

You are viewing this thread in readonly mode.

[Final] Past Exams - 2023 #984

E **Eric Che** STAFF 3,763
7 months ago in **Exam - Final** VIEWS




You can find the past exams here: <https://cs61c.org/sp24/resources/exams/>. Please check the linked past Piazza/Ed Q&A PDFs first before asking here. Many of the questions are already answered in those! Video walkthroughs (if available), are also linked on that page!

When posting questions, please reference the semester, exam, and question in this format so it's easier for students and staff to search for similar questions:

Semester-Exam-Question Number

For example: **SP22-Final-Q1**, or **SU22-MT-Q3**

As a note, some questions will be out of scope because set-associative caches are out of scope this semester.

 **Anonymous Cormorant** 7mth #984beec ✓ Resolved
sum23. final. Q7.3

Why there is a break with CodaBot in hour 4?

CodaBot wants help, and asks their friend EvanBot for help. Here is how long EvanBot takes to perform each task:

Task Number	Time (hours)
0	4
1	3
2	1
3	1
4	3
5	2
6	3

Q7.3 (3 points) Now, suppose both CodaBot and EvanBot can only perform one task at a time, and each task can only be performed by one Bot. What is the minimum amount of time required to complete these set of tasks?

Solution: 9 hours. Example task list:

Hour	CodaBot	EvanBot
0	Task 1	Task 0
1	Task 1	Task 0
2	Task 1	Task 0
3	Task 2	Task 0
4		Task 3
5	Task 4	Task 6
6	Task 4	Task 6
7	Task 4	Task 6
8	Task 5	

♡ ...



Andrew Liu STAFF 7mth #984beed

CodaBot has nothing to run there — all the tasks remaining (3,4,5,6) have prerequisites that are not met.

♡ ...



Anonymous Elk 7mth #984becb

✓ Resolved

Fa23-Final-Q2.6

Why wouldn't it be None of the above since if $rs1 == rs2$, then we wouldn't right anything back?

Q2.6 (1.5 points) WBSel

- PC + 4
- ALUOut
- MemReadData
- Doesn't matter
- None of the above

Solution: We need to write `PC + 4` back to `rd`, so we should select `PC + 4` for WBSel.

♡ ...



Minh Nguyen STAFF 7mth #984becd

The problem states: if $rs1 == rs2$, then $rd = PC + 4$; so we do write to `rd` !

♡ ...



Anonymous Elk 7mth #984bece

Sorry I meant when $rs1 \neq rs2$, then we wouldn't go into the branch and write anything back, right?

♡ ...



Andrew Liu STAFF 7mth #984becf

↩ Replying to Anonymous Elk

Notice how `RegWEn` depends on `rs1` and `rs2` — for this reason, it is always safe to write back `PC + 4` and let `RegWEn` handle that case

♡ ...



Anonymous Horse 7mth #984beba

✓ Resolved

fa23 final q5

```
1 bool more_even_simd(uint32_t* array, uint32_t n)
2   vector counts = vec_setnum( 0 );
3   vector mask = vec_setnum(1);
4   for (uint32_t i = 0; i < n / 4 * 4; i+=4) {
5     vector temp = vec_load(array + i);
6     vector masked = vec_and(temp, mask);
7     counts = vec_add(counts, masked);
8   }
9   return (n / 2) >
10          (vec_sum(counts));
11 }
```

In line 5, why is it `array + i` and not `array[i]`?

♡ ...



Anonymous Snail 7mth #984bebb **ENDORSED**

`Array[i]` would give you the value at index `i`. You need the index of array to load in the values at that index which can be found by offsetting array with `+i`.

♡ ...



Anonymous Horse 7mth #984beee

And that already does the `* sizeof(int)` for us right?

♡ ...



Jedidiah Tsang STAFF 7mth #984befa

↩ Replying to Anonymous Horse

Correct, pointer arithmetic multiplies the constant by the size of whatever the pointer is pointing at.

♡ ...



Anonymous Red panda 7mth #984beac

✓ Resolved

fa23-final-q3



I'm slightly confused at the wording of this question. In the beginning explanation, it says "the CPU will always predict that branches are not taken". However for the line `beq x0 x0 label`, we are guaranteed that this branch is always taken.

♡ ...

 **Anonymous Snail** 7mth #984bebc  **ENDORSED**

The prediction is incorrect which leads to a hazard that you need to deal with. If the prediction were correct you would have no hazard.

♡ 1 ...

 **Anonymous Dragonfly** 7mth #984beaa  **Resolved**

fa23-final-q8

For line 6, I had `jalr x0 t3 4` (jump one more instruction) with everything else prior the same: my logic was that we would calculate which instruction to jump through with $2(x) + 1 = \text{number of instructions to jump from line 6 with } x = a0 \% 4$. First we get the $\%4$ store that in `t3`, then multiply by 8 (this is effectively $2(x)$) and then load PC into `t7` then finally we also need to add the 1 instruction (4 bytes) in line 6. Why would we want to jump another 3 instructions? For example if $x = 0$, we just want to move 1 instruction down to line 7 not 3 instructions down.

Also for line 13, since we are told we are only given unsigned ints, would `srli` vs `slli` make a difference?

♡ ...

 **Andrew Liu** STAFF 7mth #984bedd

Remember that `auipc` loads the current PC + `imm` into the register it specifies, so you need to count the jump from the line with `auipc`, not the line with `jalr`.

♡ ...

 **Anonymous Dragonfly** 7mth #984bdff  **Resolved**

fa23-final-q3.2

Why do we not need to stall at all between `addi t0 x0 9` and `addi t1 t0 2`? Shouldn't we wait until finishing the EX to get the correct value of `t0` to use for the next line? Is this something that is resolved by forwarding or double pumping (and what is the difference)? Does forwarding resolve all data hazards except for loads? Thank you!

Q3.2 (3 points) If we implement double pumping and all forwarding paths, during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is still the control hazard from line 1 to line 7, as the branch will always be taken. The data hazard from line 7 to line 8, for register `t0` is resolved via forwarding. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11
1. <code>beq x0 x0 Label</code>	IF	ID	EX	MEM	WB						
2. <code>addi x0 x0 3 -> nop</code>		IF	X	X	X	X					
3. <code>addi x0 x0 1 -> nop</code>			IF	X	X	X	X				
4. <code>addi x0 x0 4 -> nop</code>				IF	X	X	X	X			
7. <code>addi t0 x0 9</code>					IF	ID	EX	MEM	WB		
8. <code>addi t1 t0 2</code>						IF	ID	EX	MEM	WB	
9. <code>xori t1 x0 6</code>							IF	ID	EX	MEM	WB

♡ ...



Anonymous Cormorant 7mth #984bebd

since we're using double pumping and forwarding, the value of the previous execution is passed on the the next execution the following cycle.

