CS 61C Summer 2023

Charles, Jero, Rosalie Midterm

PRINT your name: _____, _____
(last) (first)

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:	15	24	15	25	12	8	1	100

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

- ○ Unselected option (completely unfilled)
- ● Only one selected option (completely filled)

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

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

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

If an answer requires hex input, make sure you only use capitalized letters! For example, `0xDEADBEEF` instead of `0xdeadbeef`. Please include hex (`0x`) or binary (`0b`) prefixes in your answers unless otherwise specified. For all other bases, do not add any prefixes or suffixes.

Read the following honor code and sign your name.

I understand that I may not collaborate with anyone else on this exam, or cheat in any way. I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct and may further result in, at minimum, negative points on the exam.

SIGN your name: _____

Q1 *Potpourri* **(15 points)**

Q1.1 (1 point) An n-bit number in bias notation and an n-bit two's complement number always represent the same range of numbers.

○ True ○ False

Q1.2 (1 point) An n-bit two's complement number can represent exactly twice the number of unique values as an n-bit unsigned number.

○ True ○ False

Q1.3 (1 point) For any number representation system, you must have at least $2n + 1$ bits to represent the value 2^{2n}.

○ True ○ False

Q1.4 (1.5 points) Convert `0xA7` to decimal, assuming the data was stored as a two's complement one-byte integer.

Q1.5 (1 point) The output of the compiler never contains pseudoinstructions.

○ True ○ False

Q1.6 (1 point) The loader produces an executable.

○ True ○ False

For Q1.7 and Q1.8, consider the following code snippet and assume that `int`s are 4 bytes.

```
int x = 257;
int y = strlen((char *) &x);
```

Q1.7 (1.5 points) In a little-endian system, what will `y` contain?

Q1.8 (1.5 points) In a big-endian system, what will `y` contain?

Q1.9 (2 points) Write a Boolean expression that simplifies the Boolean expression below. You may use at most 3 Boolean operations. $\sim$ (NOT), | (OR), & (AND) each count as one operation. We will assume standard C operator precedence, so use parentheses when uncertain.

```
~(A | ~C) | ((~A & B) | B)
```

Q1.10 (2 points) Translate the following RISC-V instruction to its hexadecimal counterpart.

```
slli a0 t2 4
```

0x

Q1.11 (1.5 points) Represent 1.5×2^{-511} in hex using a binary floating point representation, which follows IEEE-754 standard conventions, but has 10 exponent bits (and a standard bias of -511) and 21 mantissa bits.

0x

Q2 *#include "library.c"* **(24 points)**

```
 1 #define MAX_BORROWS 25
 2
 3 typedef struct {
 4   char* book_name;
 5   bool borrowed;
 6 } Book;
 7
 8 typedef struct {
 9   char* user_id;
10   Book* borrowed_books[MAX_BORROWS];
11 } User;
12
13 typedef struct {
14   User* users;
15   int users_len;
16   Book* books;
17   int books_len;
18 } Library;
```

The city of Eddy B.C wants to build a new library! The `init_users` function receives the following input:

- `Library* lib`: A pointer to an uninitialized `Library` struct. You may assume that memory has already been correctly allocated on the heap for the `Library` struct.
- `char** user_ids`: An array of well-formatted strings of nonzero length except the last element. The last element is `NULL`. You may assume that all strings are allocated on the stack.

The function should make sure the following properties are held:

- `users_len` should be set to the number of strings in `user_ids`.
- Each `User` in `users` should be initialized as follows:
  - The `user_id` of the `i`th `User` in `users` should be set to the `i`th string in `user_ids`.
  - `borrowed_books` should be an array of `NULL`s to indicate that no `Book` has been borrowed.
- Every `User` and its contents must persist through function calls.

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);

// memset sets the first num bytes of the block of memory pointed to by ptr
// to the specified value (interpreted as an unsigned char).
void* memset(void* ptr, int value, size_t num);
```

(15 points) Fill in `init_users` so that it matches the described behavior. Assume that all necessary C libraries are included.

```
 1 void init_users(Library* lib, char** user_ids) {
 2   int i = 0;

 3   while (_____Q2.1_____) {

 4     lib_____Q2.2_____ = _____Q2.3_____;

 5     User* cur_user = _____Q2.4_____;

 6     cur_user_____Q2.5_____ = _____Q2.6_____;

 7     strcpy(cur_user_____Q2.7_____, _____Q2.8_____);

 8     memset(cur_user_____Q2.9_____, _____Q2.10_____,
                    MAX_BORROWS * _____Q2.11_____);
 9     i++;
10   }

11   lib_____Q2.12_____ = i - 1;
12 }
```

The `remove_users` function takes in one argument:

- `Library* lib`: A pointer to a `Library` struct where the contents have been allocated on the heap.

The `remove_users` function should free every `User` and any additional memory that might have been allocated for the `User`.

