CS61C Fall 2024

Garcia, Kao Final

PRINT Your Name: _____

PRINT Your Student ID: _____

You have 170 minutes. There are 11 questions of varying credit. (100 points total)

Question:	1	2	3	4	5	6	7	8	9	10	11	Total
Points:	8	12	7	10	6	8	14	14	11	10	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 (`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: _____

Q1 *Potpourri* **(8 points)**

Q1.1 (3 points) Consider an 8-bit floating point format that follows the IEEE-754 standard, with 1 sign bit, 4 exponent bits (with a standard bias of −7), and 3 mantissa bits.

What is the minimum distance between any two denormalized numbers in this floating point format? Express your answer as a power of 2.

Q1.2 (3 points) Consider the following multi-threaded code block.

```
1  int32_t a = 0;
2  int32_t b = 2;
3
4  #pragma omp parallel {
5    while (b > 0) {
6      a = a + b;
7      #pragma omp critical {
8        b = b - 1;
9      }
10   }
11 }
```

If we run this code with two threads, what is the largest possible value of `a` after both threads finish execution?

Note that the expression `a = a + b` is equivalent to four instructions: load the value of `a`, load the value of `b`, sum `a` and `b`, and then store the result in `a`.

Q1.3 (2 points) Select all true statements about the manager-worker framework.

- ☐ If one program crashes, the others keep going.
- ☐ Programs communicate by sending messages between each other.
- ☐ The manager-worker framework splits a problem into independent subtasks and tries to minimize communication between programs.
- ☐ The manager is able to assign a task to a worker before the worker is ready.
- ○ None of the above

Q2 *LibraR(y)ISC-V* **(12 points)**

```
1  typedef struct {
2    uint32_t page_num;
3    char *content;
4    uint32_t is_read;   // 0 if unread, 1 if read
5  } page_t;
6
7  typedef struct {
8    page_t *pages;     // array of pages
9    page_t *bookmark;  // pointer to the first unread page
10 } book_t;
```

Implement `read_pages` to match the described behavior.

`read_pages`: Sets `is_read` to `1` for the next `n` pages of a `book_t`, starting from the bookmark. `read_pages` updates the bookmark and pages **in place**.		
Arguments	`a0`	A pointer to a `book_t` struct with at least `n` pages after the bookmark.
	`a1`	`n`, the number of pages to mark as read.
Return value	`void`	

Unfortunately, Anto has spilled apple juice on your compiler, so you need to fill in the `read_pages` function in RISC-V.

A struct stores only its members, with no metadata and no extra compiler padding. For example, if a `page_t` struct is located at address `0x1000`, its `page_num` is at `0x1000` and its `content` is at `0x1004`.

```
1  read_pages:
2    ____Q2.1____ t3 ____Q2.2____(____Q2.3____) # bookmark
3    li t1 1
4  loop:
5    beq ____Q2.4____
6    ____Q2.5____
7    addi t3 ____Q2.6____
8    addi a1 ____Q2.7____
9    j loop
10 end:
11   ____Q2.8____ t3 ____Q2.9____(____Q2.10____)
12   ret
```

Q3 *MixC Mystery* **(7 points)**

Consider the `mystery` function, which takes in one argument in `a0`, and returns one output in `a0`.

```
1 mystery:
2   lbu t0 0(a0)
3   li a0 0
4 loop:
5   andi t1 t0 1
6   add a0 a0 t1
7   srai t0 t0 1
8   bne t0 x0 loop
9   ret
```

Q3.1 (1 point) What does the `mystery` function return?