"no double pumping" means that the reg files cannot handle reading and writing at the same time

CS 61C Solution 2: Forwarding

Forwarding, aka bypassing, uses the result when it is computed.

- Don't wait for value to be stored into RegFile.
- Instead, grab operand from the pipeline stage.

Implementation:

- Make extra connections in the datapath.
- Also add forwarding control logic.

22-Pipeline-II-Hazards (12)



Pracheeti Shikarkhane STAFF 7mth #984bebf

Anonymous Cormorant is correct. The value that should be stored in t0 is computed in the Execute stage of instruction 7. Since we're told that all forwarding paths are implemented, we can forward this computed t0 value from instruction 7's Execute stage to instruction 8's Execute stage (which occurs in the next cycle) so that instruction 8 ends up using the correct t0 value.

Forwarding is able to resolve most data hazards depending on what forwarding paths exist in that problem.



Anonymous Quail 7mth #984bdfe

✓ Resolved

Fa23-Final-Q10.2

Although reading the explanation in #984aabf, I'm still confused about the splitting. How do you go about?



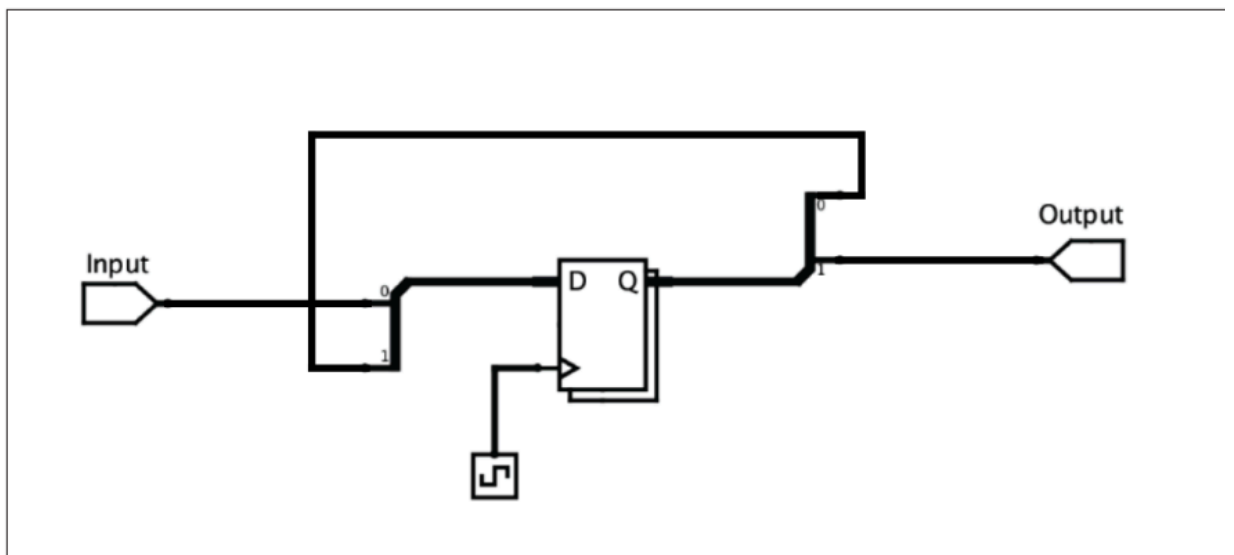
Myrah Shah STAFF 7mth #984beaf

Is there a particular part of the explanation that is confusing?



Anonymous Horse 7mth #984bdfc

✓ Resolved



Q10.2 Fa23

Wouldn't this have a 1 in the output if the input started off as 11?

♡ ...

Anonymous Gerbil 7mth #984bdfd

I got that too. I have no idea how this is the FSM

♡ ...

M Myrah Shah STAFF 7mth #984beae

Yep! From this circuit:

N = 0, Input = 1: The input to the register is 1, which is then sent to the 1st bit of the input to the register.

N = 1, Input = 1: We have the 1st bit of the input as 1 from N=0, so now we are inputting 11 into the register. After outputting from the register, we send bit 1 to output, which will be 1.

♡ ...

Anonymous Horse 7mth #984beef

But didn't we want two timesteps where output was 0?

♡ ...

Anonymous Partridge 7mth #984bdfb

✓ Resolved

FA23-Final-Q3

Q3.1 (3 points) If all hazards are resolved through stalling (no double pumping or forwarding paths), during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X	X				
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2									IF	ID	EX	MEM	WB	
9. xori t1 x0 6										IF	ID	EX	MEM	WB

Why does IF have to wait to execute after MEM in this example, but in lecture, IF was able to execute right after the previous instruction's IF?

♡ ...

 **Anonymous Cormorant** 7mth #984bebe  **ENDORSED**

the question stated there's no double pumping. "no double pumping" means that the reg files cannot handle reading and writing at the same time

♡ 1 ...

 **Anonymous Monkey** 7mth #984bdcc  **Resolved**

Sp23-Final-Q1



how do we know how many times to count the mux? i thought the mux would be counted 3 times (one for asel, one for bsel, and one for wbsel) but we only count it twice. is there a reason for this?

♡ ...

 **A** **Abhinav Vedati** STAFF 7mth #984bddd

(assuming that you mean Q2) Notice that the Asel and Bsel muxes are in parallel; there's no single path that goes through both muxes. When we are trying to calculate a critical path, we want a single path.

♡ ...

 **Anonymous Whale** 7mth #984bdbc  **Resolved**

Fa23-Q5

Isn't it supposed to return $(n+1)/2$ instead of $n/2$?

because `vector_sum(counts)` returns number of odd numbers.

if $n=7$, and there are 3 odds, then there are 4 evens. it should return true

♡ ...

 **A** **Abhinav Vedati** STAFF 7mth #984bdbd

This is true, but n is a multiple of 4, so it will always be even.

♡ 2 ...

 **Anonymous Leopard** 7mth #984bdae  **Resolved**

Sp23-Final-Q8.3

Is a valid alternative answer to this question to have B do 0 and 1, A do task 4 and C do task 2 and 3?

♡ ...

 **Andrew Liu** STAFF 7mth #984bede

Yeah that sounds fine to me, writing out your solution, we have

$t = 0$: B does 0 taking 50 minutes,

$t = 40$: C does task 2 taking 60 minutes

$t = 40$: B does task 1 taking 50 minutes

t = 100 : A does task 4 taking 100 minutes

t = 100: C does task 3 taking 60 minutes

t = 160: C finishes

t = 200: A finishes

♡ ...



Anonymous Barracuda 7mth #984bcec

✓ Resolved

Fa23-Q9.1

Is flip bit 12 of instruction bne to turn into beq a valid alternative sol?

♡ ...



