CS 61C Fall 2023

Garcia, Yokota Midterm

Print your name: ____

Print your student ID: ______________________

You have 110 minutes. There are 7 questions of varying credit (100 points total).

Question:	1	2	3	4	5	6	7	Total
Points:	13	20	14	19	15	18	1	100

For questions with **circular bubbles**, you may select only one choice.

- ◯ Unselected option (completely unfilled)
- ⬤ Only one selected option (completely filled)
- ☑ Don't do this (it will be graded as incorrect)

For questions with **square checkboxes**, you may select one or more choices.

- ☐ You can select
- ■ multiple squares
- ■ (completely filled)

Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation. For coding questions with blanks, you may write at most one statement per blank and you may not use more blanks than provided.

If an answer requires hex input, you must only use capitalized letters (`0xDEADBEEF` instead of `0xdeadbeef`). For hex and binary, please include prefixes in your answers unless otherwise specified, and do not truncate any leading 0's. For all other bases, do not add any prefixes or suffixes.

Write the statement below in the same handwriting you will use on the rest of the exam.

I have neither given nor received help on this exam (or quiz), and have rejected any attempt to cheat; if these answers are not my own work, I may be deducted up to 0x0123 4567 89AB CDEF points.

____

____

____

____

____

Sign your name: ______________________

This page intentionally left (mostly) blank.

The exam begins on the next page.

Q1 🍯🪴🍲🫗 *(Potpourri)* **(13 points)**

For Q1.1-Q1.3, convert the 8-bit binary `0b0100 1011` to decimal, treating it as...

Q1.1 (1 point) ...an **unsigned** integer.

Q1.2 (1 point) ...a **two's complement** integer.

Q1.3 (1 point) ...a **sign-magnitude** integer.

Q1.4 (2.5 points) What is the decimal value of `z` in the snippet of C code below?

```
int8_t x = 101;
int8_t y = 77;
int8_t z = x + y;
```

Q1.5 (3 points) Translate the following RISC-V instruction to hexadecimal: `sw s0 10(s2)`

0x

For Q1.6-Q1.8, indicate the stage of CALL that...

Q1.6 (1.5 points) ...converts pseudoinstructions into equivalent instructions.

○ Compiler ○ Assembler ○ Linker ○ Loader

Q1.7 (1.5 points) ...determines if functions are called with valid variable types.

○ Compiler ○ Assembler ○ Linker ○ Loader

Q1.8 (1.5 points) ...jumps execution to the start of the program.

○ Compiler ○ Assembler ○ Linker ○ Loader

Q2 🍕🍉🍰🥧 *(C)* **(20 points)**

Recall in lab, we implemented some functions for a `vector_t` struct. In this question, we will add support for slicing a `vector_t`.

To do so, we've made some updates to the `vector_t` type from lab. You may assume that all necessary standard libraries are included.

```
typedef struct vector_t {
    // Number of elements in the vector; you may assume size > 0
    size_t size;

    // Pointer to the start of the vector
    int* data;

    // Number of child slices
    size_t num_slices;

    // Array of the vector's child slices, or NULL if num_slices == 0
    struct vector_t** slices;

    // true if the vector is a child slice of another vector, otherwise false
    bool is_slice;
} vector_t;
```

Useful C function prototypes:

```
void* malloc(size_t size);
void free(void *ptr);
void* calloc(size_t num_elements, size_t size);
void* realloc(void *ptr, size_t size);

size_t strlen(char* s);
char* strcpy(char* dest, char* src);
```

Implement the function `vector_slice`, which should return a slice of a `vector_t` at the given indices, with the following signature:

- `vector_t* v`: A pointer to the parent vector to create the slice from.
- `int start_index`: The beginning index of the new slice's data (inclusive)
- `int end_index`: The ending index of the new slice's data (exclusive). You may assume that `end_index > start_index`.
- Return value: A `vector_t*` representing data as described by `start_index` and `end_index`. A parent `vector_t` shares (portions of) its `data` array with all of its descendant slices.