- ◯ Bitwise `and` of `0x01` and the byte pointed to by `a0`
- ◯ Number of binary `1`s in the byte pointed to by `a0`
- ◯ Always returns `8`

For Q3.2 – Q3.4, suppose we replaced the `lbu` on line 2 with `lb`, which introduces a bug. The buggy `mystery` and the original `mystery` now behave differently. To help find the difference between their behaviors, we use the C function defined below:

```
1 void mysterytest(uint8_t n) {
2   printf("%d\n", mystery(&n));
3 }
```

Q3.2 (3 points) Complete the following sentence:

If the value of `n` is `0x91`, the original `mystery` printed the decimal value…

however, the buggy `mystery` instead…

◯ prints 0 ◯ prints 27 ◯ segfaults ◯ infinitely loops

Q3.3 (2 points) What is the smallest 8-bit value of `n` that results in the buggy behavior?

Q3.4 (1 point) Which section of memory does the symbol `n` live in?

◯ Code ◯ Static ◯ Heap ◯ Stack

Q4 *Datapath Jadditions* **(10 points)**

For this question, assume we are working with the single cycle datapath.

A `jaddi` instruction in RISC-V is a new instruction described as follows:

`jaddi rd rs1 imm`

```
1 rd = rs1 + imm
2 PC = rs1 + imm
```

For each of the control signals, indicate the value it should always have for `jaddi`. You may assume that we only jump to word-aligned addresses.

Q4.1 (1 point) **`PCSel`**

- ◯ `PC + 4`
- ◯ ALU output
- ◯ Doesn't matter

Q4.2 (1 point) **`ASel`**

- ◯ RegReadData1
- ◯ `PC`
- ◯ Doesn't matter

Q4.3 (1 point) **`BSel`**

- ◯ RegReadData2
- ◯ ImmGen output
- ◯ Doesn't matter

Q4.4 (1 point) **`ALUSel`**

- ◯ `add`
- ◯ `and`
- ◯ Doesn't matter
- ◯ `sub`
- ◯ `or`

Q4.5 (1 point) **`MemRW`**

- ◯ Memory read
- ◯ Memory write
- ◯ Doesn't matter

Q4.6 (1 point) **`WBSel`**

- ◯ Mem
- ◯ ALU output
- ◯ Doesn't matter
- ◯ `PC + 4`

Q4.7 (1 point) **`RegWEn`**

- ◯ Write enabled
- ◯ Write disabled
- ◯ Doesn't matter

Consider the new instruction `jalm` below.

`jalm rd imm(rs1)`

```
1 rd = PC + 4
2 PC = 4 bytes of memory starting at address (rs1 + imm)
```

Q4.8 (3 points) What additional changes, if any, would we need to make to our single-cycle datapath in order for us to implement `jalm` (with as few changes as possible)? Select all that apply.

- ☐ Create a new instruction type and update the ImmGen.
- ☐ Add a new read input to the RegFile for a third 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 new ALU operation and update any relevant selector/control logic.
- ☐ Add a third possible value for `PCSel` and update the corresponding MUX/control logic.
- ☐ Allow the DMEM to be able to read and write at the same clock cycle and update any relevant selector/control logic.
- ☐ Add a new read input to DMEM for a second memory read output.
- ☐ Add a fourth possible value for `WBSel` and update the corresponding MUX/control logic.
- ◯ None of the above

Q5 *Hazardous Ordering* **(6 points)**

In this question, use the five-stage pipeline on the reference card. Assume that:
- The RegFile can perform write-then-read on the same clock cycle (also called double pumping).
- There is no forwarding.
- We always predict that the branch is not taken.

For questions Q5.1 – Q5.2, identify the number of stalls needed and the hazard type between the indicated lines in the code block below. If you select None as the hazard type, write "N/A" in the box.

```
1 addi t0 x0 1
2 lw t1 0(s0)
3 sw t1 4(s0)
```

Q5.1 (2 points) Between lines 1 and 2:

stall(s)

◯ Control ◯ Structural
◯ Data ◯ None

Q5.2 (2 points) Between lines 2 and 3:

stall(s)

◯ Control ◯ Structural
◯ Data ◯ None

Q5.3 (2 points) Rearrange the instructions below to minimize the number of stalls while maintaining the same behavior.