A Abhinav Vedati STAFF 7mth #984bdde

I don't see how this would be more efficient than the given RISC-V orion implementation. Essentially, this is equivalent to outputting 1 when the original implementation outputs 0, and 0 when the original implementation outputs 1, which shouldn't increase our efficiency. That being said, a valid solution to this question on the final is to flip no bit at all, so technically this is also valid solution (although it won't do anything), given that you extract ORNUM from the orion function directly in solve_orion.

♡ ...



Anonymous Wolverine 7mth #984bcea

✓ Resolved

Fa23-Q3.1

Are we assuming we have to first write back the new value before accessing in reg files? (I thought our reg files can handle reading and writing at the same time following write then read convention)

Q3.1 (3 points) If all hazards are resolved through stalling (no double pumping or forwarding paths), during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X					
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2									IF	ID	EX	MEM	WB	
9. xori t1 x0 6										IF	ID	EX	MEM	WB

♡ ...



Anonymous Gorilla 7mth #984bdab **ENDORSED**

I believe that "no double pumping" means that the reg files cannot handle reading and writing at the same time in this case.

♡ ...



Why do we only want worker to send to process 0 instead of process x?

A worker should have the following behavior:

Q6.1 (1 point) Before the main loop...

- Perform task 0
- Cleanup and exit process
- Send the `DONE(-1)` message to process 0
- Do nothing

Solution: Since `DONE(-1)` is defined to indicate that a worker is ready but has not performed a task, we send this message when a worker is about to start.

Q6.2 (1 point) Within the main loop, if we receive an `EXECUTE(x)` message...

- Perform task `x`
- Perform task `x`, then cleanup and exit process
- Perform task `x` and send the `DONE(x)` message to process 0
- Perform task `x` and send the `DONE(x)` message to process `x`
- Do nothing

Solution: We must include the task number in the done message in order to be able to handle prerequisites.

♡ 1 ...



A

Anthony Salinas Suarez STAFF 7mth #984bcfd

Recall that as part of the manager-worker framework, the worker must communicate with the manager upon executing the task in order to receive more work (if there is any left). For us, the manager is designated to be process 0 and not x!

Worker Pseudocode

CS 61C

Spring 2024

```
Set up
While True:
    Send to the manager "I'm ready for more work"
    Receive message from manager
    If message is "Here's more work":
        Do the work
    Else if message is "All work done":
        break
Finalize
```

Manager Pseudocode

CS 61C

Spring 2024

Set up

While there's work to do:

 Wait until a worker says "I'm ready for more work" (recv from all)

 Find the next task to do

 Send to the worker what task to do

Repeat #Worker times:

 Wait until a worker says "I'm ready for more work" (recv from all)

 Send to the worker "All work done"

Finalize

20



Anonymous Wolverine 7mth #984bcdb

✓ Resolved

FA23-final-Q10.1

Can someone help illustrate how we arrived at our solution to this problem? Or if there are any resources to these type of questions? I'm utterly confused by the wording (what does it mean "for the first two time steps") and I found it hard to understand the solution as well (not sure how the FSM matched the expected input/output). Thanks!

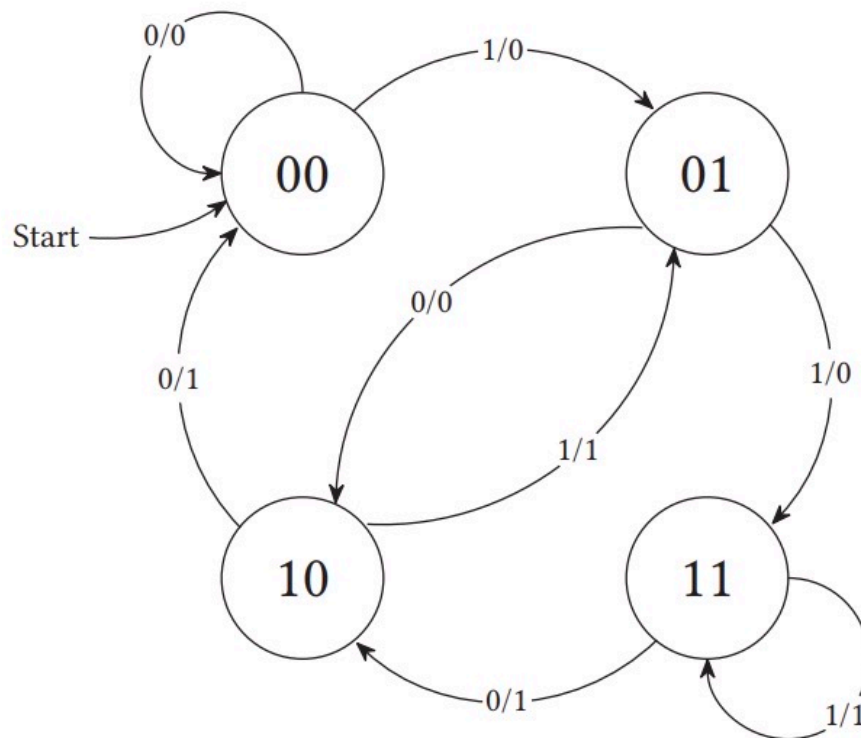
Q10 *Cumulative: Art Class*

(16 points)

You wish to create an FSM whose output at time step N is 0 for the first two time steps, and the input of time step $N - 2$ otherwise.

For example, if the input to this FSM was 0b01 1011 1011 1000 1001,
the output should be 0b00 0110 1110 1110 0010.

Q10.1 (8 points) Complete the FSM below. You may not add additional states. Note that you must also label the state transitions we have provided for you.



♡ ...

Anonymous Penguin 7mth #984bcff

Every time step, the FSM proceeds to the next state, and the exact state depends on the next bit in our input bitstring. So for example, if we were at state 01 at some instant, then one time step later we would either be at state 11 or state 10 depending on the next input we get. So in general since the N th output bit is equal to the $(N - 2)$ th input bit for $N > 2$, we need a base case for the first two time steps.

[#984bcac](#) <-- Does this help explaining the overall idea?

♡ ...

Anonymous Wolverine 7mth #984bdac

helps, thanks!

♡ ...



Anonymous Wolverine 7mth #984bdad

Ah I didn't noticed the 2 bit delay that makes a lot more sense thanks!

♡ ...



Anonymous Barracuda 7mth #984bccf

✓ Resolved

FA23-Final-Q8.7

Instead of nop, can we do addi a0 a0 0?

♡ 1 ...



Abhinav Vedati STAFF 7mth #984bddf

A Sure, I don't see why not.

♡ 1 ...



Anonymous Woodpecker 7mth #984bcfb

✓ Resolved

FA23 Final Q2.7

Hi, for this question, I don't really understand where the instruction is performing a read and write in the same cycle, and also why the rest of the control signals would be the same as the load instruction. Would appreciate clarification, thanks!

Q2.7 (4 points) What additional changes would we need to make to our datapath in order for us to implement `a1sw` (with as few changes as possible)? Select all that apply.