For example:

```
// vec_a has the elements [0, 1, 2, 3, 4]
vector_t* vec_a = /* omitted */;

// vec_b should be of size 2 and have the elements [1, 2]
vector_t* vec_b = vector_slice(vec_a, 1, 3);

vec_b->data[1] = 10;
// At this point, vec_a should be [0, 1, 10, 3, 4]
//               vec_b should be [1, 10]
```

```
vector_t* vector_slice(vector_t* v, int start_index, int end_index) {

    vector_t* slice = ______________________________________;
                                   Q2.1

    if (slice == NULL) { allocation_failed(); }

    slice->_____________________________________________;
                              Q2.2

    slice->_____________________________________________;
                              Q2.3

    slice->_____________________________________________;
                              Q2.4

    v->slices = ________________________________________;
                                Q2.5

    if (v->slices == NULL) { allocation_failed(); }

    v->slices[_____________________] = __________________;
                      Q2.6                    Q2.7

    v->_________________________________________________;
                              Q2.8
    return slice;
}
```

The **vector_t** struct definition is repeated below for your convenience:

```
typedef struct vector_t {
    // Number of elements in the vector; you may assume size > 0
    size_t size;

    // Pointer to the start of the vector
    int* data;

    // Number of child slices
    size_t num_slices;

    // Array of the vector's child slices, or NULL if num_slices == 0
    struct vector_t** slices;

    // true if the vector is a child slice of another vector, otherwise false
    bool is_slice;
} vector_t;
```

To accommodate these changes to the **vector_t** type, we need to update the **vector_delete** function to properly free our new data structure. When a **vector_t** is deleted, all descendant slices of the **vector_t** should also be freed. You may assume that **vector_delete** is only called on a **vector_t** that is not a slice.

```
void vector_delete(vector_t* v) {

    if (________________________________________) {
                        Q2.9

        ________________________________________;
                        Q2.10
    }
    for (int i = 0; i < v->num_slices; i++) {

        ________________________________________;
                        Q2.11
    }
    if (v->num_slices > 0) {

        ________________________________________;
                        Q2.12
    }

    ____________________________________________;
                        Q2.13
}
```

Q3 👉🎈🎈👈 *(Floating Point)* **(14 points)**

For the entirety of this question, assume that we are using an IEEE 754 standard floating-point representation with 16 bits: 1 sign bit, 7 exponent bits (with a standard bias of -63), and 8 significand bits.

For questions Q3.1 and Q3.2, convert the value to hexadecimal, using the floating-point representation above.

Q3.1 (2 points) -2.25

0x

Q3.2 (2 points) 1.75×2^{-63}

0x

For questions Q3.3 and Q3.4, what is the value of the floating-point number? Express your answer as $x \times 2^y$, where x is a decimal such that $1 \leq |x| < 2$ and y is an integer. For NaN, write "NaN"; for positive infinity, write "$+\infty$"; for negative infinity, write "$-\infty$".

Q3.3 (2 points) `0x7F9A`

Q3.4 (2 points) `0x7000`

Q3.5 (3 points) List all numbers k in this floating-point representation such that the smallest floating-point number strictly greater than k is exactly $2k$. Express your answer as $x \times 2^y$, where x is a decimal such that $1 \leq |x| < 2$ and y is an integer. If there are no such numbers, write "None".

Q3.6 (3 points) Express, in hexadecimal, the smallest strictly positive representable number k such that $k + 1$ is also representable in this floating-point representation.

0x

Q4 🚀🛸🚁🛩️ *(RISC-V)* **(19 points)**

The Jetidiah Propulsion Laboratory (JPL) is planning to race some vehicles around Jezero crater, including PersevRyance, CuriosEddy, and Bengenuity. They would like to write some RISC-V code to determine the best vehicle, and they need your help!

The **`race`** function, which has been correctly implemented for you, is defined as follows:

- Input in `a0`: **`id`**, the ID of a vehicle. You may assume that $0 \leq \texttt{id} < 200$.
- Output in `a0`: The distance traveled (in nanometers) by the vehicle in a sprint as an unsigned integer. You may assume that this value is less than 2^{32}.

Implement **`best_vehicle`** as follows, using the **`race`** function:

- Input in `a0`: **`num_vehicles`**, the total number of vehicles being raced. The function should race vehicles with IDs between 0 and **`num_vehicles - 1`** (inclusive).
  You may assume that $0 < \texttt{num\_vehicles} \leq 200$.
- Output in `a0`: The vehicle ID that traveled the farthest.
- **`best_vehicle`** should race the vehicles and return the ID of the vehicle that traveled the farthest. In case of a tie between multiple vehicles, return the lowest ID among those vehicles.

