CS 61C Summer 2026 — Yokota Midterm

Solutions last updated: Monday, July 27, 2026

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 110 minutes. There are 8 questions of varying credit. (100 points total)

Question:	1	2	3	4	5	6	7	8	Total
Points:	20	16	7	17	13	8	19	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:

_____

This content is protected and may not be shared, uploaded, or distributed.

Q1 *Potpourri* **(20 points)**

Q1.1 (1.5 points) Convert −134 to its 16-bit 2's complement hexadecimal representation. If your answer has leading zeros, make sure to include them.

0xFF7A

Q1.2 (1 point) You are told that adding 2 two's complement numbers, A and B, produced a result where overflow did occur. Which of the following must be true?

- ○ A and B **must** have different sign bits.
- ● A and B **must** have the same sign bit.
- ○ Neither of the above.

For the following two subparts, select the step of CALL that performs the operation.

Q1.3 (1 point) Translates assembly instructions into machine code, resolving local labels but leaving external references unresolved.

○ Compiler ● Assembler ○ Linker ○ Loader

Q1.4 (1 point) Copies the executable into memory and sets up the program's address space.

○ Compiler ○ Assembler ○ Linker ● Loader

For the following two subparts, assume the following C code is run on a 32-bit system.

```
1  int main() {
2      uint32_t x[] = {
3          0x61A76767,
4          0x0061B0BA,
5          0xC500C61C
6      };
7     uint32_t *p = x;
8     printf("%d, ", strlen((char *)p));
9     p++;
10    printf("%d", strlen(((char *)p) + 1));
11 }
```

Q1.5 (2 points) What is printed if our system is **little-endian**?

7, 2

Q1.6 (2 points) What is printed if our system is **big-endian**?

4, 4

This content is protected and may not be shared, uploaded, or distributed.

Q1.7 (2.5 points) Suppose we are using an 8-bit IEEE-754 standard floating point format with 1 sign bit, 4 exponent bits (with a bias of −7), and 3 mantissa bits. Convert `0x84` to decimal. You may express your answer as a power of two.

-2^{-7}

Q1.8 (3 points) Suppose we are using the standard IEEE-754 **Single Precision** floating point format. What is the smallest positive integer `n` such that 2^{30} + `n` is *NOT* representable but 2^{30} + `2n` *IS* representable?

2^{6}

Q1.9 (3 points) Translate the RISC-V instruction on Line 7 to 32-bit hexadecimal machine code.

```
1  li a0 10
2  li a1 22
3  loop:
4     beq a0 x0 end
5     addi a1 a1 1
6     addi a0 a0 -1
7     j loop     # Translate this line
8  end:
9     jr ra
```

0xFF5FF06F

Q1.10 (3 points) Select the minimum set of registers that we need to save/restore from the stack such that function `Toy` will satisfy RISC-V register calling conventions.

```
1  Toy:
2    addi s0 s1 5
3    addi a0 a0 2
4    addi t1 a1 2
5
6    jal ra Story
7
8    mv a0 t1
9    jr ra
10
11 Story:
12   ...  # Implementation omitted
```

