

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `cs61c@berkeley.edu`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- You must choose either this option
- Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- You could select this choice.
- You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

Please complete and submit these questions before the exam starts.

- (a) What is your full name?

- (b) What is your student ID number?

- (c) If an answer requires hex input, make sure you only use capitalized letters! For example, 0xDEADBEEF instead of 0xdeadbeef. You will be graded incorrectly otherwise! Please always add the hex (0x) and binary (0b) prefix to your answers or you will receive 0 points. For all other bases, do not add the suffix or prefixes.

Do not add units unless the problem explicitly tells you to!

Some of the questions may use images to describe a problem. If the image is too small, you can click and drag the image to a new tab to see the full image. You can also right click the image and download it or copy its address to view it better. You can use the image below to try this. You can also click the star by the question if you would like to go back to it (it will show up on the side bar). In addition, you are able see a check mark for questions you have fully entered in the sidebar. Questions will auto submit about 5 seconds after you click off of them, though we still recommend you click the save button.



Good luck!

1. POTPOURRI

(a) Q1

- i.** You have a program that spends some percentage of its time waiting for requests and the rest of the time performing calculations. Suppose you have 8 threads which you can use to parallelize calculations, with no overhead or non-parallelizable calculations. What is the maximum fraction of time that your sequential program can spend on waiting for requests if we would like to achieve at least 4 times speedup? Leave your answer as a single simplified fraction.

(b) Q2

The company you work at has a datacenter. Answer the following questions:

i. Q2A

- A.** The Mean Time Between Failures (MTBF) for this particular data center is 4000 hours. The Mean Time To Repair (MTTR) is 3 hours. What is the availability for this datacenter? Express your answer as a single simplified fraction.

ii. Q2B

- A.** Your company would like to restrict the annualized failure rate to be 1% for the individual machines in a large cluster. What does the Mean Time To Failure (MTTF) have to be to satisfy this annualized failure rate? Assume that the MTTF in this question is unrelated to that of part a. Write down your answer in years.

(c) Q3

For each of the following functionalities, select the stage(s) in CALL for which the statement is true.

i. Q3V2

A. Replaces pseudo instructions.

- Compiler
- Assembler
- Linker
- Loader

B. Calculate all absolute addresses.

- Compiler
- Assembler
- Linker
- Loader

C. Generates parse trees.

- Compiler
- Assembler
- Linker
- Loader

D. Constructs symbol table.

- Compiler
- Assembler
- Linker
- Loader

(d) Q4

Which of the following code blocks would see a performance improvement if we placed a `#pragma omp for` over the outer for loop? Select all that apply.

i. A

```
for (int i = 0; i < 5000 - 3; i += 3) {  
    a[i+2] = a[i] + a[i+1];  
}
```

B

```
for (int i = 0; i < 5000 - 2; i += 2) {  
    a[i+2] *= a[i];  
}
```

C

```
for (int i = 0; i < 5000 - 3; i++) {  
    a[i+2] = a[i] + a[i+1];  
}
```

D

```
for (int i = 0; i < 5000; i++) {  
    a[i] = 100;  
}
```

A

B

C

D

E. None of the above

2. Virtual Memory Caching

(a) TIO

Assume all systems are 32-bit.

- i. We're given a system with an 4-way set associative cache of size 512 KiB with 128 B blocks. How many bits are allocated to the tag, index, and offset bits respectively?

A. Tag:

B. Index:

C. Offset:

D. Code Analysis

We run the following code with our caching setup unchanged from the previous question. What is the hit rate of the following lines of code?

```
E. 1 #define base_arr_addr 0x12345678
   2 #define ARR_SIZE 4096
   3 #define I_BOUNDARY 2048
   4 #define J_BOUNDARY 4096
   5 #define I_STRIDE 128
   6 #define J_STRIDE 64
   7 int arr[ARR_SIZE];
   8
   9 uint32_t dummy_func(void) {
  10     //Loop 1
  11     for (int i = 0; i < I_BOUNDARY; i += I_STRIDE) {
  12         for (int j = 0; j < J_BOUNDARY; j += J_STRIDE) {
  13             arr[i] = arr[i] + arr[j];
  14         }
  15     }
  16
  17     //Loop 3
  18     for (int j = 0; j < J_BOUNDARY; j += J_STRIDE) {
  19         for (int i = 0; i < I_BOUNDARY; i += I_STRIDE) {
  20             arr[j] = arr[j] * arr[i];
  21         }
  22     }
  23 }
```