- Create a new instruction type and update the ImmGen
- Add a new output to the RegFile for a third register value
- Add another WriteData and WriteIndex input to the RegFile
- Add a new input to the AMux and update any relevant selector/control logic
- Add a new input to the BMux and update any relevant selector/control logic
- Add a new ALU operation and update any relevant selector/control logic
- Add a third input into ALU and update any relevant selector/control logic
- Allow the ALU to send out more than 1 output and update any relevant selector/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 input to the WBMux and update any relevant selector/control logic
- None of the above

Solution: Since this instruction has `rd`, `rs1`, `rs2`, and `imm`, we need to create a new instruction type (as no current instruction type support all of these fields). We also need to create a new immediate type for this new instruction type since we can't share an immediate type with another instruction type.

This instruction also performs one read and one write memory operation in the same cycle, which is currently not supported by our DMEM (as it only has one port). Therefore, we need to allow the DMEM to read and write in the same cycle.

The remaining control signals remain the same as a load instruction, so we don't need to add any additional inputs/outputs to other components.



Andrew Liu STAFF 7mth #984bedf

The instruction does the following:

```
rd = *(rs1 + imm) // DMEM Read from address rs1 + imm
*(rs1 + imm) = rs2 // DMEM Write to address rs1 + imm
```

So we have both a DMEM read and write in the same cycle.

The rest of the control signals are the same because we have basically the same behavior as a load instruction — some register gets written to, its not a branch, we use `rs1` and `imm` as inputs to the ALU, the ALU, adds, etc.



Anonymous Lobster 7mth #984bcbd

✓ Resolved

FA23-Final-Q8

Can I get a walkthrough of the solution? I don't understand why we are doing `andi t3 a0 3`. According to other threads, it is equivalent to $x \bmod 4$ but, I'm not sure why.

$$f(x) = \begin{cases} x + 9 & \text{if } x \% 4 == 0 \\ x * 2 & \text{if } x \% 4 == 1 \\ x & \text{if } x \% 4 == 2 \\ x // 8 & \text{if } x \% 4 == 3 \end{cases}$$

Jero wants to write this function in RISC-V, but he couldn't get his CS61CPU's branch instructions to work! As a result, you may use any RV32I instruction **except** branch instructions.

Write a function, `f`, which accepts one argument `x` in `a0`, and returns `f(x)`. You may assume that there is no overflow.

```

1 f:
2  andi t3 a0 3
   q8.1
3  slli t3 t3 3
   q8.2
4  auipc t7 0
   q8.3
5  add t3 t3 t7
6  jalr x0 t3 12
   q8.4
7  addi a0 a0 9
   q8.5
8  j f_end
9  slli a0 a0 1
   q8.6
10 j f_end
11 nop
   q8.7
12 j f_end
13 srli a0 a0 3
   q8.8
14 f_end:
15 ret

```



Anonymous Monkey 7mth #984bdba

i think we're doing $t_3 a_0 3$ so we can retain the last 2 bits because the last 2 bits is all we need to know to see the remainder of a_0 if we divided it by 4. (3 in binary is $0b11$, which is why the bit masking works, and if you walk through a few examples such as $9=0b100\underline{1}$ and $11=0b10\underline{11}$, u can see that you only need the 2 least significant bits to know what those numbers mod 4 would look like. for example, $9 \% 4 = 1$, which is $0b\underline{01}$ and $11 \% 4 = 3$ which is $0b\underline{11}$)

sorry if any of this was not explained well but i hope it helped! it took me a minute to figure it out too

edit: i think thread [#984bec](#) has an answer by a ta

♡ ...



Anonymous Pony 7mth #984bcaf

✓ Resolved

SP 23 Final Q2.1

I wonder why we have imm-gen

since we divide the datapath to IF/ID and EX/MEM/WB, so I am thinking we should find the t logic max. The calculation in the answer includes imm-Gen. However, isn't imm-Gen in stage 1 (ID), and we find that the maximum logic path is in stage 2?

Thanks!

Sp23 Final Q2.1

since the load instruction write back to regfile and write back wouldn't be pipelined, why wouldn't regfile setup be present instead of the register setup in the answer?

Q2 IF Only ID Pipelined Better

(10 points)

In Project 3, we implemented a RISC-V CPU with two stages; stage 1 included IF and stage 2 included ID/EX/MEM/WB. For this question, imagine instead that we implement a two-stage pipeline with a different split; stage 1 will include IF/ID and stage 2 will include EX/MEM/WB (IF/ID/EX/MEM/WB are defined equivalently to the pipelined CPU on the reference card).

For Q2.1 and Q2.2, assume the following delays for each component. Any component not listed is assumed to have a negligible delay.

Component	Delay
$\tau_{\text{clk-to-q}}$	35ps
τ_{setup}	20ps
Mux	75ps
Regfile Setup	20ps
Regfile Read	175ps
Immediate Generator	150ps
Branch Comparator	200ps
ALU	200ps
Memory Read	300ps

Q2.1 (3 points) What is the minimum clock period of this circuit, in picoseconds, to achieve correct behavior?

Solution: 855ps ($\tau_{\text{clk-to-q}}$ (35) + Immediate Generator (150) + Mux (75) + ALU (200) + Memory Read (300) + Mux (75) + τ_{setup} (20)).

The critical path occurs in stage 2 for a load instruction.

♡ ...



Anonymous Penguin 7mth #984bcc

#984aa

♡ ...



Anonymous Newt 7mth #984bbfe

✓ Resolved

FA23-Final-Q10

I'm having trouble understanding what Q10 is doing despite watching the FSM lecture and reading the slides. What is going on in this FSM? Are they breaking the input of the FSM into 2 bits each (01 10 11 10 etc)? How did they determine the input (whether input is 1 or 0)?

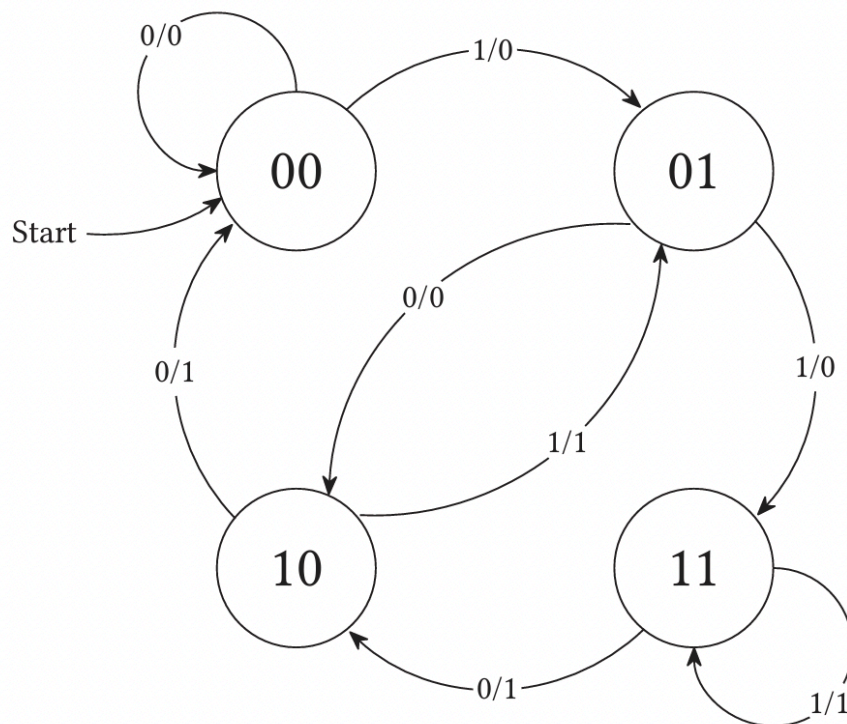
Q10 Cumulative: Art Class

(16 points)

