

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 11 questions of varying credit. (100 points total)

|           |    |    |   |    |    |    |   |   |   |    |    |       |
|-----------|----|----|---|----|----|----|---|---|---|----|----|-------|
| Question: | 1  | 2  | 3 | 4  | 5  | 6  | 7 | 8 | 9 | 10 | 11 | Total |
| Points:   | 11 | 17 | 5 | 15 | 17 | 10 | 5 | 7 | 5 | 8  | 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 ~~eross-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:

---

Clarifications made during the exam:

Q2.6 - 2.14: You may also use  $x_0$  and  $r_a$  in your solution.

Q5: The blurb above Q5.3 should read “For Q5.3-5.6, assume that ...”, and the blurb above Q5.7 should read “For Q5.7, assume the ...”

Q4.4: 2 is not an answer to any of the WBSel options

Q1.1 (1 point) Suppose we declare `int8_t x = 0x9C`; What is the decimal representation of `x`?

Q1.2 (1 point) Suppose we execute the following C code on a 64-bit little-endian system.

```
1 // arr is located at address 0x10000000
2 int64_t arr[3] = {0x01020304, 0x10203040, 0xDE00ABBA};
3 printf("%d", strlen((char*) &arr[1]));
```

What would the `printf` statement output?

Q1.3 (2 points) Assume we are using the standard IEEE-754 Single Precision Floating Point Convention. What is the smallest positive multiple of 2048 (2 Ki) that we cannot represent? Express your answer in sums or differences of powers of 2.

Q1.4 (1 point) Which stage of CALL enables separate compilation of files in a multi-file program?

- Compiler     Assembler     Linker     Loader     None of the above

Q1.5 (1 point) Which stage of CALL directly replaces C code with machine code?

- Compiler     Assembler     Linker     Loader     None of the above

Q1.6 (1 point) Convert the following RISC-V hexadecimal machine code into its corresponding RISC-V instruction. Express immediates in decimal and use the corresponding register names instead of numbers (i.e. `s5` instead of `x21`).

0x043C8713



(Question 1 continued...)

```
1 # Takes a0 a1 a2 as arguments, returns one value in a0
2 stranger:
3   # Q1.7 Prologue
4   srl  s0 s1 2
5   slli a2 a2 2
6   addi s2 t0 4
7
8   # Q1.8 Save
9   jal  ra things
10  # Q1.8 Load
11
12  add  a0 a0 a1
13
14  # Q1.7 Epilogue
15  jr   ra
16
17 # Takes a0 and a1 as arguments, returns two values in a0 and a1
18 things:
19  add  a0 a0 a1
20  sub  a1 a0 a1
21  jr   ra
```

For Q1.7 – Q1.8, select the minimum number of registers which need to be saved and restored to satisfy calling convention.

Q1.7 (1 point) Select all registers we need to save/restore from the stack at the locations marked Q1.7.

a0  s0  a1  s1  a2  s2  t0  t1  None of the above

Q1.8 (1 point) Select all registers we need to save/restore from the stack at the locations marked Q1.8.

a0  s0  a1  s1  a2  s2  t0  t1  None of the above

Q1.9 (2 points) Anto has a program where 1/4 of it is sequential (i.e., not parallelizable), which is executed on 3 cores. What is the max speedup this program can achieve? For full credit, express your answer in its most simplified form.

× speedup

Q2



S d s

(17 points)

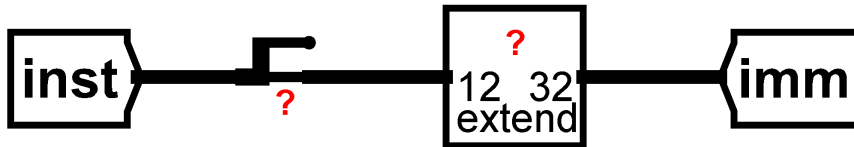
Alex would like to implement a new RISC-V instruction, `lai rd rs1 imm`, with the following functionality.

```
R[rd] = Mem[R[rs1]] [31:0]
R[rs1] = R[rs1] + imm
```

(2 points) This instruction can be implemented using two consecutive, existing RISC-V instructions. Write the instructions and the arguments required to match the functionality of `lai`. Write your answer in terms of `lai` parameters: `rd`, `rs1`, and `imm`.

|   |   |                 |
|---|---|-----------------|
| 1 | <div style="text-align: center; font-size: small;">Q2.1</div> | # Instruction 1 |
| 2 | <div style="text-align: center; font-size: small;">Q2.2</div> | # Instruction 2 |

Q2.3 Oh no! Alex’s dog has eaten their immediate generator design. They tried to recall the design from memory but they need some help remembering some details for their Logisim implementation.



