CS 61C Spring 2023

Garcia, Yokota Midterm

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

PRINT your student ID: _____

You have 110 minutes. There are 6 questions of varying credit (100 points total).

Question:	1	2	3	4	5	6	Total
Points:	24	24	14	16	10	12	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 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* **(24 points)**

Q1.1 (2 points) For some $n > 0$, n-bit sign-magnitude and n-bit two's complement can represent the same number of values.

○ True ○ False

Q1.2 (2 points) Offsets for jump instructions will always be resolved in the linker step.

○ True ○ False

Q1.3 (2 points) `gcc` executes all four steps of CALL.

○ True ○ False

Q1.4 (2 points) Using dynamic linking may result in a smaller executable size compared to static linking.

○ True ○ False

Q1.5 (2 points) Calling convention dictates that you must save `ra` onto the stack in the prologue.

○ True ○ False

Q1.6 (3 points) Translate the following RISC-V instruction to its hexadecimal counterpart.

```
jal s3 588
```

Hint: 588 = 512 + 64 + 8 + 4

0x

Q1.7 (3 points) Write a Boolean expression that evaluates to the truth table below. You may use at most 2 Boolean operations. $\sim$ (NOT), | (OR), & (AND) each count as one operation. We will assume standard C operator precedence, so use parentheses when uncertain.

W	Y	Z	Out
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

What is the output of this program on a **32-bit, little endian** system?
Reminder: the reference sheet has a list of common C format specifiers.

```
#include <stdint.h>
#include <stdio.h>

int main() {
    int8_t i = 0xA7;

    // %hhd is like %d except it interprets i as an 8-bit integer
    printf("Q1.8: %hhd\n", i);

    // %hhu is like %u except it interprets i as an 8-bit integer
    printf("Q1.9: %hhu\n", i);

    char* str = "hello!!";

    printf("Q1.10: %x\n", ((int8_t *) str)[1]);
    printf("Q1.11: %x\n", ((int32_t *) str)[1]);
    return 0;
}
```

Q1.8 (2 points)

Q1.8:

Q1.9 (2 points)

Q1.9:

Q1.10 (2 points)

Q1.10:

Q1.11 (2 points)

Q1.11:

Q2 *I Can't C My Cheatsheet;* **(24 points)**

```
1  #define NUM_PAGES 8
2
3  typedef struct Page {
4      int num;
5      char* data;
6  } Page;
7
8  typedef struct Cheatsheet {
9      int student_id;
10     int total_length;
11     Page pages[NUM_PAGES];
12 } Cheatsheet;
```

The function `cheatsheet_init` receives as input the following:

- `Cheatsheet** ch`: The location in memory to store the pointer to the created `Cheatsheet`.
- `int student_id`: The student ID of the owner of the `Cheatsheet`.
- `char** contents`: An array of well-formatted strings of nonzero length. You may assume that this array contains exactly `NUM_PAGES` strings, and that all strings are allocated on the **stack**.

It should create a well-formed `Cheatsheet` with the following properties, and save a pointer to that `Cheatsheet` at the address `ch` points to:

- `student_id` should be set to the `student_id` argument.
- `total_length` should be set to the sum of the `strlen` of all strings in `contents`.
- Each `Page` in `pages` should be set as follows:
  - The `num` of `pages[i]` is set to `i`.
  - The `data` of `pages[i]` is set to the `i`th string of `contents`.
- The resulting `cheatsheet` (and all data in its pages) must persist through function calls, even if the strings in `contents` are deallocated.

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);
```

(15 points) Fill in `cheatsheet_init` so that it matches the described behavior.