```
1 void remove_users(Library* lib) {
2   for (int i = 0; i < lib->users_len; i++) {
3     free(lib->users[i]);
4   }
5 }
```

Q2.13 (3 points) Is the `remove_users` function implemented correctly? If "Yes", no justification is required. If "No", please explain in two sentences or fewer.

○ Yes ○ No

For each of the following symbols, choose which section of memory it would live in.

Q2.14 (1 point) `MAX_BORROWS`

○ Stack ○ Heap ○ Data/Static ○ Code

Q2.15 (1 point) `init_users`

○ Stack ○ Heap ○ Data/Static ○ Code

Q2.16 (1 point) `lib`, the parameter for `init_users`

○ Stack ○ Heap ○ Data/Static ○ Code

Q3 *0x5f3759df* **(15 points)**

Thanton remembered from class that right-shifting a binary number is the same as dividing it by two (rounding down). They did get a little too eager, though, and have tried applying the concept to arbitrary numbers including floating point numbers! They wrote a function, `mystery`, to logically right shift the binary representation of the input value by 1 and return the result as a floating point value.

For this question, assume that we're working with standard IEEE-754 single-precision floating point numbers.

For Q3.1 to Q3.4, what would each of the following function calls return?

Q3.1 (2 points) `mystery(0)`

Q3.2 (2 points) `mystery(-0)`

Q3.3 (2 points) `mystery(`∞`)`

Q3.4 (2 points) `mystery(`$-\infty$`)`

For Q3.5 and Q3.6, "normal numbers" are numbers that are not denormalized, zero, NaN, or infinity.

Q3.5 (3.5 points) Suppose we run `mystery` on every possible NaN value. What are all of the possible return values? Select all that apply.

- ☐ Denormalized numbers
- ☐ NaN
- ☐ Normal numbers
- ☐ Zero
- ☐ Infinity
- ☐ None of the above

Q3.6 (3.5 points) Suppose we run `mystery` on every possible denormalized value. What are all of the possible return values? Select all that apply.

- ☐ Denormalized numbers
- ☐ NaN
- ☐ Normal numbers
- ☐ Zero
- ☐ Infinity
- ☐ None of the above

Q4 *RISC-V, RISC-XVI, RISC-VIII* **(25 points)**

A *wondrous* sequence of positive integers is defined as follows: If n is even, then the next number is $\frac{n}{2}$. Otherwise, the next number is $3n + 1$.

Q4.1 (2 points) We want to create a pseudoinstruction to check whether a number is odd or not. This instruction, written `is_odd rd rs1`, will put the value `1` in `rd` if the value in `rs1` is odd, and the value `0` otherwise. What is the RISC-V instruction that `is_odd rd rs1` would translate to? You may only use one instruction, and you may not use any pseudoinstructions.

Note: Your solution may include `rd` and `rs1`.

Using `is_odd`, write a function (that follows calling convention) to compute the next wondrous number and return it in `a0`, given the current wondrous number as input in `a0`. Your code may not use `mul` or any `t` registers.

```
1  next_number:
2      _____ Q4.2
3      _____ Q4.3
4      is_odd _____ Q4.4
5      beq s0 x0 else
6      _____ Q4.5
7      _____ Q4.6
8      _____ Q4.7
9      j exit
10 else:
11     _____ Q4.8
12 exit:
13     _____ Q4.9
14     _____ Q4.10
15     jr ra
```

Using `next_number`, write a function (that follows calling convention) to count the number of steps until a wondrous sequence reaches the number `1` and return the number of steps in `a0`, given a starting number as input in `a0`.

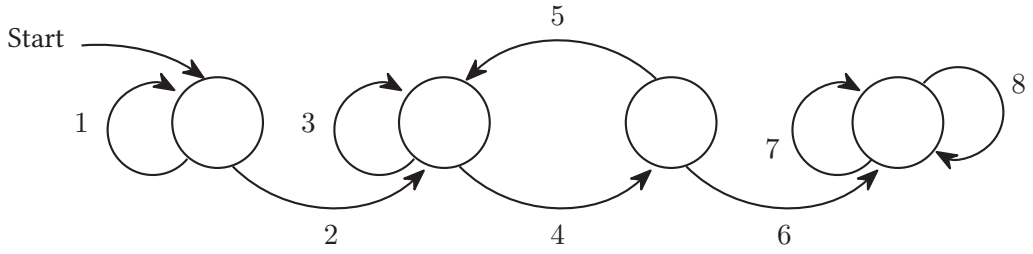
You may assume that `a0` will be a positive integer. You may not use any additional `s` registers beyond those provided in the skeleton code.

