CS 61C Fall 2025

Garcia Midterm

PRINT Your Name: _____

PRINT Your Student ID: _____

PRINT the Name and Student ID of the person to your left: _____

PRINT the Name and Student ID of the person to your right: _____

PRINT the Name and Student ID of the person in front of you: _____

PRINT the Name and Student ID of the person behind you: _____

You have 170 minutes. There are 7 questions of varying credit. (100 points total)

Question:	1	2	3	4	5	6	7	Total
Points:	22	12	15	23	15	13	0	100

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

- ○ Unselected option (Completely unfilled)
- ◌✓ Don't do this (it will be graded as incorrect)
- ● Only one selected option (completely filled)

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

- ■ You can select
- ■ multiple squares
- ☑ (Don't do this)

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 (`0xB0BACAFE` instead of `0xb0bacafe`). 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.

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I will follow the rules of this exam.

Acknowledge that you have read and agree to the honor code above and sign your name below:

Q1 *Trees...* 💡🧠 (22 points)

Suppose that we implement a binary tree using the `node_t` struct, defined as follows:

```
1 typedef struct Node {
2   char name[6];
3   struct Node *left, *right;
4   int64_t *data;
5 } node_t;
```

For Q1.1 – Q1.14, you may assume that:

- We are on a **64-bit** system.
- `sizeof(size_t) == sizeof(void*) == 8`
- All pointers are word-aligned
- In any `node_t`, `name` is a valid string containing at most 5 characters.

Q1.1 (2 points) What is `sizeof(node_t)`?

	bytes

(10 points) Implement the `node_new` function. This function should create a new `node_t` on the heap with the provided argument as its `name`, and return a pointer to the newly created `node_t` struct.

`node_new`: Initializes a `node_t` with the given `name`.		
Arguments	`char *name`	The name of the `node_t` to be created. Assume `name` points to a valid string and `strlen(name) <= 5`.
	`int64_t data`	The value to be stored in the newly created `node_t`
Return value	`node_t *`	A pointer to the newly created `node_t` struct.

```
1 node_t* node_new(char *name, int64_t data) {
2     node_t *n = calloc(_____Q1.2_____, _____Q1.3_____);
3     if (!n) return NULL;
4     memcpy(_____Q1.4_____, _____Q1.5_____, _____Q1.6_____);
5     n->data = malloc(_____Q1.7_____);
6     if (!(n->data)) return NULL;
7     _____Q1.8_____;
8     return n;
9 }
```

Useful C function prototypes:

```
size_t strlen(const char *s);
void *memcpy(void *dest, const void *src, size_t n);
```

(4 points) Implement the `free_tree` function. This function should free all memory allocated for the given binary tree rooted at `t`, including all `data` and subtrees. You may assume that all nodes in the tree are heap-allocated.

```
1 void free_tree(node_t *t) {
2     if (!t) return;
3     _____Q1.9_____;
4     _____Q1.10_____;
5     _____Q1.11_____;
6     _____Q1.12_____;
7 }
```

For Q1.13 – Q1.14, assume that you have finished running the `node_new` function but have not returned from the function yet. For each of the following symbols, indicate which segment of memory it is stored in.

Q1.13 (1 point) `memcpy`

◯ Stack ◯ Heap ◯ Static ◯ Code

Q1.14 (1 point) `n->name`

◯ Stack ◯ Heap ◯ Static ◯ Code

Q1.15 *The Call*

For each of the following questions, select the step of CALL that performs the operation.

Q1.15.1 (1 point) Copies the executable file into memory to prepare for execution.

◯ Compiler ◯ Assembler ◯ Linker ◯ Loader

Q1.15.2 (1 point) Replaces the immediate in `la label` with its final value. (`label` is defined in a different file than the `la` instruction)

◯ Compiler ◯ Assembler ◯ Linker ◯ Loader

Q1.15.3 (1 point) Expands pseudoinstructions into their corresponding machine instructions.

◯ Compiler ◯ Assembler ◯ Linker ◯ Loader

Q1.15.4 (1 point) Takes high-level language code as input.

◯ Compiler ◯ Assembler ◯ Linker ◯ Loader

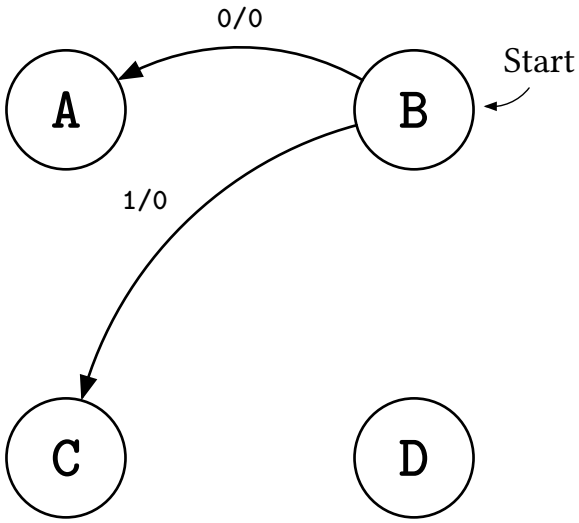
Q2 *It Takes Two* 👫 **(12 points)**

Complete the below FSM that swaps the left and right bit in each pair of bits. For example, if we let each letter represent a bit, the input `stuvwxyz-` should yield an output of `0tsvuxwzy`, as shown in the table below.