```
1  void cheatsheet_init(Cheatsheet** ch, int student_id, char** contents) {
2      Cheatsheet* sheet = ____________________________________;
                                        Q2.1
3      sheet_____________ = student_id;
              Q2.2
4      for (int i = 0; i < NUM_PAGES; i++) {
5          Page* page = ____________________________________;
                                    Q2.3
6          page_____________ = i;
                 Q2.4
7          page_____________ = ____________________________;
                 Q2.5                      Q2.6
8          strcpy(page_____________________, contents[i]);
                                Q2.7
9          sheet_____________________ += strlen(contents[i]);
                           Q2.8
10     }
11     ____________________________________________________;
                                Q2.9
12 }
```

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

Q2.10 (2 points) `NUM_PAGES`

○ Stack ○ Heap ○ Code ○ Data/Static

Q2.11 (2 points) `sheet`

○ Stack ○ Heap ○ Code ○ Data/Static

Q2.12 (2 points) `*sheet`

○ Stack ○ Heap ○ Code ○ Data/Static

The `update_contents` function should update the contents of a `Page` if `new_data` has a length that is less than or equal to `MAX_STR_LEN`. `update_contents` does not have to update the `total_length` of a `Cheatsheet`.

Q2.13 (3 points) Is the following implementation of `update_data` correct (follows the described behavior) and memory efficient?

```
1  int MAX_STR_LEN = 100;
2
3  // You may assume that new_data is stored on the heap
4  // and page is well-formed
5  void update_data(Page* page, char* new_data) {
6      if (strlen(new_data) > MAX_STR_LEN) {
7          return;
8      }
9      page->data = new_data;
10 }
```

○ Yes ○ No

If you selected "No", provide a **brief** explanation. If you selected "Yes", leave this box blank. We will only grade the first 15 words of your answer.

Q3 *A Bit of Floating Point* **(14 points)**

For this question, assume that we are working with a binary floating point representation, which follows IEEE-754 standard conventions, but has 5 exponent bits (and a standard bias of -15) and 10 mantissa bits.

Q3.1 (2 points) What is the value of the floating point number `0x7F00`?

Q3.2 (2 points) Consider the floating point number -2.125. What is the largest (closest to $+\infty$) possible value we can represent by modifying a single bit of the floating point representation of this number? Write your answer as a decimal number (e.g. 10.5).

Q3.3 (5 points) Consider the floating point number 7.625. What is the largest (closest to $+\infty$) possible value we can represent by modifying a single bit of the floating point representation of this number? Write the binary representation of each component of your answer.

Sign: 0b

Exponent: 0b

Mantissa: 0b

Q3.4 (5 points) How many non-zero numbers `x` are there in this floating point system where `x` and `2x` differ by exactly 1 bit?

Write your answer as a sum or difference of unique powers of 2 (e.g. $2^3 - 2^2 + 2^1$).

Q4 *The R in RISC-V* **(16 points)**

For each of the following instructions, provide a sequence of RISC-V instructions that computes the equivalent to the given instruction.

- You may **not** use the instruction itself.
- You may **not** use any pseudoinstructions.
- You may **not** have unaligned memory accesses.
- You may assume that `rs1` or `rs2` are not the same register as `rd`.
- You may use **at most** the number of lines we provide.
- You may **only** have one instruction per line.

Each subpart is independent from one another. We've provided `sub` as an example.

`sub rd rs1 rs2`

```
sub_alternative:
    xori rd rs2 -1
    addi rd rd  1
    add rd rs1 rd
```

Q4.1 (5 points) `lbu rd imm(rs1)`

You can use `rs1`, `rd`, and `imm` in your answer. You may not modify any registers other than `rd`.

```
lbu_alternative:

    _____________________________________________

    _____________________________________________
```

Q4.2 (5 points) `bne rs1 rs2 label`

You can use `rs1`, `rs2`, and `label` in your answer. You may not modify any registers.