F. `arr[i] = arr[i] + arr[j]`

G. `arr[j] = arr[j] * arr[i]`

H. NOTE: for all parts, assume changes propagate unless otherwise stated.

As we're working on running the code snippet, we realise we want to run different instances of the same code. We choose to employ virtual memory on our memory space. We have 4 GiB of virtual memory and 16 MiB of physical memory mapped with a single level page table with a page size of 4 KiB. We choose to store 8 bits of metadata with each page table entry.

- I.** After running one iteration of the inner loop for the code given in line, how many physical pages will our page table take up?

- J.** If our caching system remains as seen in question 1, how many caches would be needed to fully fit our page table? Give your answer as a decimal to two decimal places.

- K.** We now switch our system to having a 2-level page table instead. Assuming we restart our system and run one iteration of the inner loop, and that the VPN bits are split evenly between levels, how many pages will our active page tables span?

(b) **NOTE:due to a staff error in versioning, everyone will receive full points for this question.**

- i. Assume now our system chooses to switch to another process. Which of the following events will occur. Select all that apply.
 - Save the page table to the stack.
 - The user triggers a timer interrupt.
 - Invalidate the TLB.
 - None of the above.

Given the two-level page table we have in Figure 1, translate the following virtual addresses to physical addresses. Assume all page tables shown are active. If an address does not point to an active page table at any point, write "Invalid Address".