Example Input	`stuvwxyz-`
Example Output	`0tsvuxwzy`

Assume the first output is always a `0` (since you can't output bit `t` until you see it), and the `-` at the end represents some input bit so that `y` can be outputted.

Q2.1 (12 points) Fill in the table describing the state transitions of this FSM. Two transitions have been given to you in the diagram and filled out in the table for your reference.



Current State	Input	Next State	Output
A	0	○ A ○ B ○ C ○ D	○ 0 ○ 1
A	1	○ A ○ B ○ C ○ D	○ 0 ○ 1
B	0	● A ○ B ○ C ○ D	● 0 ○ 1
B	1	○ A ○ B ● C ○ D	● 0 ○ 1
C	0	○ A ○ B ○ C ○ D	○ 0 ○ 1
C	1	○ A ○ B ○ C ○ D	○ 0 ○ 1
D	0	○ A ○ B ○ C ○ D	○ 0 ○ 1
D	1	○ A ○ B ○ C ○ D	○ 0 ○ 1

Q3 *Jump Up, Super Star!* ⭐ **(15 points)**

Q3.1 (3 points) Translate the instruction on line `3` to hexadecimal RISC-V machine code.

```
1 label: addi a1 x0 1
2        slli a1 a1 2
3        bne a3 t2 label
```

`0x`

Q3.2 (1 point) The `imm[0]` bit of a branch instruction can be either `0` or `1`.

◯ True ◯ False

Q3.3 (2 points) If a branch instruction is taken, what is the maximum number of **32-bit** instructions we can jump backwards by? You may express your answer as a power of 2.

Q3.4 (4 points) Consider an ISA called RISC-V 16I, where **every instruction is 16 bits long**.

RISC-V 16I uses a 6-bit, 2's complement signed immediate for B-type instructions, with an implied `0` as the LSB. Everything else about B-type instructions remains the same. What is the range of target addresses for branch instructions, **inclusive**? Write your answers in terms of powers of 2.

$$[\text{PC} - \square\,,\ \text{PC} + \square]$$

(5 points) Implement the `diff_sign` function. You **may only** use the `a0` and `a1` registers in your solution.

`diff_sign`: Determines whether registers `a0` and `a1` have different signs.		
Arguments	`a0`	a nonzero 2's complement integer
	`a1`	a nonzero 2's complement integer
Return value	`a0`	`0` if the inputs are both positive or both negative, `1` if they have different signs

```
1 diff_sign:
2     ____________________ Q3.5
3     ____________________ Q3.6
4     jr ra
```

Q4 *The World Revolving ♠* **(23 points)**

Binary search is an algorithm to find the index of some desired value in a sorted array `arr`. The below C implementation of binary search is given for your reference. For this question, you may assume that `target` appears in `arr`.

```
1  int binary_search(int* arr, int left, int right, int target) {
2    int mid = (left + right) / 2;  // C division rounds down
3    if arr[mid] == target {
4      return mid;
5    } else if target < arr[mid] {
6      return binary_search(arr, left, mid - 1, target);
7    } else {
8      return binary_search(arr, mid + 1, right, target);
9    }
10 }
```

(20 points) Implement the `binary_search` function.

`binary_search`: determines the index of the value in `a3` in the array `a0`.		
Arguments	`a0`	A pointer to a sorted array with integers
	`a1`	Left index; initially set as zero
	`a2`	Right index; initially set as length of array - 1
	`a3`	The target value we want to find
Return value	`a0`	The index of the target element

(Question 4 continued...)

```
1  binary_search:
2      # prologue omitted
3      add s0 a1 a2
4
5      slli s1 s0 2
6
7
8      beq ____Q4.4____ found
9      blt ____Q4.5____ search_left
10     blt ____Q4.6____ search_right
11
12 found:
13     mv a0 s0
14     j done
15 search_left:
16
17
18     j done
19 search_right:
20
21
22 done:
23     # epilogue omitted
24     jr ra
```

Q4.1 ____Q4.1____

Q4.2 ____Q4.2____

Q4.3 ____Q4.3____

Q4.7 ____Q4.7____

Q4.8 ____Q4.8____

Q4.9 ____Q4.9____

Q4.10 ____Q4.10____

Q4.11 (3 points) Select which registers need to be saved in the prologue and restored in the epilogue for `binary_search` to satisfy calling conventions.