```
bne_alternative:

    _____________________________________________

    _____________________________________________
continue: # This is a label, but you do not have to use it.
```

Q4.3 (6 points) `auipc rd imm`

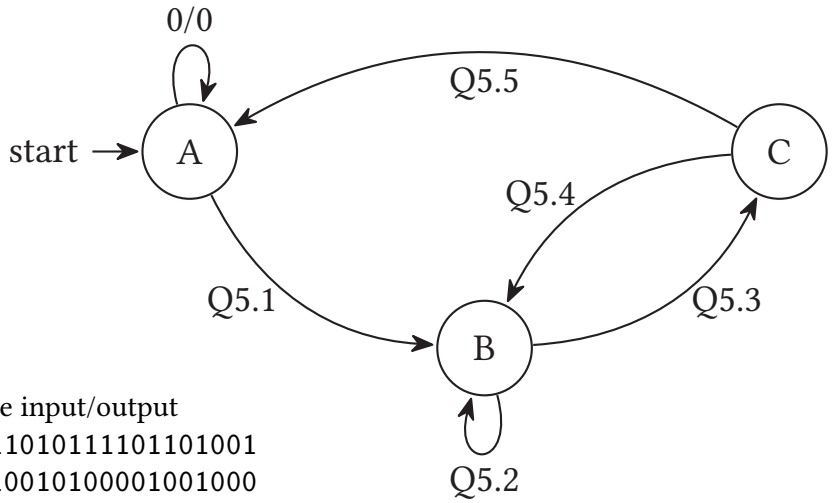
You can use `rd` and `imm` in your answer. You may only modify `rd` and `t0`.

Note: the value of PC used for `auipc` computation should be the PC of the final instruction in your answer. For example, if your answer has four lines, you should add `imm` to the PC of the fourth line.

```
auipc_alternative:

    _____________________________________________________
temp_label: # This is a label, but you do not have to use it.

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________
```

Q5 *FSM* **(10 points)**

The following finite state machine (FSM) should output 1 if the last 3 input bits are 101. Otherwise, it should output 0.



Example input/output

Input: 1011010111101101001

Output: 0010010100001001000

For each of the transitions above, fill in the appropriate input and output values such that the FSM behaves as described.

Q5.1 (2 points)

/

Q5.2 (2 points)

/

Q5.3 (2 points)

/

Q5.4 (2 points)

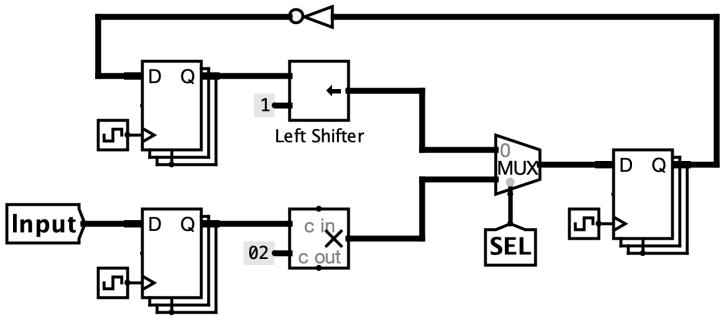
/

Q5.5 (2 points)

/

Q6 *SDS* **(12 points)**

Consider the following circuit diagram. `SEL` is a single bit control signal that updates instantaneously at the rising edge of every clock cycle and remains stable during any given clock cycle. You may assume that `Input` will not cause any hold time violations.



$t_{\text{NOT}} = 8$ ps
$t_{\text{mux}} = 25$ ps
$t_{\text{multiplier}} = 1000$ ps
$t_{\text{shifter}} = 2$ ps
$t_{\text{clk-q}} = 30$ ps
$t_{\text{setup}} = 20$ ps

The left shifter combinational logic block shifts the top input by the number of bits indicated by the bottom input. The shifter in the diagram shifts the output of the connected register left by 1 bit.

Q6.1 (3 points) What is the minimum clock period for the circuit above such that it will always result in well-defined behavior?

ps

Q6.2 (3 points) What is the maximum hold time the registers can have so that there are no hold time violations in the circuit above? Reminder: you may assume that `Input` will not cause any hold time violations.

ps

Eric wants to make one change to increase the circuit's frequency the most without changing the behavior of the circuit. He cannot change the delays of any component.

Q6.3 (3 points) **Briefly** describe a change Eric should make. We will only grade the first 15 words of your answer.

Q6.4 (3 points) What is the new minimum clock period after implementing the change in Q6.3?

ps

Nothing on this page will be graded.

Is there anything you want us to know?