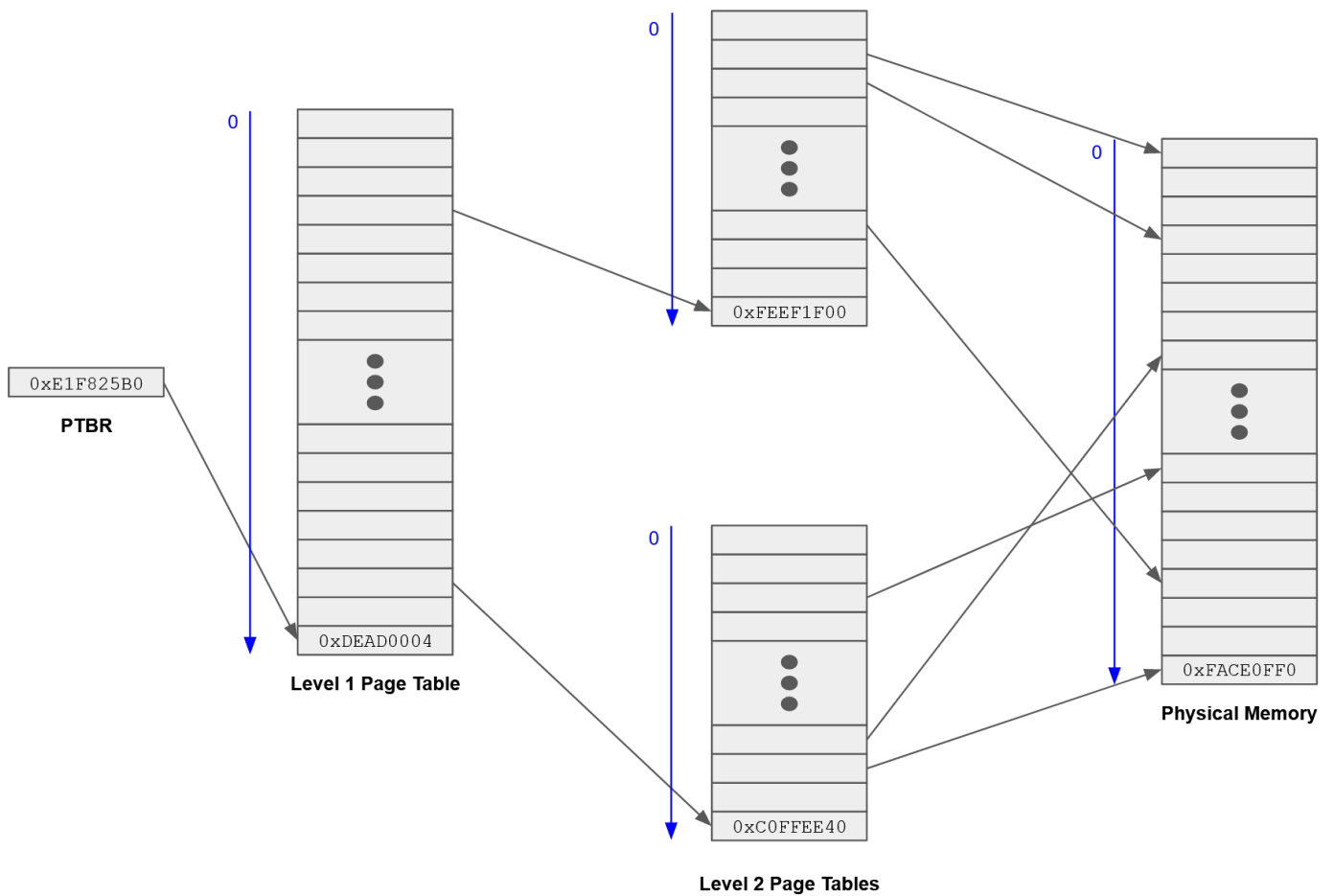


Figure 2

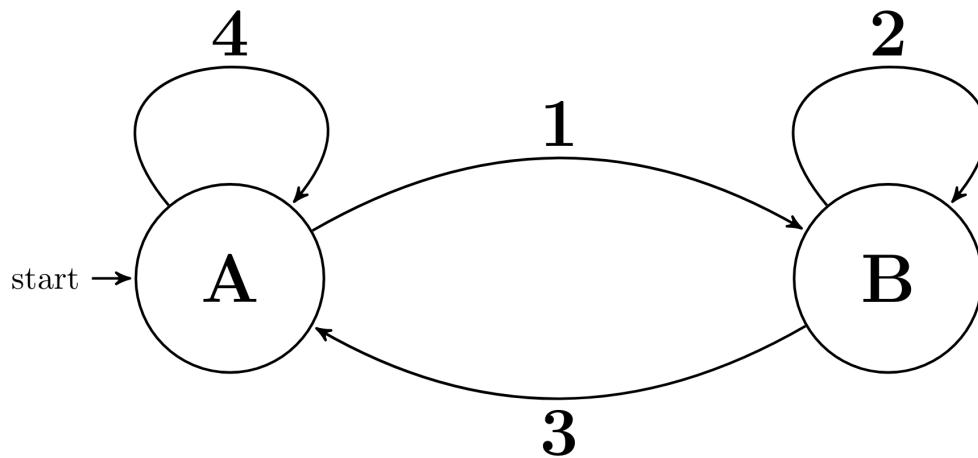
- ii. A. Address: 0xB7C90AEA

3. FSM

We want to construct a finite-state machine that determines if adding together two binary unsigned numbers causes an overflow. The machine consumes two bit strings of equal length, starting from their least significant bits. After consuming each pair of bits from the two inputs, the machine outputs 1 if addition of the two strings (as seen so far) would cause an overflow, or a 0 otherwise.

For example, if the machine consumes the two bit strings 0111 and 0100, the sequence of output values will be 0, 0, 1, 0.

A diagram for this finite-state machine is shown below. We have given the states generic placeholder names. Transition labels use the notation $x,y/o$, where x is a bit read from the first input string, y is a bit read from the second input string, and o is the output. To simplify, you are allowed to assign multiple labels to the same arrow in the diagram – each label corresponds to a different transition with the same start and end states.



State Machine

- (a) i. (1.0 pt) Which of the following labels (each corresponding to one transition) should be assigned to **Arrow 1** in the diagram?

- 1,1/0
- 0,0/0
- 0,1/0
- 1,0/0
- 1,0/1
- 0,1/1
- 0,0/1
- 1,1/1

ii. (1.0 pt) Which of the following labels (each corresponding to one transition) should be assigned to **Arrow 2** in the diagram?

- 1,1/1
- 1,0/0
- 0,1/0
- 0,0/1
- 0,1/1
- 1,0/1
- 0,0/0
- 1,1/0

iii. (1.0 pt) Which of the following labels (each corresponding to one transition) should be assigned to **Arrow 3** in the diagram?