Q2.3.1 (1 point) Considering that `lai` is best represented by an I-type format, what range of bits should be selected by the bottom branch of the splitter? Express your answer in the form `MSB:LSB` (inclusive).

:

Q2.3.2 (1 point) What “Extension Type” should the bit extender use?

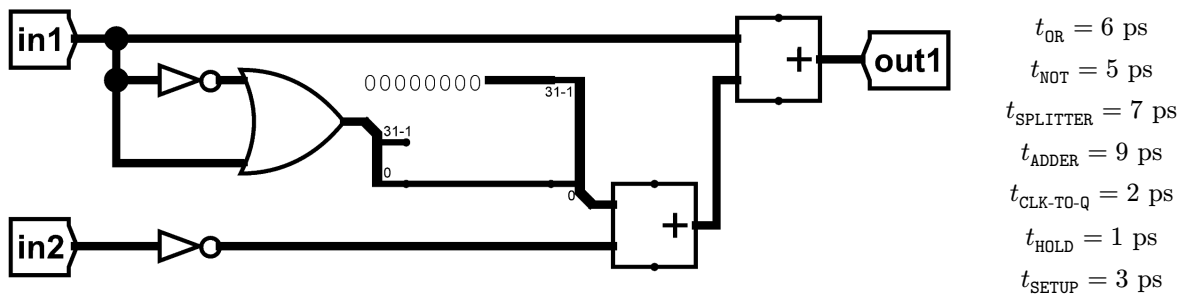
- Zero   
  One   
  Sign   
  Input   
  None of the above

(Question 2 continued...)

Q2.4 (3 points) Now that the immediate generator is fixed, Alex can implement the `lwi` instruction from earlier. What additional changes are required to our single-cycle datapath to implement this, **with as few changes as possible**? Select all that apply.

- Add a new read input to the RegFile
- Add a new read output to the RegFile
- Add a new write data & index input to the RegFile and update the corresponding control logic
- Add a third possible value for `ASe1` and update the corresponding MUX/control logic
- Add a third possible value for `BSe1` and update the corresponding MUX/control logic
- Add a third possible value for `PCSe1` and update the corresponding MUX/control logic
- 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
- Widen `RegWEn` to 2 bits
- None of the above

Q2.5 While we were fixing our pipeline, Alex's dog bit another circuit component! Enoch proposes the following to replace the missing component. Assume each input and output label is connected to a register.



Q2.5.3 (2 points) Express `out1` in terms of `in1`, `in2` and **arithmetic** (not bitwise) operators.

`out1 =`

Q2.5.4 (1 point) What is the minimum clock period this circuit can have to exhibit well-defined behavior, given the above timings?

ps

(Question 2 continued...)

(7 points) Alex wants to use `lai` to find and return the maximum value in an array of 32-bit signed integers. All other load instructions are forbidden. You can assume that the array has at least one element. **You may only use registers `a0`, `a1`, `t0`, and `t1`.**

The description of `lai rd rs1 imm` is copied for your reference:

```
R[rd] = Mem[R[rs1]] [31:0]
R[rs1] = R[rs1] + imm
```

|   |           |  |
|---|-----------|--|
| <b>max_element:</b> Finds and returns the largest signed integer in an array. |           |  |
| <b>Arguments</b>  | <b>a0</b> | Pointer to the start of a signed integer array |
|   | <b>a1</b> | Number of elements in the array                |
| <b>Return value</b>   | <b>a0</b> | The largest value in the array                 |

```
1 max_element:
2   mv t0 a0
3   lai _____
4           Q2.6
5           _____
6           Q2.7
7 loop:
8   beq a1 _____
9           Q2.8      Q2.9
10          _____
11          Q2.10
12          _____
13          Q2.11
14          _____
15          Q2.12
16 skip:
17          _____
18          Q2.13
19   j loop
20 done:
21          _____
22          Q2.14
```

Q3



(5 points)

Complete the C function `is_hazardous`, which determines if two sequential R-type instructions, on a 5-stage pipeline with no forwarding, have any hazards between them (if so, return 1, otherwise 0).