You wish to create an FSM whose output at time step N is 0 for the first two time steps, and the input of time step $N - 2$ otherwise.

For example, if the input to this FSM was 0b01 1011 1011 1000 1001,
the output should be 0b00 0110 1110 1110 0010.

Q10.1 (8 points) Complete the FSM below. You may not add additional states. Note that you must also label the state transitions we have provided for you.



♡ ...

 **Anonymous Penguin** 7mth #984bcac

00, 01, 10, and 11 are just names for the states and are not the input/output themselves. Between two states (or a state to itself), the label A/B implies that A is the input and B is the output when transitioning. We can try going through the FSM using the given example input: Beginning at state 00, the first input bit is 0, so we stay at state 00 and output 0. The next bit is 1, so we go to state 01 from state 00 and output 0. The next bit is 1, so we go to state 11 from state 01 and output 0. You can continue through this process and obtain the full output string.

I think I finally understood this problem when I noticed that every time we go from one state to another, we're removing the leftmost bit in the name of our original state, outputting that bit, and adding the next input bit to the end of the name. For example, when we transition from state 01 to 11, we output 0 and so remove that from the name, then we add 1 to the end (since it's our input) to get 11 as our new state name. This is really where the $N - 2$

delay idea comes from, because we always have a delay of 2 steps between when we receive some input x and output x (since the names of our states are 2 bits long, so x takes two steps to "get through" the entire name).

♡ ...



Anonymous Newt 7mth #984bcad

how did you figure out what the input/outputs are?

♡ ...



Anonymous Penguin 7mth #984bcba

↩ Replying to Anonymous Newt

Well an FSM is supposed to represent what can happen for all possible inputs and outputs, right? Whenever we input a bitstring into an FSM, it makes a transition after reading each individual bit. And if we're in state 00, it means that the next two bits we need to output are 00. If we're in state 01, we need to output 01 as our next two bits, etc., because the output is always delayed by 2 steps relative to the input. So the names of the states kind of give you a clue as to what the input and outputs should be between states.

♡ 1 ...



Anonymous Newt 7mth #984bccc

↩ Replying to Anonymous Penguin

that makes sense, thanks!

♡ 1 ...



Anonymous Leopard 7mth #984bbfd

✓ Resolved

Sp23-Final-Q4.5

is this a multi-level cache? Is this in scope?

♡ ...



A **Abhinav Vedati** STAFF 7mth #984bbff

Yes, this is a multi-level cache, and multi-level caches are in scope this semester. Lecture 30 and 31 cover them, as well as the AMAT equation.

♡ ...



Anonymous Goshawk 7mth #984bbfc

✓ Resolved

FA23-Final-Q4

How do we know that each memory access is distinct? For example, in 4.2, the answer sheet assumes that `arr[0] += arr[0]` is three memory accesses (load `arr[0]`, load `arr[0]`, store `arr[0]`) where it could just as easily be the case that there is only one memory access for loading and one for storing (since the same data is loaded twice). Is this not a bit ambiguous?

♡ ...



A **Abhinav Vedati** STAFF 7mth #984bcaa

At compile time, the statement `arr[i] += arr[0]` (the original line from the test) is translated to `arr[i] = arr[i] + arr[0]`, which is where we get the 2 loads and store from. Seeing it this way, it's a bit tricky for the compiler to figure out that the same data is being loaded in twice (and this may not even be true, depending on the value of `i`). To get the accurate hit/miss statistics, you have to examine the program without any shorthands like

`+=` .



Anonymous Swan 7mth #984bbbed ✓ Resolved

SU23-Final-Q8.6/8.7:

I understand that the standard bias will be -7 but I'm having trouble understanding how the bias is added to $0b1 \cdot 2^3$ to become $0b1010$ for 8.6, and same for 8.7 with the exponent and mantissa bits. Are there lecture slides I should reference/review for floater points?

Q8.6 (1.5 points) 0x200

Solution: 0x0A00

Each chunk is 64 bytes, so that accounts for the lower 6 bits of the address, 0x00. The chunk address is the upper 6 bits of the address, is 0x08. We then must translate 0x08.00 to the floating point system described.

$0x08.00 = 0b1 \cdot 2^3$, so the exponent is 0b1010 after applying the bias, and the mantissa is all 0's. This gives us 0x0A00.

Q8.7 (1.5 points) 0x7D8

Solution:

Each chunk is 64 bytes, so that accounts for the lower 6 bits of the address, 0x18. The chunk address is the upper 6 bits of the address, is 0x1F. We then must translate 0b1 1111.01 1000 to the floating point system described.

$0b11111.011000 = 0b1.1111011 \cdot 2^4$, so the exponent is 0b1011 after applying the bias, and the mantissa is 0xF6. This gives us 0x0BF6.



A Abhinav Vedati STAFF 7mth #984bcab

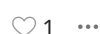
I think that the explanations for these are a bit ambiguous. To clarify, the actual value of the exponent for Q8.6 (ignoring biased representation) is 3, as the number we are representing is $0b1 \cdot 2^3$. To find the biased representation of 3, we **add** our bias of 7 (as per lecture 6, our bias is positive), resulting in 10 or 0b1010 in binary. Similarly, for Q8.7, our actual exponent is 4, and to represent 4 in biased notation, we **add** our bias of 7 to get 11 or 0b1011 in binary.

This works because we are trying to convert our exponent from the range of $[-\text{bias}, 2^{(n-1)} - \text{bias}]$, where n is the # of exponent bits, to the range of $[0, 2^{(n-1)}]$ so that we can represent it with our n exponent bits without dealing with 2s complement negative numbers. Thus, we have to add the bias to get from the first range to the second range. I'd recommend reviewing the example on lecture 6, slide 24, as well as that lecture as a whole, to better understand this process.



Anonymous Swan 7mth #984bcfb

Thank you that makes a lot of sense!