- 1,1/1
- 0,1/1
- 0,0/0
- 1,0/0
- 1,1/0
- 0,1/0
- 1,0/1
- 0,0/1

iv. (1.0 pt) Which of the following labels (each corresponding to one transition) should be assigned to **Arrow 4** in the diagram?

- 0,1/0
- 1,1/1
- 1,0/1
- 0,0/0
- 0,0/1
- 1,1/0
- 0,1/1
- 1,0/0

(b) Select the different ways you can decrease the critical path (ignore wire delay)?

- Make the clock speed faster
- Move components closer together
- Simplify circuit (boolean) logic
- Add pipeline registers
- Use transistors instead of AND, OR, and NOT gates
- Use AND and OR gates instead of NAND and NOR gates

5. Boolean Algebra

(a) Select all of the following which are equivalent to $\overline{A} + \overline{BC}$.

$\overline{A(\overline{BC} + C\overline{B})} + \overline{CA}$

$\overline{C\overline{BA} + C(\overline{CA} + BC)}$

$A\overline{B} + \overline{C}$

$A\overline{A} + \overline{(ABC + \overline{CA})}$

6. C Structures**(a) (6.0 points) The Structure of Structures**

- i. Assuming a 32-bit architecture with RISC-V alignment rules:

Consider the following structure definition and code:

```
struct foo {  
    char a;  
    uint16_t b;  
    char *c;  
    struct foo *d;  
}
```

What is `sizeof(struct foo)` (Answer as an integer, with no units)?

- ii. If `b` and `c` are swapped, this increases the size of the structure:

- True
 False

(b) Skipping Around

Consider the following code

```

struct SLN{
    void *data;
    struct SLN **next;
}

/* Only the following in the code actually matters:
   cmp(find, sl->next[level]->data)
   But if you are curious, look up "Skiplist"
*/

int SL_find(void *find, int level, SkipListNode *sl,
            (int)(*cmp)(void *, void *)){
    if(cmp(find, sl->data) == 0) return 1;
    if(level == 0 && sl->next[level] == null) return 0;
    if(sl->next[level] == null) {
        return SL_find(find, level-1, sl, cmp);
    }
    if(cmp(find, sl->next[level]->data) > 0){
        return SL_find(find, level-1, sl, cmp);
    }
    return SL_find(find, level, sl->next[level], cmp);
}

```

In translating the code `cmp(find, sl->next[level]->data)` into RISC-V we need to store arguments on the stack. So we will have `find` at `sp(0)`, `level` at `sp(4)`, `sl` at `sp(8)` and `cmp` at `sp(12)`. We're going to break up the translation into pieces

- i. Load `find` into `a0`

- ii. The next thing we need to do is get `sl->next[level]->data` into `a1`.
The RISC-V code for that would be (fill in based on comments):

Load `sl` into `t0`

- iii. Load `level` into `t1`

- iv. load `sl->next` into `t0`

v. load `s1->next[level]` into `t0`

vi. load `data` into `a0`

vii. Finally, how do we call `cmp` (using `t0` as a temporary)

7. Datapath

(a) NEWINSTR

Given the standard RISC-V datapath (Figure 1), determine if the following is implementable or not without any additional functional units? Assume the instruction is not a pseudoinstruction encoding.