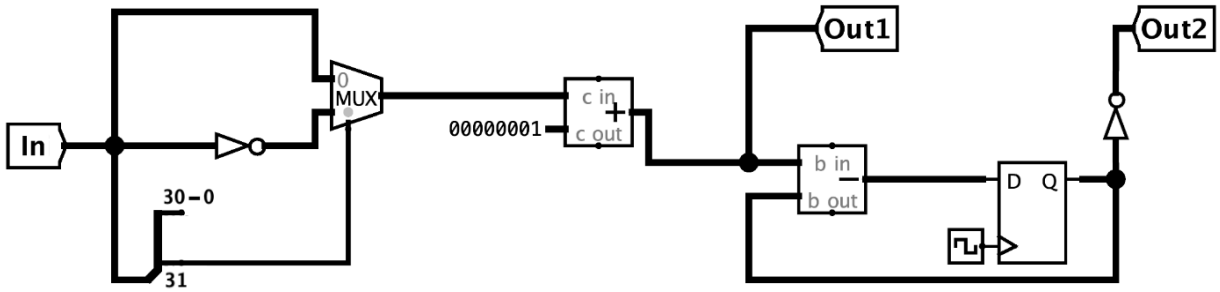
```
1 addi t0 x0 4  # Instruction A
2 addi t1 t0 4  # Instruction B
3 lw s0 0(s1)   # Instruction C
4 add a0 s0 t1  # Instruction D
```

Format your answer as a comma-separated list. For example, the instruction order in the code block above would be described as "A, B, C, D".

Q6 *Separate Timings* **(8 points)**

In this question, assume that:

- All registers are initialized to 0.
- The 32-bit tunnels `In`, `Out1`, and `Out2` are directly connected to registers.
- The select bit of the MUX is wired to bit 31 of `In` via the splitter.
- The NOT gate outputs the bitwise `not` of its input (e.g. `not(0b01010) == 0b10101`).



$t_{\text{clk-to-q}} = 10\text{ns}$ $t_{\text{setup}} = 5\text{ns}$ $t_{\text{not}} = 30\text{ns}$

$t_{\text{mux}} = 15\text{ns}$ $t_{\text{adder}} = 10\text{ns}$ $t_{\text{subtractor}} = 20\text{ns}$

Q6.1 (2 points) What is the **minimum clock period** for this circuit to function properly, in nanoseconds?

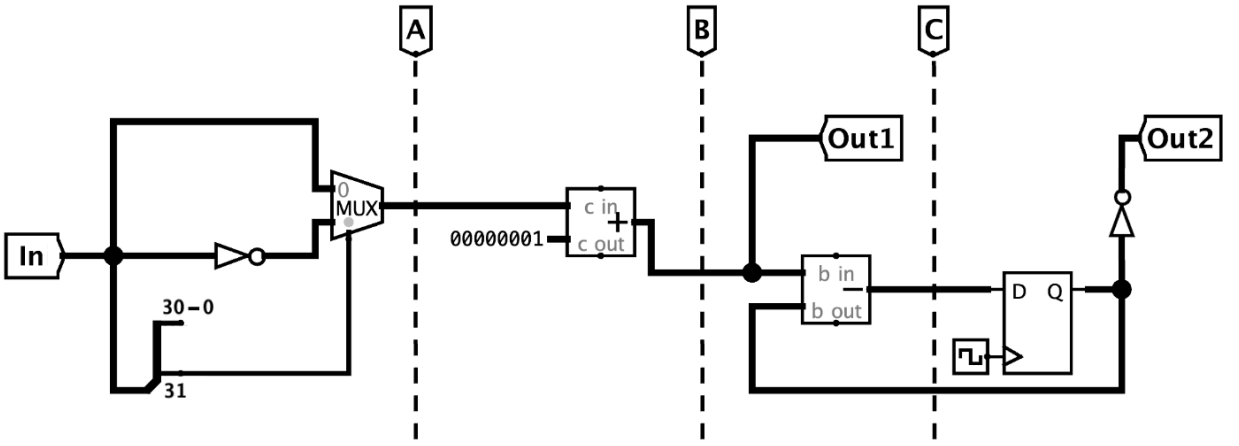
ns

Q6.2 (2 points) What is the **maximum hold time** for this circuit to function properly, in nanoseconds?

ns

Q6.3 (2 points) If we pass `In` $= -6$ to the circuit as a 32-bit two's complement integer, what will the output at `Out1` be, in decimal?