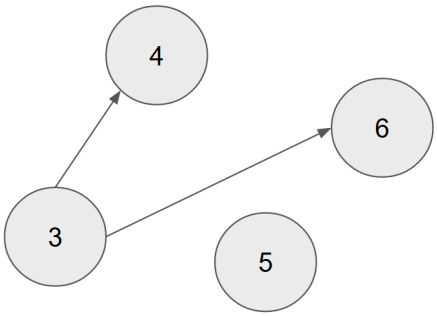
■ s0 ☐ a0 ■ t1 ○ None of the above

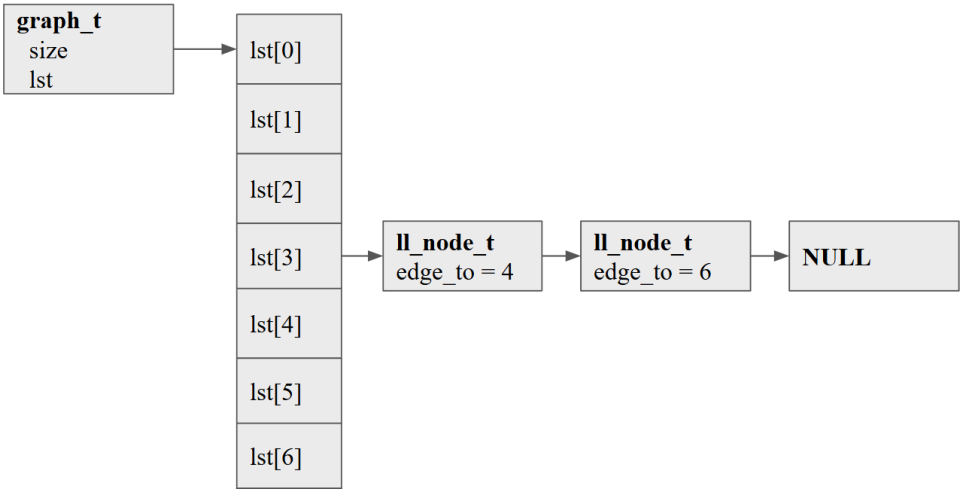
☐ s1 ☐ a1 ■ ra

This content is protected and may not be shared, uploaded, or distributed.

Q2 *Making Connections* **(16 points)**

An adjacency list is an array used to represent a directed graph. Each index in the array identifies a vertex in the graph and stores a `NULL` terminated linked list of neighboring vertices. For example, if a graph contains a directed edge from vertex 3 to vertex 4 and a directed edge from vertex 3 to vertex 6, index 3 of the array will contain a linked list of nodes corresponding to vertex 4 and vertex 6.






```
1  typedef struct ll_node_t {
2    // the destination vertex of the edge that this node represents
3    uint16_t edge_to;
4    // pointer to the next ll_node_t in linked list of neighbors
5    struct ll_node_t* next;
6  } ll_node_t;
7
8  typedef struct graph_t {
9    // the length of adjacency_list
10   uint16_t size;
11   // adjaceny list of nodes in the graph
12   ll_node_t** lst;
13 } graph_t;
```


This content is protected and may not be shared, uploaded, or distributed.

Q2.1 (4 points) The function `create_graph` initializes an empty `graph_t` with no nodes and edges.

```
1 graph_t* create_graph() { return <INSERT CODE HERE>; }
```

Which of the following lines will initialize the graph as specified, assuming that we are working on a **32-bit** system? Select all that apply.

- ☐ `NULL`
- ☐ `calloc(6, 1)`
- ■ `calloc(sizeof(graph_t), 1)`
- ☐ `malloc(6)`
- ■ `calloc(8, 1)`
- ☐ `&((graph_t) {0, NULL})`
- ☐ `malloc(8)`
- ☐ `malloc(sizeof(graph_t))`
- ○ None of the above

(12 points) Implement `add_edge`, which adds a new directed edge from a source vertex to a destination vertex. If either `src` or `dest` is larger than the current graph's `size`, `graph` should be expanded to accommodate the node with the highest id. You may assume that all memory allocations succeed.

`add_edge`: Add an edge in `graph` from vertex `src` to vertex `dest`		
Arguments	`graph_t *graph`	The graph to insert this new edge
	`uint16_t src`	The source vertex of the edge
	`uint16_t dest`	The destination vertex of the edge
Return value	`void`	No return value

```
1  void add_edge(graph_t* graph, uint16_t src, uint16_t dest) {
2      int max_id = src < dest ? dest : src;
3      if (max_id >= graph->size) {                                  // Q2.2
4          graph->lst = realloc(graph->lst, sizeof(ll_node_t*) * (max_id + 1));   // Q2.3
5          for (int i = graph->size; i <= max_id; i++) {
6              graph->lst[i] = NULL;                                 // Q2.4
7          }
8          graph->size = max_id + 1;                                 // Q2.5
9      }
10     ll_node_t* new_node = malloc(sizeof(ll_node_t));              // Q2.6, Q2.7
11     new_node->edge_to = dest;                                     // Q2.8
12     new_node->next = graph->lst[src];                             // Q2.9
13     graph->lst[src] = new_node;                                   // Q2.10
14 }
```


This content is protected and may not be shared, uploaded, or distributed.

Q3 *Reduce* **(7 points)**

For the following subparts, write your answer as a Boolean expression using the operators and symbols from the table below.