☐ a0 ☐ s0 ☐ a1 ☐ s1 ☐ a2 ☐ s2 ☐ ra ◯ None of the above

Q5 *On Little Cat Feet* ☀ **(15 points)**

We want to implement a new RISC-V instruction `smaller rd rs1 rs2` that does the following:

```
1 if (R[rs1] < R[rs2])
2   R[rd] = R[rs1]
3 else
4   R[rd] = R[rs2]
```

Q5.1 (2 points) What instruction type would be the most suitable for `smaller`?

◯ R ◯ I* ◯ B ◯ J

◯ I ◯ S ◯ U ◯ No existing instruction type works

Q5.2 (1 point each) We decide to modify **only** the ALU to implement this (and select a new `ALUSel` value to trigger any potential new operations). What are the values of the control signals necessary to implement `smaller`? Fill in `*` if the value of the control signal does not matter.

`PCSel`	◯ `PC + 4`		◯ ALU	◯ Depends on `BrEQ` and `BrLT`	◯ `*`
`ASel`	◯ Reg		◯ `PC`	◯ Depends on `BrEQ` and `BrLT`	◯ `*`
`BSel`	◯ Reg		◯ Imm	◯ Depends on `BrEQ` and `BrLT`	◯ `*`
`MemRW`	◯ Read		◯ Write	◯ Depends on `BrEQ` and `BrLT`	◯ `*`
`RegWEn`	◯ `1`		◯ `0`	◯ Depends on `BrEQ` and `BrLT`	◯ `*`
`ImmSel`	◯ I	◯ B	◯ S	◯ Depends on `BrEQ` and `BrLT`	◯ `*`
`WBSel`	◯ ALU	◯ Mem	◯ `PC + 4`	◯ Depends on `BrEQ` and `BrLT`	◯ `*`

(Question 5 continued...)

Suppose we keep the ALU modification but make the **`smaller`** instruction do the following instead:

```
1 if (R[rs1] < R[rs2])
2   R[rd] = R[rs1]
3 else
4   Mem[R[rd]] = R[rs2]  # New change
```

Q5.3 (2 points) What instruction type would be the most suitable for the new version of **`smaller`**?

◯ R ◯ I* ◯ B ◯ J

◯ I ◯ S ◯ U ◯ No existing instruction type works

Q5.4 (4 points) What additional changes, if any, would we need to make to our single-cycle datapath for us to implement this (with as few changes as possible)? Select all that apply.

☐ Add a new read input to the RegFile for a third register value.

☐ Add a new read output to the RegFile for a third read register value

☐ Add a new WriteData and WriteIndex input to the RegFile

☐ Add a third possible value for **`ASel`** and update the corresponding MUX/control logic

☐ Add a third possible value for **`BSel`** and update the corresponding MUX/control logic

☐ Add a third possible value for **`PCSel`** and update the corresponding MUX/control logic

☐ Add a new read input to DMEM for a second memory read output.

☐ Add a MUX in front of DMEM's **`addr`** input and a new control line to control it

☐ Add a MUX in front of DMEM's **`wdata`** input and add a new control line to control it

☐ Add a fourth possible value for **`WBSel`** and update the corresponding MUX/control logic

◯ None of the above

Q6 *StarDrop Saloon* 🧑‍🍳 **(13 points)**

Q6.1 (3 points) Write a Boolean expression for the below table in terms of `A`, `B`, and `C`. For full credit, you may use at most 2 operators. Your answer may consist of the following operators and symbols:

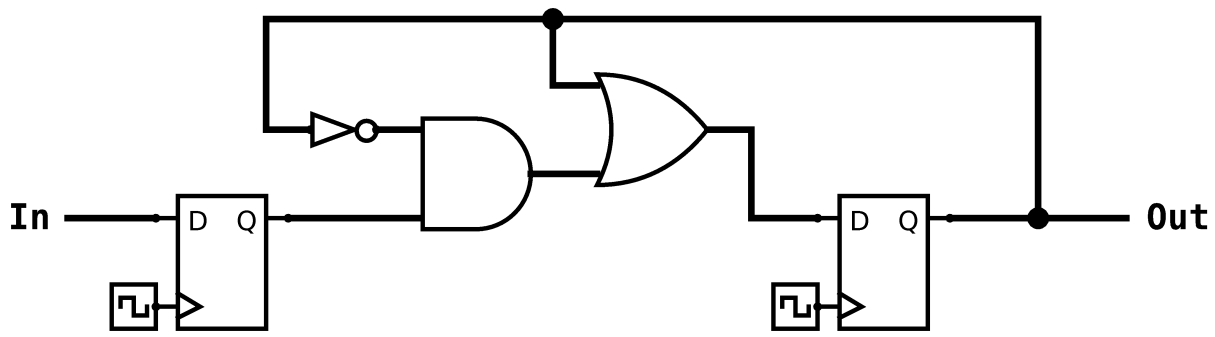
Operators			Symbols		
NOT	AND	OR	Inputs	Constants	Parentheses
`!A`	`A * B`	`A + B`	`A, B, C`	`0, 1`	`()`

A	B	C	Output
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

Output =

The rest of this page left intentionally (mostly) blank.