To help with this task, write the utility function `get_register_bits` which returns a 5-bit register index given an instruction's machine code, and the location of the rightmost bit in that register field (i.e., the number above the rightmost bit on the reference card).

For the examples below, note that the machine code representation of `add x5 x9 x13` is `0x00D482B3`.

| Example Input  | Example Output |
|--|----------------|
| <code>get_register_bits(0x00D482B3, &lt;rightmost bit of rd&gt;)</code>  | 5              |
| <code>get_register_bits(0x00D482B3, &lt;rightmost bit of rs1&gt;)</code> | 9              |
| <code>get_register_bits(0x00D482B3, &lt;rightmost bit of rs2&gt;)</code> | 13             |

```

1 uint8_t get_register_bits(uint32_t inst, uint8_t rightmost) {
2     return _____;
3     }
4
5 bool is_hazardous(uint32_t inst1, uint32_t inst2) {
6     // inst1 and inst2 are guaranteed to be R-type instructions
7     uint8_t a = get_register_bits(inst1, _____);
8     uint8_t b = get_register_bits(inst2, _____);
9     uint8_t c = get_register_bits(inst2, _____);
10    return _____;
11 }

```

The Sui siblings Shu and Ling are pipelining their CPU and have asked you for help with hazards! For the rest of this problem, consider the following code snippet:

```

1 auipc a1 0
2 addi a0 x0 3
3 srai t0 a0 2
4 bne x0 t0 done
5 ... # omitted
6 done:
7 ... # omitted
    
```

For Q4.1 – Q4.3, Ling has a **5-stage pipelined CPU**, given on the reference card.

For a fully unoptimized datapath, which hazards occur when the program is run? Please indicate the two instructions that cause the hazard as well as the number of stall cycles needed to fully resolve the hazard.

You may assume that:

- There is no same cycle write-then-read
- There is no branch prediction
- There are no forwarding paths
- All hazards are resolved via stalling

Order the hazards by ascending line number of the originating instruction, then the affected instruction.

Q4.1 (2 points) Hazard 1:

Between instructions  and  Cycles Stalled   Data  Structural  
 Control  N/A

Q4.2 (2 points) Hazard 2:

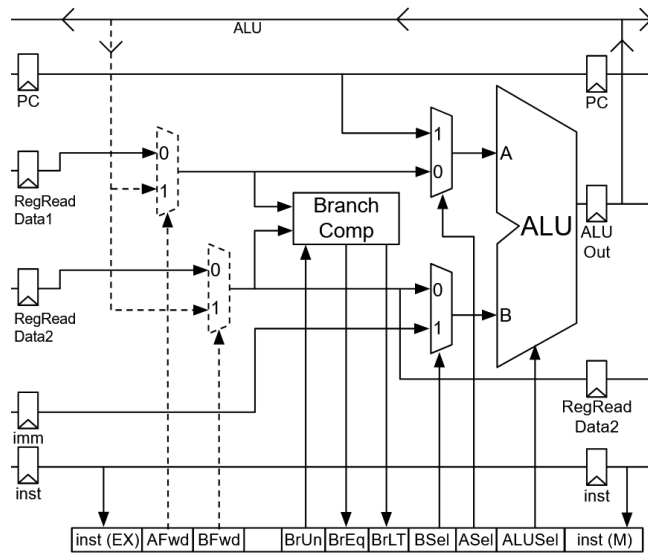
Between instructions  and  Cycles Stalled   Data  Structural  
 Control  N/A

Q4.3 (2 points) Hazard 3:

Between instructions  and  Cycles Stalled   Data  Structural  
 Control  N/A

(Question 4 continued...)

Wanting to optimize Ling's design even further, Shu adds forwarding paths to the 5-stage pipeline



Q4.4 (7 points) Fill in the control signals for each instruction according to the code on the previous page. Select \* if the signal value does not matter. A portion of the registers for each instruction has been written below.

|                | AFwd                    | BFwd                    | ASel                    | BSel                    | PCSel                   | WBSel                   | RegWEn                  |
|----------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| auipc a1 0     | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 |
|                | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 |
|                | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * |
| addi a0 x0 3   | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 |
|                | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 |
|                | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * |
| srai t0 a0 2   | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 |
|                | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 |
|                | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * |
| bne x0 t0 done | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 | <input type="radio"/> 0 |
|                | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 | <input type="radio"/> 1 |
|                | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * | <input type="radio"/> * |

Q4.5 (2 points) For the code snippet on the previous page, how many cycles can we save by adding these forwarding paths to the original 5-stage pipeline?

cycles

Q5



(17 points)

Elden is debugging his code and is analyzing how a cache behaves during the runtime of his program.