Anonymous Whale 7mth #984bbeb ✓ Resolved

Fall 2023 Q3, I am wondering the logic behind line1 ->line 7. Does PC get updated during DMEM stage or ALU stage? It seems like PC gets updated during DMEM stage, as only after DMEM stage does PC gets updated to the correct value, so program knows the next instruction is at line 7, correct?

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X	X				
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2									IF	ID	EX	MEM	WB	
9. xori t1 x0 6										IF	ID	EX	MEM	WB

♡ 2 ...



A **Abhinav Vedati** STAFF 7mth #984bcae

The PC actually updates during the next IF stage (lecture 23, slide 24), because the PC is a register itself, and needs to wait for a clock pulse to update. The reason that the control hazard is from line 1 to line 7 is that line 1 is a branch that will always be taken to line 7. This isn't known until 3 cycles later, and the PC isn't updated until 4 cycles later. This is why we need to stall for 3 cycles.

♡ ...



Anonymous Horse 7mth #984bbdc

✓ Resolved

Su 23

Q8.1 (2 points) If we had 4KiB of memory, with chunk addresses 0, 1, 2, etc., what is the minimum number of exponent bits in our floating point memory address required to access every byte, assuming that we use a standard bias?

Solution: Given 4KiB memory and 64B chunks, we have 64 chunks total. Therefore, we need 4 exponent bits to be able to represent values 0 through 63. Given 4 exponent bits, the largest non-infinity/NaN exponent value (pre-bias) is $2^4 - 2$, which, after applying the standard bias of -7, is 7, which allows us to represent values up to 63.

If we used 3 exponent bits, the largest exponent value after applying the standard bias of -3 is 3, which cannot represent the value 63.

Q8.3 (1.5 points) What is the minimum number of mantissa bits in our floating point memory address required to address every byte?

Solution: The largest memory address we need to represent is $0b11\ 1111.111111\ (63 + \frac{63}{64})$. To represent this value, we need 11 bits of mantissa ($0b1.11111111111 \cdot 2^5$).

Could I get an explanation for these two questions? Just because we have 64 chunks, why do we need the exponent to represent 0-63? Also for 8.3, how to the 1s after the decimal evaluate to $\frac{63}{64}$?

♡ ...

N **Nikhil Kandkur** STAFF 7mth #984bbea

We want to be able to address every chunk provided, and since we are working with exponent bits, we need to set our exponent bits such that 64 is the maximum number represented by the exponent (we need to be able to support an exponent of 8). To represent $\frac{63}{64}$, remember that each 1 after the decimal point represents $\frac{1}{2^{\text{place after decimal point}}}$, so $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64}$ would be $\frac{63}{64}$, so we would need 6 1's after the decimal point for this to be represented.

♡ ...

Anonymous Wolverine 7mth #984bbdb ✓ Resolved
Sp23 Q2.2

Why can't we also move ASel and Bsel muxes to the first stage as well? which by calculation, the first stage will cost 570 + 35 + 20, and second stage will cost 575 + 35 + 20, which will decrease the minimum clock period.

♡ ...

A **Abhinav Vedati** STAFF 7mth #984bcbb

The ASel and Bsel muxes' outputs potentially depend on the RegReadData1 and RegReadData2 values, which will not be known until the third stage. We can move the immediate generator because it only depends on the instruction and control logic, both of which are known in the first stage.

♡ ...

Anonymous Horse 7mth #984bbbb ✓ Resolved
Sp23 Q4.2

Q4.2 (2.5 points) What is the overall hit rate for a call to `convolve_1d`? No need to simplify the fraction.

Solution: $\frac{2}{15}$

There are a total of three iterations of the outer loop, and two iterations of the inner loop per iteration of the outer loop. For each iteration of the inner loop, there are two memory accesses: reading `b[j]` and reading `a[i + j]`. Each outer loop has an additional memory access: writing to `results[i]`. In total, there are 5 accesses per iteration of the outer loop, and 15 accesses overall.

For each set of accesses to `a` and `b`, the program experiences thrashing since the index of these blocks conflict. As a result, the accesses to `a` and `b` will always be misses (first two compulsory, then the rest are conflict).

The accesses to `results` do not cause any thrashing, so out of the three accesses (one per iteration of the outer loop), there is one compulsory miss and two hits, giving us a hit rate of $\frac{2}{15}$.

How do we get a hit at all? There are only 4 lines in the cache, so once the inner loop is done, the cache is filled. Result runs, misses. Then outer loop runs inner loop again and fills up the cache with 4 new values. How does result even hit?

♡ ...

S **Skyler Chan** 7mth #984bbbc

The 2 hits are from results I believe. Since it's direct mapped, and since `results = 0x3030`, the first outer for loop is a compulsory miss for results but this stores it in the cache. Then the second and third(last) outer for loop are hits since results is still in that designated spot in the cache.

♡ 1 ...

 **Fouad Sinno** 7mth #984becc

Why do the accesses to a and b not count as hits when their indices are repeated? For example, b[0] will occur more than once, so why doesn't that count as a hit?

♡ ...



Anonymous Barracuda 7mth #984bbba

✓ Resolved

F23-Final-Q5

Would this alternative still be correct:

5.2 = vec_setnum(31) // this is to make the binary encoding 11111

5.5 = vec_xor(mask, temp) // this would produce 1 if even and 0 if odd

5.7 = vec_sum(counts) // this is where counts is even and not counting the odds

5.8.= n - vec_sum(counts) // this is to obtain the amount of odds

♡ 1 ...



A **Abhinav Vedati** STAFF 7mth #984bcbe

No, but your logic is almost correct. The issue is that if we load 2, for example, and try the xor, our result will be $0b00010 \wedge 0b11111 = 0b11101 = 29$, but we want 1.

♡ ...



Anonymous Penguin 7mth #984bccb

I have a slightly different solution, does this work:

5.3 = n

5.7 = n - vec_sum(counts)

5.8 = 0

with all other lines being the same as in the official solutions.

♡ ...

A

Abhinav Vedati STAFF 7mth #984bcd

↩ Replying to Anonymous Penguin

No, because this results in a final return value of $(n - \text{vec_sum}(\text{counts})) > 0$, with all else being equal to the original solution. In the original solution we return $(n / 2) > (\text{vec_sum}(\text{counts}))$. This is equal to $(n) > (2 * \text{vec_sum}(\text{counts}))$, or $(n - \text{vec_sum}(\text{counts})) > (\text{vec_sum}(\text{counts}))$.

♡ 1 ...



Anonymous Barracuda 7mth #984bcce

Would the logic be correct if it was notor instead of xor?

♡ ...

A

Abhinav Vedati STAFF 7mth #984bcda

↩ Replying to Anonymous Barracuda

No, because with a not-or gate, if even one of the inputs is 1 the result will always be 0. We are using 0b11111 as our mask, so the output of a not-or operation will always be 0.

♡ ...