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

Q3.1 (3.5 points) Simplify the following expression. For full credit, you must use as few operators as possible. Partial credit will only be given to equivalent expressions with a suboptimal number of operators.

```
(!(A * C) + B * A) * B + !(A * B)
```

Output = `1`

Q3.2 (3.5 points) Simplify the following truth table:

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

For full credit, you must use as few operators as possible. Partial credit will only be given to equivalent expressions with a suboptimal number of operators.

Output = `!B + C`

This content is protected and may not be shared, uploaded, or distributed.

Q4 *61enCoding* **(17 points)**

Enoch is a manager at 61ChainMatcha, a startup that stores matcha quality sensor readings in an array before shipping them off to a (not very trustworthy) third-party logging service. He doesn't want to pay for real encryption, but he'd still like to encrypt the sensor readings in case of an eavesdropper when sending them to his manager.

Enoch assigns William to combine each pair of adjacent array elements into a scrambled **16 bit** value using a homemade bitwise operation, called `hashing`, which works as follows.

Given two 16-bit values `x` and `y`, `hashing` should do the following:

1. XOR `x` and `y` together
2. Then flip every other bit of the result.

For example:

```
x =             0b1011 0001 0000 0000
y =             0b0000 1111 0000 0000
x XOR y =       0b1011 1110 0000 0000 # XOR x and y together
hashing(x,y) = 0b0001 0100 1010 1010 # Flip every other bit of the result
```

(4 points) Write a RISC-V function that implements `hashing`. For full credit, you may only modify the `t0`, `a0`, and `a1` registers. You may use pseudoinstructions

`hashing`: Returns the result of running Enoch's bitwise operation on `a0` and `a1`		
Arguments	`a0`	A 16 bit integer
	`a1`	A 16 bit integer
Return value	`a0`	The result of Enoch's homemade bitwise operation

```
hashing:
    xor t0 a0 a1

    li a1 0xAAAA
        Q4.1

    xor a0 t0 a1
        Q4.2

    jr ra
     Q4.3
```


This content is protected and may not be shared, uploaded, or distributed.

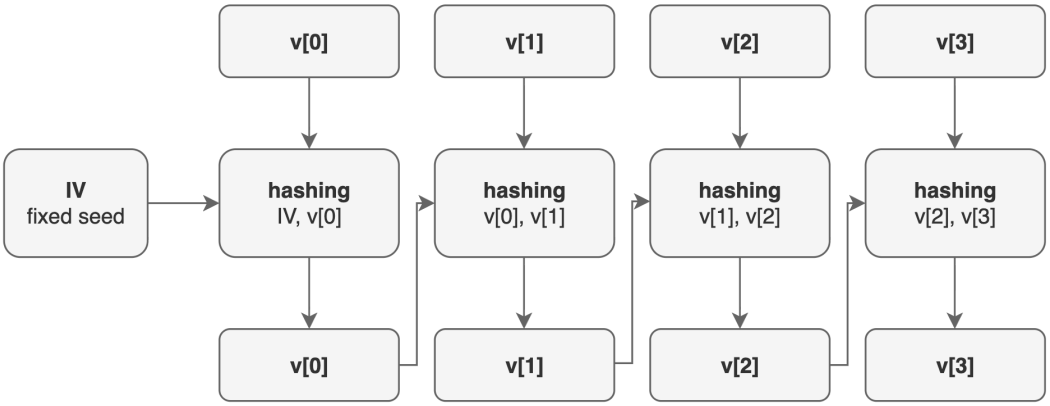
Now that William has hashing working, he wants to write a function to encode an array of sensor readings before sending it. William picks a fixed 16-bit constant initialization value, `IV`, known to both the sender and receiver ahead of time. You may refer to it as `IV` in your code.

Given an array `v` of `n` 16-bit sensor readings (`n` ≥ 1), `encode` should overwrite `v` in place so that:

```
1 v[0] = hashing(IV, v[0])
2 v[i] = hashing(v[i], v[i-1])     for i = 1, 2, ..., n-1
```

where `v[i-1]` refers to the already-encoded value, since the array is being modified in place.

Here is a diagram illustrating the chained encoding process:



(13 points) Complete `encode`, which modifies the input array in place with its encoded values. You may call `hashing` and assume that it works correctly regardless of your implementation above.