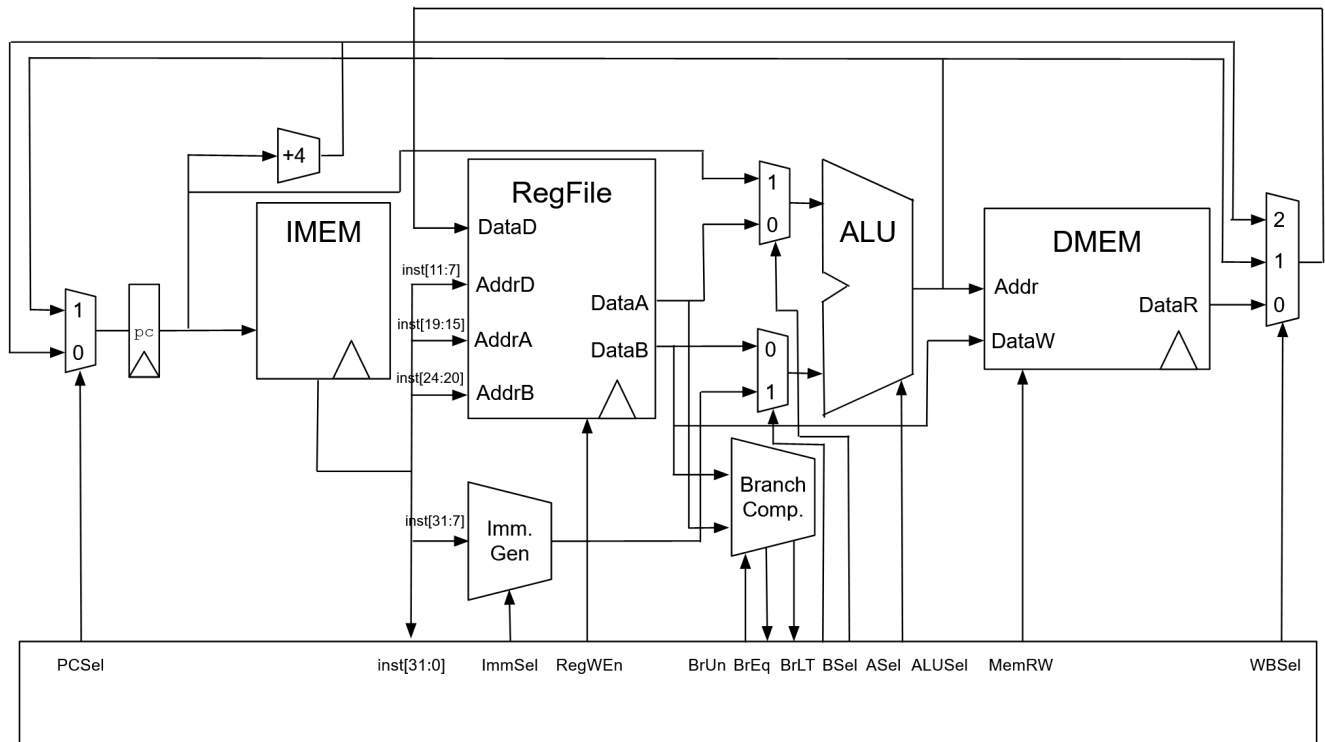


Figure 1

i. Is Null

A. `is_null rd, rs1`: check if an input given through `rs1` is considered NULL or not by C standard. The result is returned through `rd` as a bit.

- Implementable
- Not implementable

- B.** What changes would you need to make in order for the instruction to be able to execute correctly? Assume all modifications and additions are done on top of the existing single cycle datapath. Select all that apply
- Modify the control signals to the ALU.
 - Modify Branch Comparator logic.
 - Modify the control logic for `WBSel`.
 - Modify the control logic for parsing `instr[31:0]`.
 - Modify control logic for ALU/ALUSel.
 - Add an additional comparator.
 - Add additional control signals for the writeback mux.
 - Modify ALU buses.
 - Modify the control logic for the Branch Comparator.
 - None of the above.
- C.** `is_null rd, rs1` is not in a standard RISC-V instruction format; as we're attempting to reduce the number of hardware changes in our datapath. We instead choose to implement our instruction as a pseudoinstruction in the following format. Which of the following statements is true? Assume earlier changes propagate. Select all that apply.

Format: R-Type Instruction

- We need to wire `x0` as `rs2` and modify the control signals.
- We need to provide a second argument `x0` when calling the instruction and modify the control signals.
- We need to provide a second argument `x0` as a comparator for all branch comparisons.
- We need to wire `x0` as a comparator for all branch comparisons.
- It is impossible to represent as an R-Type instruction.

(b) Cached Datapath

We now choose to add caching to the single cycle datapath. We add a single level cache that is shared between instruction and data memory and is always accessed prior to a direct memory access.

As we observe our datapath, we see that having heavy caching in addition to our respective memory arrays does help our average access times in comparison to having no caching, as is the expected behaviour. However, in our worst-case scenarios, we see that the time taken is suboptimal. We instead propose an alternative method such that our memory arrays (IMEM and DMEM) are replaced by caches. We have one shared cache such that IMEM and DMEM accesses both go through Cache A. What modifications need to be made to the control logic signals? Give your answer in 10 words or fewer.

i.