Q5.1 (2 points) Given a **16-bit address space** and a **128 B 4-way set associative cache** with a **block size of 8 B**, determine the number of bits used for the tag, index, and offset.

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

For Q5.2 – Q5.6, assume that we have a **direct-mapped** cache, and a **16-bit address space** with 10 bits for tag, 2 bits for index, and 4 bits for offset.

```

1 #define A_SIZE 16
2 #define B_SIZE 4
3
4 int main() {
5     uint32_t A[A_SIZE]; // Base address: 0xFF00
6     uint32_t B[B_SIZE]; // Base address: 0xFFB0
7
8     /*
9     Code Omitted - Array A and B are populated with values
10    */
11
12    for (int i = 0; i < A_SIZE; i++) {
13        B[i % 4] = A[i] + B[i % 4];
14    }
15 }

```

Q5.2 (4 points) Assume that the program has just completed the fourth ( $i = 3$ ) iteration of the for loop. The **current** contents of main memory and the cache are provided in the tables below:

#### Cache

| Index | Valid | Dirty | Tag   | +F | +E | +D | +C | +B | +A | +9 | +8 | +7 | +6 | +5 | +4 | +3 | +2 | +1 | +0 |
|-------|-------|-------|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0     | 1     | 0     | 0x3FC | 00 | 10 | 32 | 00 | CC | 90 | 82 | FE | 11 | 9B | 3F | AD | 39 | E4 | 1B | 67 |
| 1     | 0     | 0     | 0x155 | 93 | E1 | A4 | 2B | 7C | 56 | F1 | 0D | 21 | 48 | B9 | EE | 5A | 00 | 74 | 3C |
| 2     | 0     | 0     | 0x2A7 | DE | 31 | F9 | A4 | 81 | C2 | 50 | 3B | 4E | 7A | 16 | CF | 02 | 7D | AB | 91 |
| 3     | 1     | 1     | 0x3FE | 2E | 9A | D3 | F1 | 7B | D4 | 19 | 0E | A6 | F5 | 5C | 44 | 00 | 61 | 34 | 00 |

(Question 5 continued...)

### Memory

| Memory Address | +F | +E | +D | +C | +B | +A | +9 | +8 | +7 | +6 | +5 | +4 | +3 | +2 | +1 | +0 |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0xFF00         | 00 | 10 | 32 | 00 | CC | 90 | 82 | FE | 11 | 9B | 3F | AD | 39 | E4 | 1B | 67 |
| 0xFF10         | D8 | 3A | A1 | 95 | 0B | 57 | E3 | C4 | A1 | 4F | 22 | D9 | 00 | 19 | 20 | 00 |
| 0xFF20         | 6F | 01 | 32 | E9 | EB | 84 | 71 | C0 | 55 | 39 | D5 | A2 | 2C | 00 | B4 | 8A |
| 0xFF30         | 9A | 4C | E6 | DF | 01 | 38 | C2 | 21 | F7 | 7D | A8 | 0C | 5E | 91 | B7 | 3B |
| 0xFF40         | 8E | 4D | 73 | A9 | 51 | C2 | FF | 03 | A1 | C9 | 7D | DE | 0B | 62 | E4 | 81 |
| ...            |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xFFB0         | 00 | 72 | 13 | 00 | 7B | D4 | 19 | 0E | A6 | F5 | 5C | 44 | 00 | 56 | 23 | 00 |

In the next iteration ( $i = 4$ ), exactly three memory accesses occur in the following order:

1. Reading an integer from A
2. Reading an integer from B
3. Writing an integer to B

For each of these three accesses when  $i = 4$ , **indicate whether it results in a cache hit or miss, and specify the value read or written in hexadecimal.** Assume that both the cache and main memory are **little-endian**.