For Q6.2 — Q6.4, consider the following circuit and timings



with the following circuit times for the components used above

$$t_{\texttt{setup}} = 10 \text{ ps} \quad t_{\texttt{clk-to-q}} = 4 \text{ ps} \quad t_{\texttt{NOT}} = 1 \text{ ps} \quad t_{\texttt{AND}} = 2 \text{ ps} \quad t_{\texttt{OR}} = 3 \text{ ps}$$

Q6.2 (3 points) What is the minimum clock period this circuit can have to consistently exhibit well-defined behavior?

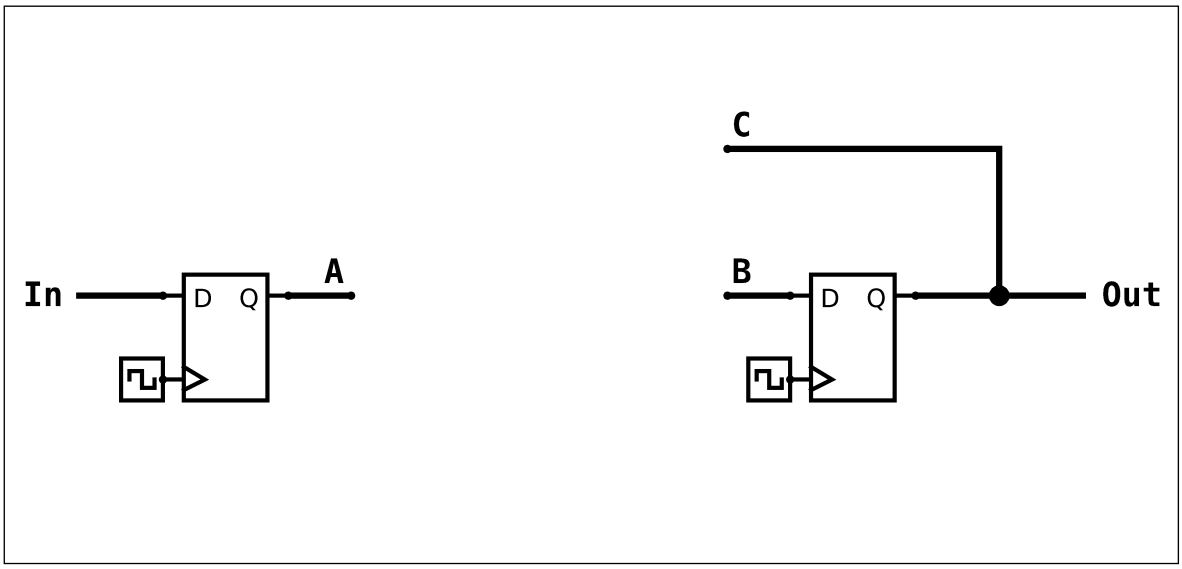
	ps

Q6.3 (3 points) What is the range of possible hold time values (in ps) for this circuit to consistently exhibit well-defined behavior?

E.g. if the hold time had to be bigger than than 99ps you would write $\texttt{99ps} \leq t_{\texttt{hold}}$ *and leave the second box blank. If the hold time had to be between 57ps and 88ps, you would write* $\texttt{57ps} \leq t_{\texttt{hold}} \leq \texttt{88ps}$

	ps	$\leq t_{\texttt{hold}} \leq$	ps

Q6.4 (4 points) To increase the maximum clock frequency, simplify and reconnect the circuit by appropriately joining the wires labeled (`A`, `B`, `C`) and drawing **at most one** gate, ensuring that the circuit's functionality remains identical to the original.



Q7 *The Finishing Touch* 🥤🤯 **(0 points)**

These questions will not be assigned credit. Feel free to leave them blank.

Q7.1 What is the common theme across these question names? What references do you recognize? :>

Q7.2 If there's anything else you want us to know, or you feel like there was an ambiguity in the exam, please put it in the box below.

For ambiguities, you must qualify your answer and provide an answer for both interpretations. For example, "if the question is asking about A, then my answer is X, but if the question is asking about B, then my answer is Y". You will only receive credit if it is a genuine ambiguity and both of your answers are correct. We will only look at ambiguities if you request a regrade.