`encode`: Encodes the input array `v` pointed to by `a0` **in-place**, using the initialization constant `IV`		
Arguments	`a0`	A pointer to a `uint16_t` array `v`
	`a1`	The number of elements in `v`
	`a2`	The initialization constant `IV`
Return value	`a0`	A pointer to `v`, after overwriting its data with its encoded form

This content is protected and may not be shared, uploaded, or distributed.

(Question 4 continued...)

```
1  encode:
2      # prologue
3      addi sp sp -16          # Q4.4
4      sw s0 0(sp)             # Q4.5
5      sw s1 4(sp)             # Q4.6
6      sw s2 8(sp)             # Q4.7
7      sw ra 12(sp)            # Q4.8
8      mv s0 a0 # Save a copy of a0 to return later
9      mv s1 a0
10     mv s2 a1
11     mv a0 a2                # Q4.9
12 loop:
13     beq x0 s2 end           # Q4.10
14     lh a1 0(s1)             # Q4.11
15     jal ra hashing          # Q4.12
16     sh a0 0(s1)             # Q4.13
17     addi s1 s1   2          # Q4.14
18     addi s2 s2 -1           # Q4.15
19     j loop
20 end:
21     mv a0 s0                # Q4.16
22
23     # epilogue ommitted, you may assume it correctly mirrors your prologue
24
25     jr ra
```


This content is protected and may not be shared, uploaded, or distributed.