(a) Reading an integer from A

- Cache Hit  
 Cache Miss

0x

(b) Reading an integer from B

- Cache Hit  
 Cache Miss

0x

(c) Writing an integer to B

- Cache Hit  
 Cache Miss

0x

Joanne wants to evaluate the cache's efficiency across the entire for loop (from  $i = 0$  to  $i = 15$ ). For Q4.3 – Q4.6, assume that the **cache starts cold** at the **beginning** of the loop (i.e. all valid bits are initially 0), and that arrays A and B begin at the same base addresses given above.

Q5.3 (1 point) How many misses occur in the first iteration of the for loop,  $i = 0$ ?

(Question 5 continued...)

Q5.4 (1 point) How many misses occur in the second iteration of the for loop,  $i = 1$ ?

Q5.5 (1 point) How many misses occur in the thirteenth iteration of the for loop,  $i = 12$ ?

Q5.6 (3 points) Across the entire for loop, what is the overall hit rate of the cache? *Express your answer as a single reduced fraction*

Nathan believes the previous cache design can be optimized. For Q4.7, assume the cache is now **2-way set associative**, with a size of 64 B and block size of 16 B. The cache uses the **LRU replacement policy**, and **starts off cold**.

Q5.7 (3 points) Across the entire for loop, what is the new overall hit rate of the cache? *Express your answer as a single reduced fraction*

Rachel is evaluating the performance of a new cache system with the following characteristics:

- Cache access time: **1 cycle**
- Main memory access time: **20 cycles**
- Cache hit rates
  - For read accesses, the hit rate is **0.80**
  - For write accesses, the hit rate is **0.60**
- **25%** of all memory accesses are writes

Q5.8 (2 points) On average, how many cycles does each memory access take? *Hint: Use a weighted average over reads and writes.*

Q6



(10 points)

Oh no! Andrew's cat ate Chloe's cheat sheet! Your task is to help remake Chloe's cheat sheet by filling in information about virtual memory.

Q6.1 (1 point) The page table is indexed by:

- PPN   
 VPN   
 Tag   
 Index   
 Offset   
 PA

Q6.2 (1 point) The TLB is updated with data from:

- Memory   
 Cache   
 TLB   
 Page Table

Q6.3 (1 point) The PA is formed by concatenating

and

Q6.4 (1 point) Recall that a cache has tag (T), index (I), and offset bits (O). Which of these has width equivalent to the width of (T + I + O)?

- PPN   
 PA   
 VPN   
 VA   
 TLB

For Q6.5 – Q6.7, you are given a single-level page table with the following properties:

- Virtual address (VA) width = 12 bits
- Page size = 16 B
- Page table entries (PTEs) formatted as follows:

|       |       |        |     |   |
|-------|-------|--------|-----|---|
| 15    | 14    | 13     | 12  | 0 |
| Valid | Dirty | Status | PPN |   |

Q6.5 (2 points) What is the size of a fully-populated page table in bytes, given this setup? You may express your answer in terms of powers of 2.

bytes

Q6.6 (2 points) Break the address 0x61C into its VPN and offset bits respectively.

VPN:       Offset:

Q6.7 (2 points) How much physical RAM does the system have? Express your answer in IEC form (e.g., 64 GiB)

## Q7 CASHE

(5 points)

Help Elva 61Cashout at Chippy Bank!

Q7.1 (3 points) Elva executes the following function `withdraw_money` with **two** threads to withdraw money from her Chippy Bank account.

```
1 void withdraw_money() {
2     int balance = 100;
3     #pragma omp parallel
4     {
5         int thread_id = omp_get_thread_num();
6         int amount = 40 + thread_id * 30;
7         if (balance >= amount) {
8             balance -= amount;
9             printf("Withdrew %d\n", amount);
10        }
11    }
12    printf("Final balance: %d", balance);
13 }
```

Select all of the following outputs that are possible:

Withdrew 40  
Final balance: 60

Withdrew 70  
Final balance: 30

Withdrew 40  
Withdrew 70  
Final balance: -10

Withdrew 40  
Withdrew 70  
Final balance: 60

Withdrew 40  
Withdrew 70  
Final balance: 30

(Question 7 continued...)

Q7.2 (1 point) On which line would you add a `#pragma omp critical` directive to ensure that `withdraw_money` never results in a negative balance, and the final balance accurately reflects how much money Elva withdrew (i.e: if Elva attempts to withdraw 50 dollars, the balance reduces by exactly 50 dollars).