We want to improve the performance of this circuit by adding a **single** pipeline register to create a **two-stage** pipeline. Three possible separations between the stages are drawn on the circuit diagram below.



The delays are repeated for your convenience:

$$t_{\text{clk-to-q}} = 10\text{ns} \qquad t_{\text{setup}} = 5\text{ns} \qquad t_{\text{not}} = 30\text{ns}$$
$$t_{\text{mux}} = 15\text{ns} \qquad t_{\text{adder}} = 10\text{ns} \qquad t_{\text{subtractor}} = 20\text{ns}$$

Q6.4 (2 points) Which separation allows for the highest possible clock frequency? Explain in ten words or fewer.

◯ Separation A ◯ Separation B ◯ Separation C

Q7 *Caches* **(14 points)**

Q7.1 (2 points) What is the tag-index-offset breakdown of a 256B, fully-associative cache with a 32B block size on a 16-bit system?

Tag: bit(s)	Index: bit(s)	Offset: bit(s)

For Q7.2 – Q7.4, assume we are using a 2-way set-associative cache with a First-In-First-Out (FIFO) replacement policy, and each address has 9 tag bits, 3 index bits, and 4 offset bits.

You may assume the following:

- The cache starts out empty.
- In expressions of the form `a += b`, `a` is read first, then `b` is read, and then `a` is written.

```
1 int32_t arr[32 * 32]; // arr starts at address 0x0100
2 for (register int32_t i = 0; i < 32 ; i++) {
3   for (register int32_t j = 1; j < 32; j++) {
4     arr[i] += arr[32 * j + i];
5   }
6 }
```

Q7.2 (2 points) Which address is accessed on the first compulsory miss?

0x

Q7.3 (2 points) How many cache hit(s) occur on the first execution of the innermost statement? (`i = 0`, `j = 1`)

◯ 0 hits ◯ 1 hit ◯ 2 hits ◯ 3 hits

Q7.4 (2 points) How many cache hit(s) occur on the second execution of the innermost statement? (`i = 0`, `j = 2`)

◯ 0 hits ◯ 1 hit ◯ 2 hits ◯ 3 hits

For Q7.6 – Q7.9, complete the code below to be more cache-efficient while maintaining the same behavior as above. You may assume the original cache parameters: 2-way set-associative cache with 9 tag bits, 3 index bits, 4 offset bits, and FIFO replacement policy.

```
1 int32_t array[32 * 32]; // arr starts at address 0x0100
2 for (register int32_t i = ______; i < 32; i++) {
                            Q7.6
3   for (register int32_t j = [______]; j < 32; j++) {
                               Q7.7
4     array[______] += array[______];
            Q7.8                Q7.9
5   }
6 }
```

Q7.6 (1 point) ◯ `1` ◯ `0` ◯ `31` ◯ `32 * 32`

Q7.7 (1 point) ◯ `1` ◯ `0` ◯ `31` ◯ `32 * 32`

Q7.8 (1 point) ◯ `i` ◯ `j` ◯ `32 * i` ◯ `32 * j`

Q7.9 (1 point) ◯ `32 * i` ◯ `32 * j` ◯ `32 * j + i` ◯ `32 * i + j`

Q7.10 (2 points) Consider the two below memory systems:

	Hit Time	Hit Rate
L1 Cache	10ns	60%
DRAM	100ns	100%

	Hit Time	Hit Rate
L1 Cache	10ns	60%
L2 Cache	70ns	?
DRAM	100ns	100%

If the AMAT (average memory access times) of the two systems are equal, what is the local hit rate of the L2 Cache in the right system?