```
1 num_steps:
      # Prologue
      # Omitted

2     addi s0 x0 _____________________________
                          Q4.11

3 loop_start:

4     addi t0 x0 _____________________________
                          Q4.12

5     _______________________________________
                        Q4.13

6     _______________________________________
                        Q4.14

7     _______________________________________
                        Q4.15
8     j loop_start

9 loop_end:

10    _______________________________________
                        Q4.16
      # Epilogue
      # Omitted
11    jr ra
```

Q5 *Fully Secure Machine* **(12 points)**

A garage door opens if it ever sees the password 011 in a transmission. More formally, this FSM takes a bitstring consisting of 0's and 1's as its input, and continually outputs 0's until it sees the substring 011, after which it outputs 1's continuously. Example executions of this FSM are below:

Input:	010101001100001010101	Input:	00000000001000000010
Output:	00000000111111111111	Output:	00000000000000000000



For each of the numbered arrows, mark the correct FSM state transition.

Q5.1 (1.5 points) Arrow 1

○ 0/0 ○ 0/1 ○ 1/0 ○ 1/1

Q5.2 (1.5 points) Arrow 2

○ 0/0 ○ 0/1 ○ 1/0 ○ 1/1

Q5.3 (1.5 points) Arrow 3

○ 0/0 ○ 0/1 ○ 1/0 ○ 1/1

Q5.4 (1.5 points) Arrow 4

○ 0/0 ○ 0/1 ○ 1/0 ○ 1/1

Q5.5 (1.5 points) Arrow 5

○ 0/0 ○ 0/1 ○ 1/0 ○ 1/1

Q5.6 (1.5 points) Arrow 6

○ 0/0 ○ 0/1 ○ 1/0 ○ 1/1

Q5.7 (1.5 points) Arrow 7

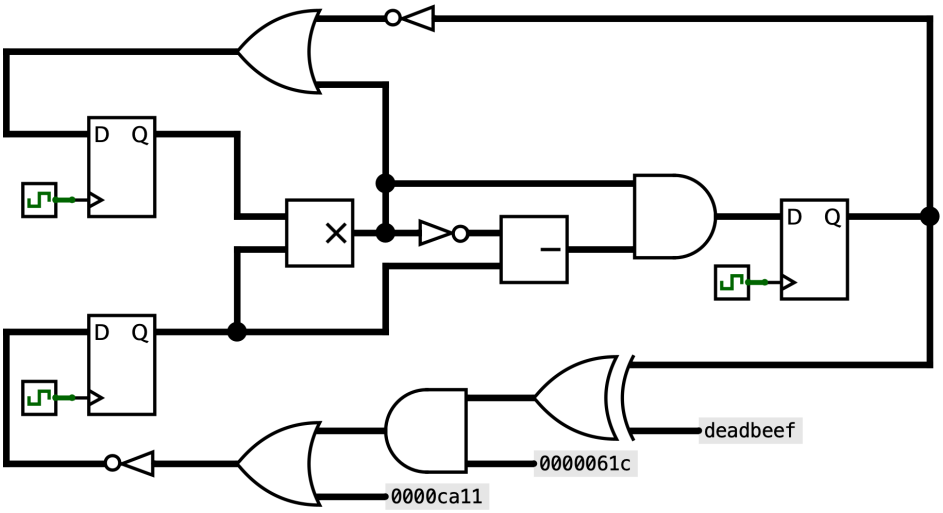
○ 0/0 ○ 0/1 ○ 1/0 ○ 1/1

Q5.8 (1.5 points) Arrow 8

○ 0/0 ○ 0/1 ○ 1/0 ○ 1/1

Q6 *SDS* **(8 points)**

Consider the following circuit diagram and component delays:



$t_{\text{AND}} = 20\text{ps}$
$t_{\text{NOT}} = 5\text{ps}$
$t_{\text{OR}} = 20\text{ps}$
$t_{\text{XOR}} = 25\text{ps}$
$t_{\text{multiplier}} = 50\text{ps}$
$t_{\text{subtract}} = 35\text{ps}$
$t_{\text{clk-q}} = 5\text{ps}$
$t_{\text{setup}} = 2\text{ps}$

Q6.1 (2 points) What is the smallest combinational delay of all paths in this circuit, in picoseconds?

ps

Q6.2 (2 points) What is the minimum allowable clock period for this circuit to function properly, in picoseconds?

ps

Q6.3 (2 points) What is the maximum hold time the registers can have so that there are no hold time violations in the circuit above?

ps

Q6.4 (2 points) Suppose this circuit only deals with two's complement integers. Currently, the subtractor component has a delay of 35ps. What is the maximum delay an adder component can have such that we could replace the subtractor component with adders, NOT gates, and constants to achieve the same delay as the subtractor while maintaining the same behavior? You may assume that constants have no delay.

As a reminder, the subtract component does the following operation:
output = top input - bottom input

ps

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

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

Q7.1 (1 point) Where are the RISC and IEEE-754 plaques in Soda Hall?

- ○ 1st Floor
- ○ 2nd Floor
- ○ 3rd Floor
- ○ 4th Floor
- ○ 5th Floor
- ○ 6th Floor
- ○ 7th Floor

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