- Between line 6 and 7
- Between line 7 and 8
- Between line 10 and 11
- No critical section needed

Q7.3 (1 point) There is only one ATM machine available. Both Elva and Alex happen to arrive at the same time to withdraw money from a shared bank account that has an initial balance of \$100. Alex executes Procedure A and Elva executes Procedure B.

Procedure A (Alex):

1. Acquire `ATM_Lock`
2. Acquire `Account_Lock`
3. Withdraw \$10 from `Account`
4. Release `Account_Lock`
5. Release `ATM_Lock`

Procedure B (Elva):

1. Acquire `Account_Lock`
2. Acquire `ATM_Lock`
3. Withdraw \$50 from `Account`
4. Release `ATM_Lock`
5. Release `Account_Lock`

Select all of the following outcomes that are possible:

- Only Alex can finish withdrawal of \$10
- Only Elva can finish withdrawal of \$50
- Alex finishes withdrawal of \$10 and Elva finishes withdrawal of \$50
- Neither Alex nor Elva can withdraw

You just got

Q8



(7 points)

(7 points) Complete the function `sum_simd` using SIMD vector instructions.

|   |                               |  |
|---|-------------------------------|--|
| <code>sum_simd</code> : Computes the sum of values in an array that are greater than or equal to 128. The function must use data-level parallelism (SIMD) to perform the calculation efficiently. |                               |  |
| Arguments   | <code>uint32_t *vals</code>   | An array of 32-bit unsigned integers.  |
|   | <code>size_t num_elems</code> | The number of elements. Guaranteed to be a multiple of 4.  |
| Return value  | <code>uint32_t</code>         | The sum of all values in the array that are greater than or equal to 128. You may assume that the sum will not overflow. |

You have access to the following SIMD operations. A single vector is a 128-bit vector register capable of holding four 32-bit integers. **You may use at most one operation per blank.**

a these

- `vector vec_load(uint32_t *A)`: Loads 4 integers at memory address `A` into a vector.
- `vector vec_setnum(uint32_t num)`: Creates a vector where every element is `num`.
- `vector vec_cmpgt(vector A, vector B)`: Computes `A > B` (with unsigned numbers) elementwise. Result is `0xFFFFFFFF` if true, 0 otherwise.
- `vector vec_and(vector A, vector B)`: Computes bitwise AND elementwise.
- `vector vec_add(vector A, vector B)`: Computes `A + B` elementwise.
- `uint32_t vec_sum(vector A)`: Adds all elements of the vector and returns the scalar sum.

```
1 uint32_t sum_simd(uint32_t *vals, size_t num_elems) {
2     vector _127 = vec_setnum(127);
3     uint32_t result = 0;
4
5     vector sum = _____;
6
7     for (unsigned int i = 0; _____; i += 4) {
8
9         vector curr = vec_load(_____);
10
11        vector mask = _____;
12
13        curr = _____;
14
15        _____;
16    }
17
18    _____;
19
20    return result;
21 }
```