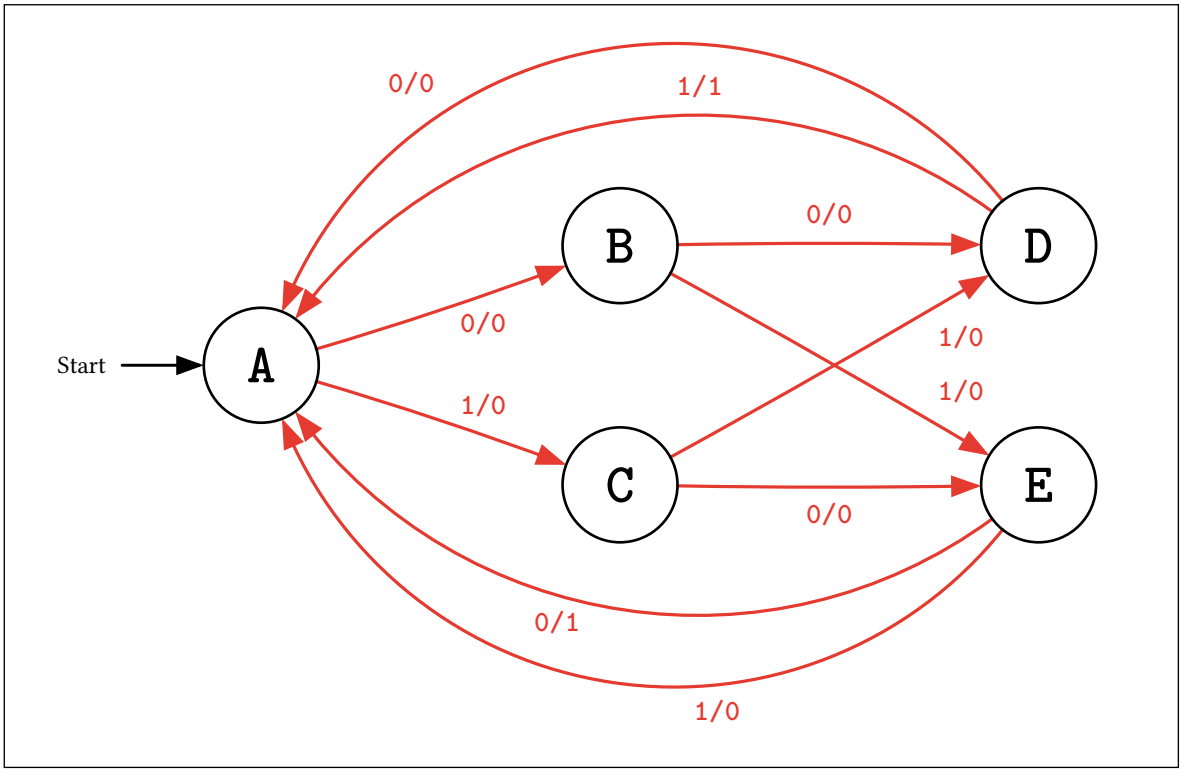
Q5 *Matcha Cafe* **(13 points)**

Adelina just opened up a matcha cafe! She wants to ensure her drinks are high-quality, she checks each batch with **3 quality sensors**. Because **someone** decided to encrypt their sensor data for some reason, a high-quality batch reading should contain an **odd** number of 1s across the 3 sensors. Adelina creates an FSM to read sensor data and output whether or not her matcha is high-quality. The sensor readings are sent into Adelina's FSM, one at a time, with an output denoting the result of the test every third input.

In	`0`	Sensor does not pass
	`1`	Sensor passes
Out	`0`	Batch has no approval yet/failed
	`1`	Batch approved

Example `Input`	`111100011`
Example `Out`	`001001000`

Q5.1 (10 points) Implement the following FSM given these parameters.



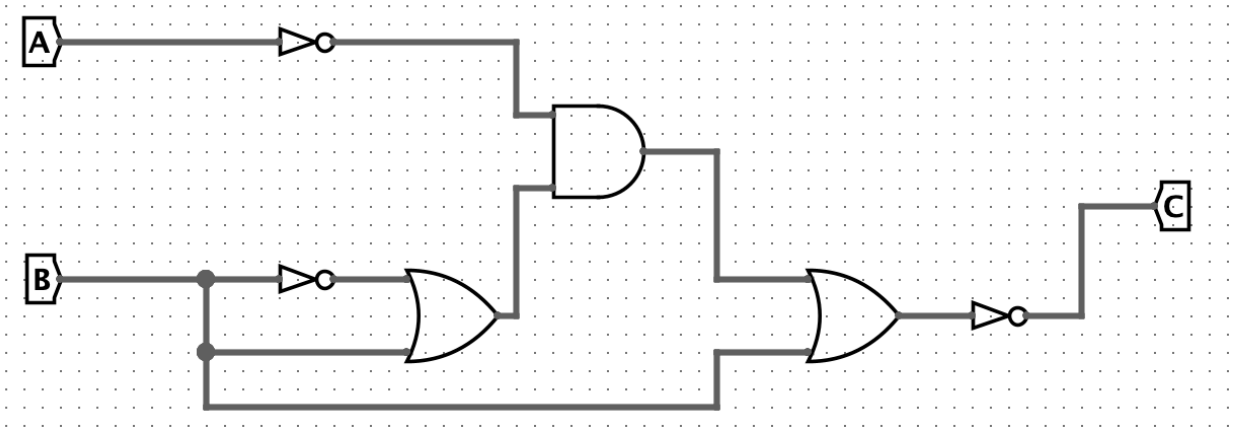
Q5.2 (3 points) Suppose each batch now contains x sensor readings instead of 3. What is the minimum number of states Adelina's FSM would need? Express your answer in terms of x.

$2x$ - 1

This content is protected and may not be shared, uploaded, or distributed.

Q6 *ReCycle* **(8 points)**

For Q6.1 — Q6.4, consider the following circuit and timings, given that `A`, `B`, and `C` are all directly connected to registers.



$t_{\texttt{setup}} = 20$ ns $\quad t_{\texttt{clk-to-q}} = 38$ ns $t_{\texttt{hold}} = 15$ ns $\quad t_{\texttt{NOT}} = 5$ ns $\quad t_{\texttt{AND}} = 8$ ns $\quad t_{\texttt{OR}} = 12$ ns $\quad t_{\texttt{XOR}} = 10$ ns

Q6.1 (2 points) What is the longest combinational logic delay?

42ns

Solution: $R_{\texttt{A}} \rightarrow R_{\texttt{B}}$: NOT ⟶ OR ⟶ AND ⟶ OR ⟶ NOT = 5 + 12 + 8 + 12 + 5 = 42ns

Q6.2 (2 points) What is the minimum allowable clock period?

100ns

Solution: $T_{\texttt{min}} = t_{\texttt{clk-to-q}} + t_{\texttt{CL,max}} + t_{\texttt{setup}} = 38 + 42 + 20 = 100$ ns

Q6.3 (2 points) What is the maximum allowable clock frequency? For credit your answer must include the correct units of measurement.