 **Anonymous Barracuda** 7mth #984bceb

↩ Replying to Abhinav Vedati

What if our mask was 0b11110? Like, `vec_setnum(30)` instead of 31

♡ ...

A **Abhinav Vedati** STAFF 7mth #984bcd

↩ Replying to Anonymous Barracuda

Yes, that should work (it will output 1 for even numbers and 0 for odd numbers). I admire the persistence! :)

♡ 2 ...

 **Anonymous Barracuda** 7mth #984bcee

↩ Replying to Abhinav Vedati

Awesome thanks Abhinav! And this would work even with the change in 5.7 and 5.8 (aka, it would be

`5.7 = vec_sum(counts) // this is where counts is even and not counting the odds`

`5.8 = n - vec_sum(counts) // this is to obtain the amount of odds), right?`

♡ ...

 **Anonymous Barracuda** 7mth #984bcef

↩ Replying to Anonymous Barracuda

Though technically this wouldn't work probably for the exam since there was no not-or function provided


♡ ...

A **Abhinav Vedati** STAFF 7mth #984bdaf

↩ Replying to Anonymous Barracuda

Yep!

♡ 1 ...

 **Anonymous Partridge** 7mth #984bbaf ✓ Resolved

SP23-Final-Q7

How did we know that there would be `num + 1` arrays in the result array? Also, for the first iteration of the for loop on line 15 when `i = 0`, doesn't `data[i] = codecopy` just set `data[0]` to all of the instructions and 0 since `codecopy` hasn't been modified yet?

♡ ...

↳ S **Skyler Chan** 7mth #984bbbd

We know it's `num + 1` arrays as if we inspect the example, it has 4 B/J instructions but the result has 5 elements. Thus, `num + 1`.

♡ ...

↳  **Anonymous Partridge** 7mth #984bbcf

How would one come to this conclusion during the exam though? I feel like I would have trouble believing that it's that simple, and I don't really understand the logic of why it is `num + 1`.

♡ ...

S **Skyler Chan** 7mth #984bcfa

↩ Replying to Anonymous Partridge

Honestly I just read the example and saw the pattern

♡ ...

Anonymous Barracuda 7mth #984bbae ✓ Resolved

F23-Final-Q5.10

Why is it slower than the naive more_even?

♡ ...

Anonymous Salmon 7mth #984bbdf 🔗 ENDORSED

since there isn't a "for" keyword in the #pragma omp parallel, each thread will run through the entire for loop once, making it number-of-thread times slower than the naive implementation

♡ ...

Anonymous Cormorant 7mth #984baef ✓ Resolved

fall23. final. Q7.3

Where do we find the 0x123 from the given tables? Why is the next PPN of 0x727 is 0x123?

Q7.3 (2 points) 0x61C0 02B1 ADE2

Solution: We split the address into VPN: 0x61 C002 and Offset: 0xB1 ADE2. Then, we look in the TLB and extract the following entry:

Dirty	Valid	VPN	PPN
1	0	0x61 C002	0x727

Since this is not valid, we go to the page table and extract the following entry:

Index	Dirty	Valid	PPN
0x61 C002	1	0	0x727

Since this is also not valid, we have a page fault, and assign the next PPN, 0x123. Thus, we have 0x123B1ADE2.

- TLB hit
- TLB miss and page table hit
- Page fault

♡ ...

Anonymous Cormorant 7mth #984bafb

nvm, it was included in the question



♡ ...

Anonymous Mink 7mth #984baed ✓ Resolved

SP23-Final-Q4.2



Each cache stores 16bit addresses which means that it doesn't even store a full integer. Does this affect our analysis or how exactly is anything being stored in the cache? like each cache entry stores 16bits but were working with integers so it can't even store a full integer or what does it store

♡ ...

 **Anonymous Penguin** 7mth #984bbce  **ENDORSED**

The 16 bit address is not the actual data that is stored. The cache has 16B blocks meaning each "row" in the cache contains 16 bytes, so up to 4 integers can be stored. The address refers to where in memory our data is stored, not the data itself, and is used to determine where in the cache data should be stored (based on the tag, index, and offset bits).

♡ ...

 **Anonymous Cormorant** 7mth #984badf 

spring 23. final. Q4.2. Why does thrashing happens in a and b, but not in result?

Q4.2 (2.5 points) What is the overall hit rate for a call to `convolve_1d`? No need to simplify the fraction.

Solution: $\frac{2}{15}$

There are a total of three iterations of the outer loop, and two iterations of the inner loop per iteration of the outer loop. For each iteration of the inner loop, there are two memory accesses: reading `b[j]` and reading `a[i + j]`. Each outer loop has an additional memory access: writing to `results[i]`. In total, there are 5 accesses per iteration of the outer loop, and 15 accesses overall.

For each set of accesses to `a` and `b`, the program experiences thrashing since the index of these blocks conflict. As a result, the accesses to `a` and `b` will always be misses (first two compulsory, then the rest are conflict).


The accesses to `results` do not cause any thrashing, so out of the three accesses (one per iteration of the outer loop), there is one compulsory miss and two hits, giving us a hit rate of $\frac{2}{15}$.

♡ ...

 **Anonymous Penguin** 7mth #984bbcd

`a` and `b` have the same index bits (calculate this from their given addresses), so they'll always be competing for the same spot in the cache since it's direct mapped. `result` doesn't run into any problems because it has different index bits and nothing else in the problem competes with it.

♡ ...

 **Anonymous Moose** 7mth #984badd 

SP23-Final-Q3

Why is `3.1 #pragma omp parallel` and not `#pragma omp parallel for`?

♡ ...

 **Wesley Kai Zheng** 7mth #984bade

If I remember correctly, the question explicitly mentioned to not use `#pragma omp parallel for`.

♡ ...

 **Anonymous Partridge** 7mth #984badb 

SP23-Final-Q3

```
1 double matrix_average(double** matrix, int num_rows, int num_cols) {
2     double global_sum = 0.0;
3
4     _____ Q3.10
5     {
6         int num_threads = omp_get_num_threads();
7         int thread_num = omp_get_thread_num();
8
9         int chunk_size = _____; Q3.11
10
11        int start_row = _____; Q3.12
12
13        int end_row = _____; Q3.13
14
15        for (int i = start_row; i < end_row; i++) {
16            double row_sum = 0.0;
17            for (int j = 0; j < num_cols; j++) {
18                row_sum += matrix[i][j];
19            }
20
21            _____ Q3.14
22
23            _____ Q3.15
24        }
25    }
26    return global_sum / (num_rows * num_cols);
27 }
```

Why is #pragma omp critical necessary in line 16 if we are just adding to a total global sum?



J J **Jedidiah Tsang** STAFF 7mth #984bafd

It's precisely that we are adding to a total global sum that gives us reason to serialize the addition (pragma omp critical). Otherwise, we could get data races if a thread context switches before storing the modified value back in the global sum variable.