Q9

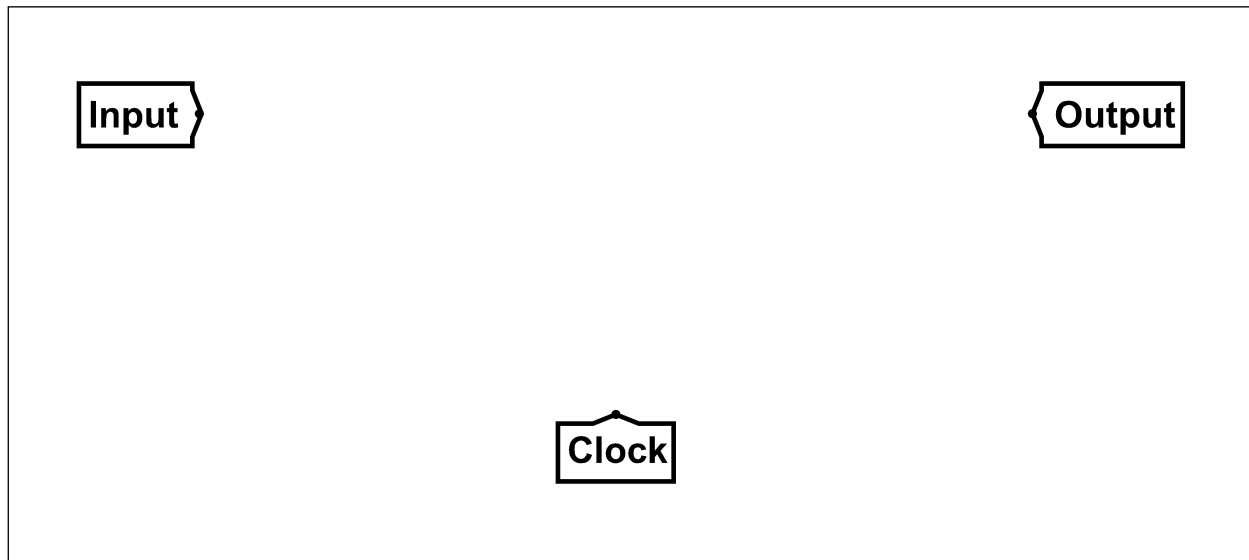


(5 points)

Create a circuit *using the fewest gates and registers* that can **flip the output** on and off with an output delay of one cycle. That is, every time the input is 1, the output changes state one cycle later.

Assume the input starts with a string of 0s and the output starts at 0.

|                |                        |
|----------------|------------------------|
| Example Input  | 0010001000110011100000 |
| Example Output | 0001111000010001011111 |



**ENTRANCE**  
*To*  
**MYSTERY SPOT**  
SANTA CRUZ, CALIF - U.S.A.

Q10

(8 points)

A C function `mystery(int a0, int a1, int a2)` compiles down to the following RISC-V code for a 32-bit machine and follows proper calling convention (i.e the return value should be left in `a0`).

```
1 mystery: beq a2 x0 loop
2           jal t0 next
3   next: lw t1 20(t0)
4           slli t2 a2 12
5           or t1 t1 t2
6           sw t1 20(t0)
7   loop: beq a1 x0 done
8           add a0 a0 a1
9           addi a1 a1 -1
10          j loop
11 done: ret
```

For Q10.1 – Q10.3, calculate the return value (in hexadecimal) for each call to `mystery`. Each call to `mystery` is independent.

Q10.1 (2 points) `mystery(1, 4, 0)`

0x

Q10.2 (2 points) `mystery(1, 4, 1)`

0x

Q10.3 (2 points) `mystery(1, 4, 4)`

0x

For Q10.4, indicate what input to `mystery` would result in the return value as shown.

Q10.4 (2 points) `mystery(0xB0BACAFE, 4, _____) → 2` // Hint: it's less than 256

This page is left intentionally (mostly) blank.

Please do not tear off any pages from the exam.

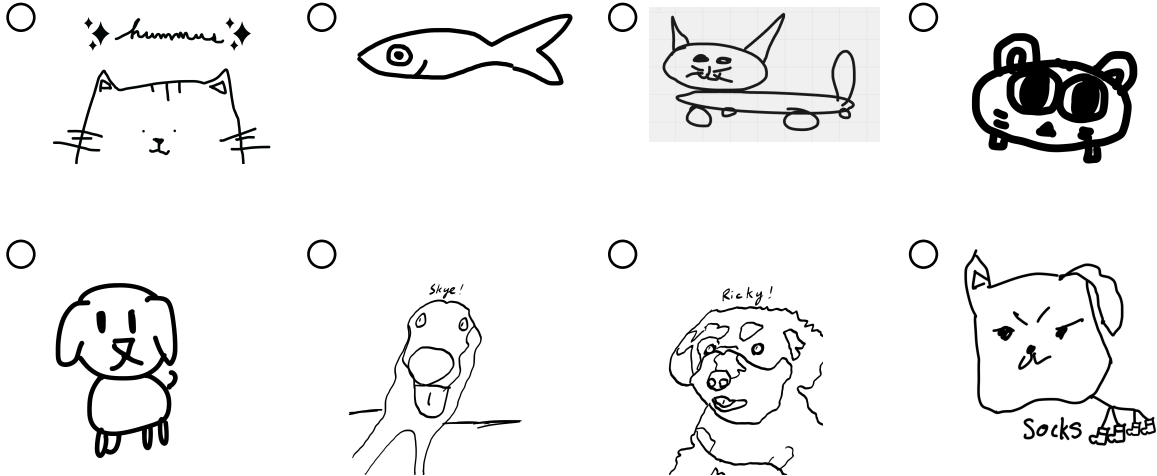
**Make sure you have completed all 10 questions on this exam.**

Q11



(0 points)

Q11.1 Favorite pet from the below options?



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.