10 MHz

Solution: $\frac{1}{T_{\texttt{min}}} = \frac{1}{100\text{ns}} = 10$ MHz

This content is protected and may not be shared, uploaded, or distributed.

Q6.4 (2 points) What is the maximum allowable hold time?

55ns

Solution: $T_{\texttt{hold}} = t_{\texttt{clk-to-q}} + t_{\texttt{CL,min}} = 38 + 17 = 55$ ns

This content is protected and may not be shared, uploaded, or distributed.

Q7 *Pop(p)i* **(19 points)**

We want to implement a new RISC-V instruction `popi rd imm(rs1)` to our RISC-V single-cycle datapath that does the following:

```
1 R[rd]  = Mem[R[rs1] + imm];
2 R[rs1] = R[rs1] + 4;
```

The two above lines happen simultaneously. If `rd` == `rs1` then `R[rs1]` is set to `Mem[R[rs1] + imm]`.

Q7.1 (4 points) Alex proposes replacing `popi rd imm(rs1)` with the following two RISC-V instructions:

```
1 lw rd imm(rs1)
2 addi rs1 rs1 4
```

Provide a program of at most 5 RISC-V instructions that produces different behavior when `popi` is a real hardware instruction compared to when it gets expanded into the two RISC-V instructions Alex proposed above. If no such program exists, write "No Solution".

```
popi sp 0(sp)
```
Alternatively:
```
beq x0 x0 8
popi t0 0(sp)
addi a0 a0 1
addi a0 a0 1
addi a0 a0 1
```

Q7.2 (1 point) What existing RISC-V instruction format would be the most suitable starting point for encoding `popi`?

○ R ○ S ○ U ○ No existing instruction type works cleanly
● I ○ B ○ J

Q7.3 (1 point) Can the standard RISC-V single-cycle datapath (as defined on the Reference Card) execute `popi` in one cycle without any hardware modifications?

○ Yes, because every instruction in a single-cycle CPU finishes in one cycle.
○ Yes, because the datapaths for `lw` and `addi` already exist.
● No, because the ALU is already used to compute `rs1 + imm`.
○ No, because the register file cannot write to `rd`.

This content is protected and may not be shared, uploaded, or distributed.

Q7.4 (5 points) What additional changes, if any, would we need to make to our single-cycle datapath in order for us to implement **popi** (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, along with a corresponding **RegWEn2** and internal logic
- ☐ 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 hardware adder
- ☐ Add a fourth possible value for **WBSel** and update the corresponding MUX/control logic
- ☐ Add a new read input to **DMEM** for a second memory read output
- ☐ Add a new Branch Comparator path
- ○ None of the above

Q7.5 (8 points) Suppose the datapath has been correctly modified to support **popi**. For each of the following control signals, indicate the value it should have for **popi**. If the value of the signal does not matter, select *****.

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**	○ *****
ALUSel	● Add		○ XOR		○ Depends on **BrEQ** and **BrLT**	○ *****
MemRW	● Read		○ Write		○ Depends on **BrEQ** and **BrLT**	○ *****
RegWEn	○ 0		● 1		○ Depends on **BrEQ** and **BrLT**	○ *****
ImmSel	● I	○ B	○ S		○ Depends on **BrEQ** and **BrLT**	○ *****
WBSel	○ ALU	● Mem	○ **PC + 4**		○ Depends on **BrEQ** and **BrLT**	○ *****

This content is protected and may not be shared, uploaded, or distributed.

Q8 *To Infinity and Beyond* **(0 points)**

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

Q8.1 Best Toy Story?

◯ Toy Story ◯ Toy Story 2 ◯ Toy Story 3 ◯ Toy Story 4 ◯ Toy Story 5

Q8.2 Which one did you like more?

◯ Inside Out ◯ Inside Out 2

Q8.3 Favorite one from below options?

◯ Coco ◯ Cars ◯ Monster's University ◯ Wall-E

◯ Hoppers ◯ Cars 2 ◯ Brave ◯ Up

◯ Toy Story 3 ◯ Cars 3 ◯ The Incredibles ◯ Ratatouille

Q8.4 What's your favorite Pixar movie?

Q8.5 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.

This content is protected and may not be shared, uploaded, or distributed.