- ii. How does missing in the cache affect how our datapath operates? Frame your answer in terms of how it affects the control signals and hazards in 2 or fewer sentences.



- iii. Which of the following is always true about our caching setup?

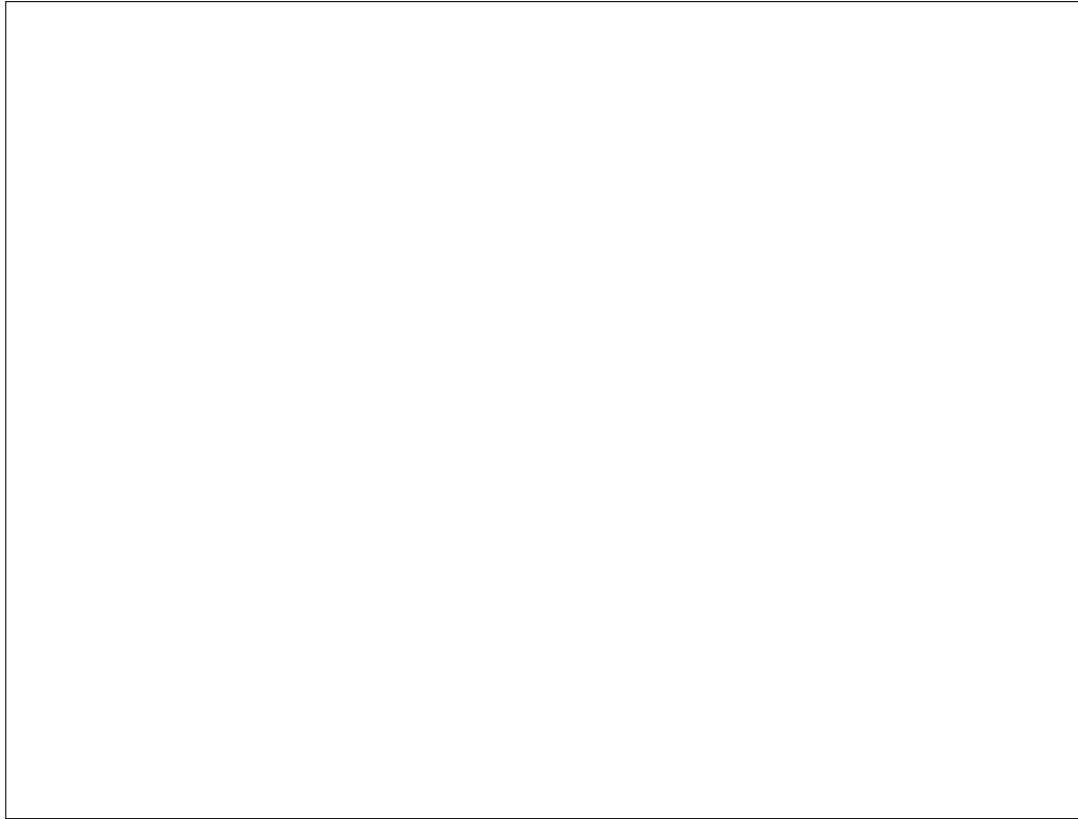
- The AMAT for instruction accesses will be different from data accesses since they reside in different portions of main memory.
- It is possible for the cache to be entirely filled with data blocks.
- Instruction accesses will affect data access hit rates.
- Assembly with few control-flow instructions will cause a high hit rate for instruction accesses.
- None of the above.

(c) Cached Datapath

We now choose to add caching to the single cycle datapath. We add 2 single level caches; Cache A is accessed only on instruction fetch and Cache B is accessed only on data memory access prior to direct memory accesses.

As we observe our datapath, we see that having heavy caching in addition to our respective memory arrays does help our average access times in comparison to having no caching, as is the expected behaviour. However, in our worst-case scenarios, we see that the time taken is suboptimal. We instead propose an alternative method such that our memory arrays (IMEM and DMEM) are replaced by identical caches. Cache A replaces IMEM and Cache B replaces DMEM. What modifications need to be made to the control logic signals? Give your answer in 10 words or fewer.

i.