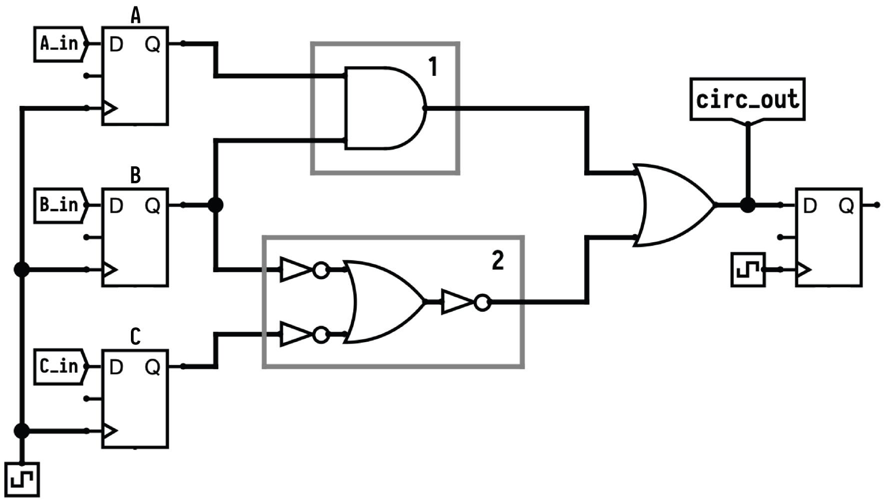
For example, given the following return values from the **`race`** function, **`best_vehicle(5)`** should race 5 vehicles (IDs 0, 1, 2, 3, and 4) and return 2, and **`best_vehicle(2)`** should race 2 vehicles (IDs 0 and 1) and return 0.

ID	`race(ID)`
0	`600,000,000`
1	`100,000`
2	`3,000,000,000`
3	`1,564`
4	`3,000,000,000`

```
best_vehicle:
    ______________________________
                Q4.1
    sw ___________________________
                Q4.2
    sw ___________________________
                Q4.3
    sw ___________________________
                Q4.4
    sw ___________________________
                Q4.5
    mv s0 a0
    addi s1 x0 0 # The best distance so far
    addi s2 x0 0 # The index of the best vehicle so far
    addi t0 x0 0 # The index of the vehicle being tested
Loop:
    ______________________________
                Q4.6
    __________________________ 0(sp)
                Q4.7
    ___________________________ race
                Q4.8
    __________________________ 0(sp)
                Q4.9
    ______________________ Continue
                Q4.10
    mv s1 a0
    mv s2 t0
Continue:
    ______________________________
                Q4.11
    ___________________________ Loop
                Q4.12
    ______________________________
                Q4.13
    # Epilogue Omitted
    # You may assume that the epilogue has been correctly
    #     implemented based on your prologue.
    ...
    jr ra
```

Q5 (*SDS*) **(15 points)**

Consider the following circuit and timings:



$t_{AND} = 10\text{ns}$

$t_{OR} = 15\text{ns}$

$t_{NOT} = 5\text{ns}$

$t_{clk\text{-}q} = 3\text{ns}$

$t_{setup} = 5\text{ns}$

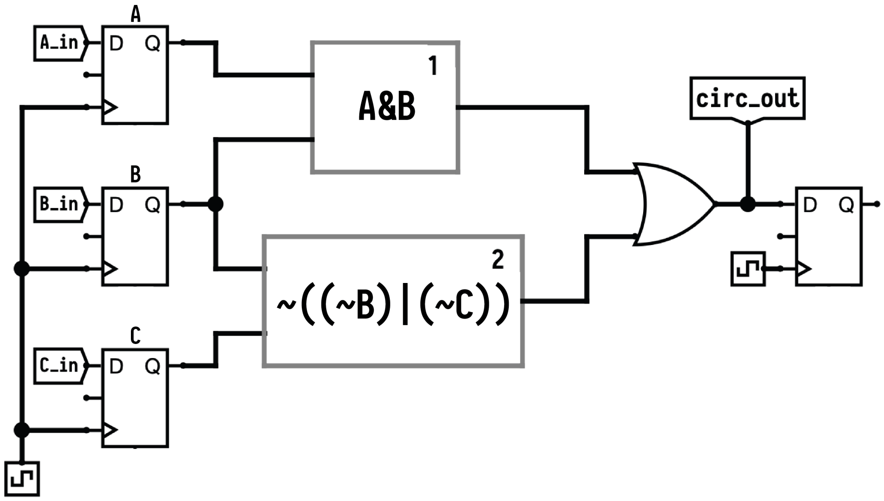
Q5.1 (2.5 points) What is the **minimum clock period**, in nanoseconds, for the circuit above such that it will always exhibit well-defined behavior?

ns

Q5.2 (2.5 points) What is the **maximum hold time**, in nanoseconds, for the circuit above such that it will always exhibit well-defined behavior?

ns

For the rest of this question, consider the updated circuit diagram and timings below. Subcircuits 1 and 2 are black-boxes that compute the Boolean expressions shown in the diagram. Regardless of your previous answers, assume the clock period is 80ns, and for subcircuits 1 and 2, their respective minimum and maximum combinational logic delays are equal. For example, if the input to subcircuit 1 changed at $t = 0$ (and no other changes occur), then the output to subcircuit 1 will change at $t = 13$ only.