Hit Rate: ____ %

Q8 `is_odd_and_parallel` **(14 points)**

Taki is given a sequence of characters and needs to find the odd characters. An odd character is a character that appears an odd number of times in the sequence. Help Taki by implementing the below function.

`get_odd_characters`:		
Arguments	`uint32_t *seq`	An array of characters. Each character is zero-extended to 32 bits and stored in a `uint32_t`.
	`int size`	The number of characters in `seq`.
Return value	`uint32_t`	A bit array of the characters that are odd, stored in a `uint32_t`.

For example, given the characters `a c c h h h b`:

- The odd characters are `a h b`.
- `get_odd_characters` should return `0b0000 0000 0000 0000 0000 0000 1000 0011`, a bit array where each bit corresponds to a character, where bit 0 corresponds to `a`, and bit 25 corresponds to `z`. (Note that bits 26–31 are unused.)

Here is a correct implementation of `get_odd_characters.` Your answer should have the same behavior.

```
1 uint32_t get_odd_characters(uint32_t *seq, int size) {
2   uint32_t res = 0;
3   for (int i = 0; i < size, i++) {
4     res ^= 1 << (seq[i] - 0x61);
5   }
6   return res;
7 }
```

You have access to the following SIMD operations. A `vector` is a 128-bit vector register capable of holding four 32-bit integers:

- `vector vec_load(uint32_t *A)`: Loads four integers at memory address `A` into a vector.
- `void   vec_store(uint32_t *dst, vector src)`: Stores `src` to `dst`.
- `vector vec_setnum(uint32_t num)`: Creates a vector where every element is equal to `num`.
- `vector vec_and(vector A, vector B)`: Returns the result of ANDing `A` and `B` element-wise.
- `vector vec_or (vector A, vector B)`: Returns the result of ORing `A` and `B` element-wise.
- `vector vec_xor(vector A, vector B)`: Returns the result of XORing `A` and `B` element-wise.
- `vector vec_add(vector A, vector B)`: Returns the result of adding `A` and `B` element-wise.
- `vector vec_sub(vector A, vector B)`: Returns the result of subtracting `B` from `A` element-wise.
- `vector vec_sll(vector A, vector count)`: Returns the result of left-shifting each element in `A` by the number of bits specified in the corresponding element of `count`.

(Question 8 continued...)

Implement `get_odd_characters` to match the described behavior using SIMD. You may use at most one SIMD instruction per line.

```
1  uint32_t get_odd_characters(uint32_t* seq, int size) {
2    uint32_t res = 0;
3    vector res_vec = vec_setnum(0);
4    vector ones = vec_setnum(1);
5    vector offset = vec_setnum(0x61);
6    for (int i = 0; i < ____Q8.1____; ____Q8.2____) {
7      vector a = ____Q8.3____;
8      vector b = ____Q8.4____;
9      vector c = ____Q8.5____;
10     res_vec = ____Q8.6____;
11   }
12   uint32_t arr[4];
13   vec_store(____Q8.7____);
14   ____Q8.8____;
15
16   for (int i = ____Q8.9____; i < size; i++) {
17     res ^= 1 << (seq[i] - 0x61);
18   }
19   return res;
20 }
```

Q9 *Virtually Valid* **(11 points)**

For Q9.1 – Q9.3, suppose we have a system with 4 GiB of virtual memory, 1 GiB of physical memory, 4 KiB pages, and 4B page table entries.

Q9.1 (2 points) How many bits are in the Virtual Page Number (VPN), Physical Page Number (PPN), and Page Offset?

VPN: bit(s)	PPN: bit(s)	Offset: bit(s)

Q9.2 (1 point) How many entries are in the page table? You may express your answer as a power of 2.

Q9.3 (2 points) How many physical pages are needed to store the page table? You may express your answer as a power of 2.

Reminder: Page table entries are 4 bytes each.