- ii. How does missing in the cache affect how our datapath operates? Frame your answer in terms of how it affects the control signals and hazards in 2 or fewer sentences.

- iii. Which of the following is always true about our caching setup?

- Assembly with few control-flow instructions will cause a high hit rate for Cache A.
- The AMAT for Cache A and Cache B will be the same since the caches are identical.
- Intended IMEM accesses will cause cache incoherence with intended DMEM accesses if the memory addresses are close.
- Instruction and data accesses will cause both Cache A and Cache B's states to change for every access.
- None of the above.

8. Number Rep

- (a) i. Convert $(52)_8$ to base 10. Leave the answer as a plain integer. DO NOT add the subscript indicating the base.

9. Floating Point

- (a) i. Consider a 20-bit floating-point number with the following components (1 sign bit, 6 exponent bits, 13 mantissa bits); i.e.,

SEEEEEEMMMMMMMMMMMMM

All other properties of IEEE754 apply (bias, denormalized numbers, NaNs, etc). The bias is the usual $-(2E-1)$, which here would be $-(25-1) = -31$.

What is the bit representation (in hex) of the floating-point number -8.25? Your answer must be in hex, must be prepended with `0x`, and all letters must be capitalized. Do not include any extra leading zeros.

10. RISC-V

- (a) Note: For the following question, you may NOT use Venus; indeed, the questions have been written so that using Venus is counterproductive.

Part 1: Mini coding questions

Implement the following functions in RISC-V 32b integer assembly. You may NOT use any extensions (eg: you cannot use `mul` and you can't use floating point or vector extensions). You may only write one line of RISC-V code per blank. Do NOT include commas.

i. 1: `_mm32add_epu8`:

Inputs: `a0` and `a1` are vectors (NOT a pointer to the vector) of four 8-bit unsigned integers **less than 100**.

Output: `a0` returns a vector of four 8-bit unsigned numbers such that `result[i] = a0[i]+a1[i]` for $0 \leq i < 4$.

Example: If the inputs `a0` and `a1` were `[15,2,3,99]` and `[5,6,7,99]` (encoded as `0x0F020363` and `0x05060763` respectively), the expected output would be `[20, 8, 10, 198]`

```
_mm32add_epu8:
----- #Code line 1
ret
```

Code line 1:

ii. 1: `floatlessthan`:

Inputs: `a0` and `a1` are **positive** non-NaN IEEE-754 32-bit floating point numbers.

Output: `a0` returns 1 if `a0 < a1` and 0 otherwise.

Example: If the inputs `a0` and `a1` were 1.5 and 1.75, the expected output would be 1. If `a0` and `a1` were 1.5 and 1.5 respectively, the expected output would be 0.

```
floatlessthan:
----- #Code line 2
ret
```

Code line 2:

iii. 1: skipline:

Inputs: None

Output: None

Effect: Skip the next assembly instruction in the caller function. We are only using the 32b RISC-V ISA (no 16 bit extension) You may assume that the next line of code exists, and is not a pseudoinstruction.

Example: Assume that the following code was run:

```
addi t0 x0 5
jal skipline
addi t0 t0 300
addi t0 t0 10
```

Then at the conclusion of this code, t0 would be equal to 15.

```
skipline:
----- #Code line 3
ret
```

Code line 3:

iv. 1: endofstring:

Inputs: a0 is a pointer to a **nonempty** string

Output: a0 returns a pointer immediately after the end of the string.

Example: Let a0 be the pointer 0x10000000 to string s, which is the string "Hello". Then the expected output would be 0x10000006.

```
endofstring:
___t0_____ #Code line 4. You must use t0 as the first register of this instruction.
----- #Code line 5
----- #Code line 6
ret
```

You **may not** use strlen or any other library function

Code line 4:

v. Code line 5:

vi. Code line 6:

(b) Part 2: RISC-V translation

As a reminder, you may NOT use Venus for this question. As a reminder, hexadecimal strings should be written with the “0x” prefix, with CAPITALIZED hex digits (ex. 0xDEADBEEF).

Translate the following instruction to hexadecimal: `srai t0 s3 16`. Remember to include the “0x” at the beginning!

No more questions.