$t_{\text{OR}} = 15\text{ns}$
$t_{\text{subcircuit 1}} = 13\text{ns}$
$t_{\text{subcircuit 2}} = 24\text{ns}$
$t_{\text{clk-q}} = 3\text{ns}$
$t_{\text{setup}} = 5\text{ns}$

Until this circuit settles at the correct input, it calculates two other Boolean expressions at the tunnel `circ_out`. Assume that register outputs are all 0 at $t = 0$, and that there are no setup or hold time violations.

Q5.3 (2 points) Let t_1 be the first time that `circ_out` changes. What is t_1 in nanoseconds? Clarified during exam: t_1 is the first time that `circ_out` *may change.*

ns

Q5.4 (2 points) Let t_2 be the second time that `circ_out` changes. What is t_2 in nanoseconds?

ns

Q5.5 (2 points) What simplified Boolean expression does the circuit calculate while $0\text{ns} \leq t < t_1$?

Q5.6 (2 points) What simplified Boolean expression does the circuit calculate while $t_1 \leq t < t_2$?

Q5.7 (2 points) What simplified Boolean expression does the circuit calculate while $t_2 \leq t < 80\text{ns}$?

Q6 😈🐶🐶🐶 *(CS61Cerberus)* **(18 points)**

Heracles has been tasked with yet another labour: taming the CS61Cerberus, the three-headed guard dog of Hades. Each head of CS61Cerberus has a favorite number (independent of the other heads), which is represented as an n-bit unsigned integer. In order to tame him, Heracles must determine the favorite numbers of all heads, by saying numbers and seeing how the heads respond.

Heracles can say any n-bit unsigned integer to any head, and the head will respond either **`true`** or **`false`** according to its rule and favorite number:

- The left head (Orion) receives its input and performs a bitwise OR on it with its favorite number. If the result is 0, Orion returns **`true`**. Otherwise, Orion returns **`false`**.
- The middle head (Andy) receives its input and performs a bitwise AND on it with its favorite number. If the result is 0, Andy returns **`true`**. Otherwise, Andy returns **`false`**.
- The right head (Luxor) receives its input and performs a bitwise XOR on it with its favorite number. If the result is 0, Luxor returns **`true`**. Otherwise, Luxor returns **`false`**.

Q6.1 Write a function **`andy`**, which receives an unsigned integer as input and returns a **`bool`** according to how Andy would react to that integer, assuming $n = 32$.

```
bool andy(uint32_t input) {
    // Andy's favorite number has been omitted.
    uint32_t andnum = /* omitted */;

    return ____________________________________________;
}
```

Heracles writes three algorithms, **`solve_orion`**, **`solve_andy`**, and **`solve_luxor`** to find and return the favorite number of each head. Each of the three solver functions accepts one argument: a function that accepts a number and returns the response from the corresponding head.

Write **`solve_orion`**, **`solve_andy`**, and **`solve_luxor`** (for simplicity, your code only needs to work when $n = 32$). Your solution must be asymptotically optimal for full credit.

Additionally, write the worst-case asymptotic time complexity of the optimal solver function in Θ notation with respect to n, the number of bits that the head accepts. The Θ bound must hold for all values of n, not just when $n = 32$.

If no such algorithm exists, write "Impossible" in the code block, and leave the Θ bound blank.

Q6.2
```
uint32_t solve_orion(bool(*orion)(uint32_t)) {
    // Your code here or Impossible
}
```

$\Theta($ $)$

Q6.3
```
uint32_t solve_andy(bool(*andy)(uint32_t)) {
    // Your code here or Impossible
}
```

$\Theta($ $)$

Q6.4
```
uint32_t solve_luxor(bool(*luxor)(uint32_t)) {
    // Your code here or Impossible
}
```

$\Theta($ $)$

Q7 🏁🔚🏁🔚 *(The Finish Line)* **(1 points)**

Everyone will receive credit for this question, even if you leave it blank.

Q7.1 (1 point) CS 61A defeated the Lambdanean Hydra in Fall 2022, CS 61B cleaned the Javaugean Stables in Spring 2023, and you've just tamed(?) CS61Cerberus. What are the 9 other labours of Heracles? Express your answer in emoji drawings.

Q7.2 (0 points) Is there anything you want us to know?