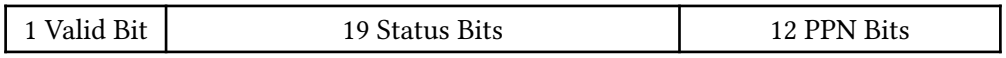
For the remaining parts, assume we have 16-bit VPNs, 12-bit PPNs, 8-bit page offsets, and 32-bit page table entries. The TLB and the first six entries of the page table are shown below.

The next available free page has PPN `0x42D`.

Page Table
0xB61C 0483
0xFB83 A61C
0x8483 3F01
0x7ABC 4103
0xC012 F7CB
0x15DA C203
...

TLB		
Valid	**VPN**	**PPN**
1	0x0000	0x483
1	0x0001	0x61C
0	0x0002	0xB83
0	0x0005	0x483

Each page table entry (PTE) is formatted as:



For each of the following virtual addresses, translate it to its corresponding physical address and answer whether accessing it will result in a TLB hit, TLB miss and page table hit, or a page fault. Assume each access occurs independently, not sequentially.

Q9.4 (2 points) `0x000529`

0x

- ◯ TLB Hit
- ◯ TLB Miss and Page Table Hit
- ◯ Page Fault

Q9.5 (2 points) `0x00018D`

0x

- ◯ TLB Hit
- ◯ TLB Miss and Page Table Hit
- ◯ Page Fault

Q9.6 (2 points) `0x002045`

0x

- ◯ TLB Hit
- ◯ TLB Miss and Page Table Hit
- ◯ Page Fault

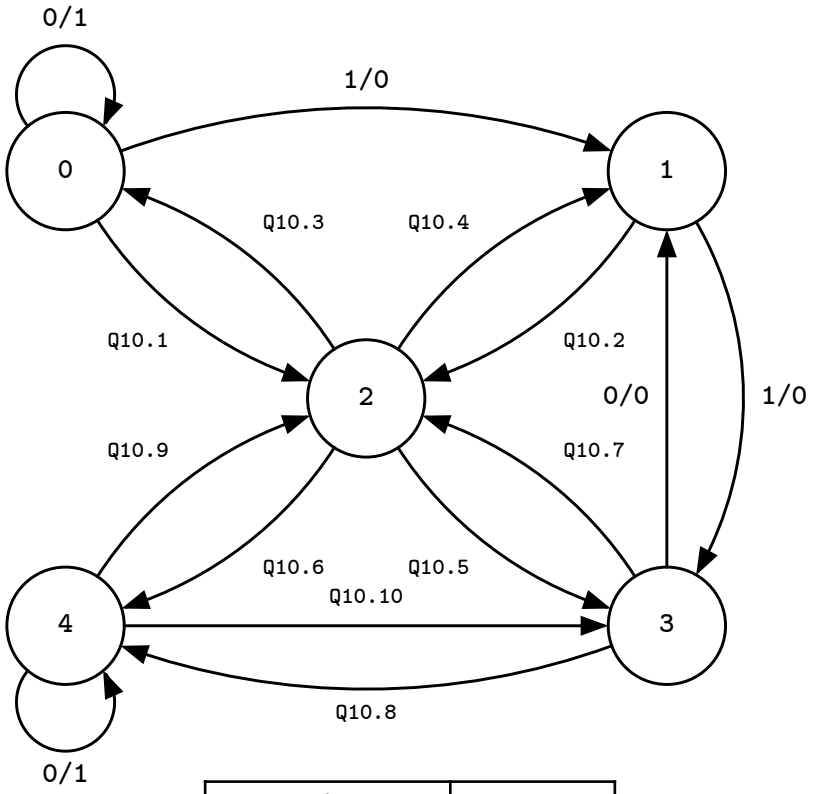
Q10 *Five diviSion Machine* **(10 points)**

Implement an FSM that returns 1 if the input bits so far, interpreted as an unsigned binary number, are divisible by 5. You may assume that the most significant bit is the first bit passed in.

Reminder: 0 is divisible by 5.

