text{(label: 8C_3) AND(X, Y) = ____________________________}
\]

END OF QUIZ 1!