Hint 1: States `0`, `1`, `2`, `3`, `4` represent the current number having a remainder of 0, 1, 2, 3, and 4 when divided by 5, respectively.

Hint 2: If you have a binary number $a =$ `0bXXX`, then `0bXXX1` $= 2a + 1$ and `0bXXX0` $= 2a$.



Example Input	01010101
Example Output	10011001

For the above example, the following table summarizes the computations performed by the FSM.

Input bits so far	Value of input bits in decimal	FSM Output
0	0	1
01	1	0
010	2	0
0101	5	1
01010	10	1
010101	21	0
0101010	42	0
01010101	85	1

Fill out the transitions below. For any unused transitions, select N/A.

Q10.1 (1 point) Transition `0` ⟶ `2`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q10.2 (1 point) Transition `1` ⟶ `2`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q10.3 (1 point) Transition `2` ⟶ `0`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q10.4 (1 point) Transition `2` ⟶ `1`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q10.5 (1 point) Transition `2` ⟶ `3`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q10.6 (1 point) Transition `2` ⟶ `4`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q10.7 (1 point) Transition `3` ⟶ `2`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q10.8 (1 point) Transition `3` ⟶ `4`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q10.9 (1 point) Transition `4` ⟶ `2`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

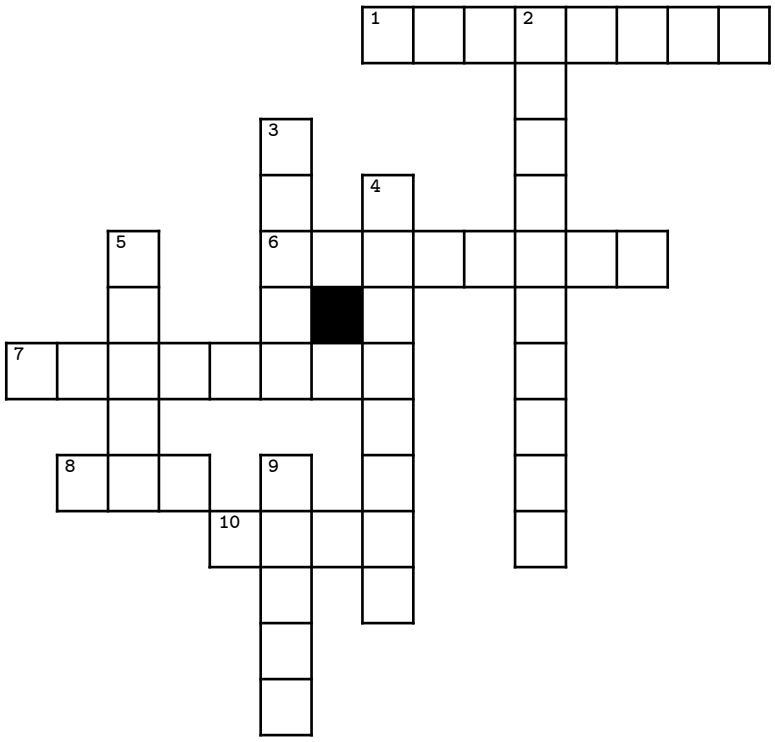
Q10.10 (1 point) Transition `4` ⟶ `3`

◯ `0/0` ◯ `0/1` ◯ `1/0` ◯ `1/1` ◯ N/A

Q11 *61Crossword* **(0 points)**

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

Q11.1 Fill out the 61Crossword!



Across

1. CS61C Fall 2024's mascot 🐿️, perhaps a pun on the class logo?
6. 0xFFFFFFFF + 0x00000001 causes?
7. 💀🥩?
8. a control signal, or a certain staff member?
10. _ _ _ _ me maybe?

Down

2. makes doing laundry faster (and the CPU)
3. _ _ _ _ _'s law of transistor scaling?
4. memory access errors, taken care of by a certain fairy?
5. animal featured in Project 1, rhyming with cake? 🎂
9. 💸?

Q11.2 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.