Anonymous Dragonfly 7mth #984bada

✓ Resolved

sp23-Final-Q7

I looked through the Ed threads related to this and I mostly understand except why do we do *codecopy = 0 and then in the while loop also check if *codecopy == 0? Also why do we increment it in two places?

(also for the return could we do n-num for the length excluding branch/jump instructions)?



T **Tyler Yang** STAFF 7mth #984bdbe

We check if *codecopy != 0 to make sure that we are hitting a valid instruction (may not be branch or jump but no instruction will have a value of 0). We do *codecopy = 0 when we either hit the end or we hit a branch instruction so that we can end the "previous data array".

We also iterate twice but for different reasons. Inside the while loop, we iterate because we only want to set a branch/jump instruction to 0. After we set this instruction to 0, we want the new array to start **after** the branch or jump so we iterate once again before we go back to the beginning of the for loop.

As for the return, I don't think n - num would work. We are specifically looking for the length of the result array which should always be 1 + the number of branch/jump instructions. (As a visualization, I like to consider each branch instruction as a "divider". These dividers "split up" the result array. (ie. if an element of result is represented as - and a branch instruction is

represented as | , if we had 5 branch instructions then the result array would contain 6 elements (-|-|-|-|-). I know this is a bit jank but let me know if you need a further clarification!

♡ 1 ...



Anonymous Dragonfly 7mth #984bacd

✓ Resolved

Sp23-Final-Q4.2 4.4

Why would this lead to thrashing? Why does result not cause a compulsory miss? Why are there three iterations of the outer loop (shouldn't it be 0 to <2 meaning 2 iterations and 10 accesses total)?

♡ 1 ...



Andrew Liu STAFF 7mth #984bedb

For the for loop iteration, notice that $a_len - b_len + 1 == 3$. Result causes 1 compulsory miss due to being loaded from a cold cache, then just stays in the cache afterwards. Thrashing happens due to conflict misses between a and b.

♡ ...



Anonymous Barracuda 7mth #984bacc

✓ Resolved

FA23-Final-Q2.1+2.2

I am a bit confused by why the signal is not constant for PCSel and RegWen. Could you explain maybe what the signals might look like? Thank you!

Q2.1 (1.5 points) PCSel

- PC + 4 Doesn't matter
 ALUOut None of the above

Solution: The value of PCSel is determined by the output of the branch comparator, so it is not a constant, therefore "None of the above".

Q2.2 (1.5 points) RegWen

- Write disabled Doesn't matter
 Write enabled None of the above

Solution: The value of RegWen is determined by the output of the branch comparator (since we only update rd if the branch is taken), so it is not a constant, therefore "None of the above".

♡ ...



Justin Yokota STAFF 7mth #984bbdd

For example, you can't tell what the value of PCSel should be if I just tell you that an instruction is a "beq". This is in contrast to many signals; for example, you can tell what ASel should be if I tell you that an instruction is a "sub".

♡ 2 ...



Anonymous Cormorant 7mth #984bacb

✓ Resolved

spring 23/ final/ Q3.

Why are we using chunk_size, start_row and end_row here? I don't understand the logic behind these calculations.

Thank you!

Parallelize matrix_average using OpenMP without using #pragma omp parallel for or reduction. Each thread should work on one or more rows of the matrix. Assume num_rows is a multiple of num_threads.

```
1 double matrix_average(double** matrix, int num_rows, int num_cols) {
2     double global_sum = 0.0;
3
4     _____ Q3.10
5     {
6         int num_threads = omp_get_num_threads();
7         int thread_num = omp_get_thread_num();
8
9         int chunk_size = _____; Q3.11
10
11        int start_row = _____; Q3.12
12
13        int end_row = _____; Q3.13
14
15        for (int i = start_row; i < end_row; i++) {
16            double row_sum = 0.0;
17            for (int j = 0; j < num_cols; j++) {
18                row_sum += matrix[i][j];
19            }
20
21            _____ Q3.14
22
23            _____ Q3.15
24        }
25    }
26    return global_sum / (num_rows * num_cols);
27 }
```

Solution:

```
Q3.10: #pragma omp parallel
Q3.11: num_rows / num_threads
Q3.12: thread_num * chunk_size
Q3.13: (thread_num + 1) * chunk_size
Q3.14: #pragma omp critical
Q3.15: global_sum += row_sum
```

♡ ...

 **Nikhil Kandkur** STAFF 7mth #984bbec

We need to use a #pragma omp parallel statement, so because of this, we need a way to partition our matrix such that each thread works on a separate portion of the matrix. The simplest way to partition our matrix is such that each thread works on a certain number of rows. That's why our chunk_size is going to be the number of rows each thread works on, and start_row/end_row are used to signify which rows bound the chunk the thread works on.

♡ ...

 **Anonymous Red panda** 7mth #984babf ✓ Resolved

SP23-final Q7)

For the line 7.8 where data[i] = codecopy, wouldn't this assign the first line of instruction to automatically the line of code even when its a branch/jal instruction and should be 0?

♡ ...

T Tyler Yang STAFF 7mth #984bdbb

Technically, yes, however since `codecopy` is a ptr, we actually modify the value of this in 7.12 to 0. Thus `data[0]` will point to 0.

♡ 1 ...

Anonymous Cormorant 7mth #984babe

✓ Resolved

spring23. final. Q2.1

How do we know which path is the longest path when calculating the min clock period? And why regfile setup and regfile read is not included?

Q2 IF Only ID Pipelined Better

(10 points)

In Project 3, we implemented a RISC-V CPU with two stages; stage 1 included IF and stage 2 included ID/EX/MEM/WB. For this question, imagine instead that we implement a two-stage pipeline with a different split; stage 1 will include IF/ID and stage 2 will include EX/MEM/WB (IF/ID/EX/MEM/WB are defined equivalently to the pipelined CPU on the reference card).

For Q2.1 and Q2.2, assume the following delays for each component. Any component not listed is assumed to have a negligible delay.

Component	Delay
$\tau_{\text{clk-to-q}}$	35ps
τ_{setup}	20ps
Mux	75ps
Regfile Setup	20ps
Regfile Read	175ps
Immediate Generator	150ps
Branch Comparator	200ps
ALU	200ps
Memory Read	300ps

Q2.1 (3 points) What is the minimum clock period of this circuit, in picoseconds, to achieve correct behavior?

Solution: 855ps ($\tau_{\text{clk-to-q}}$ (35) + Immediate Generator (150) + Mux (75) + ALU (200) + Memory Read (300) + Mux (75) + τ_{setup} (20)).

The critical path occurs in stage 2 for a load instruction.

Grading: Partial credit was given for errors that showed conceptual understanding of what the critical path is, but excluded or included an extra component's timing. We did not give partial credit for excluding some components since we cannot clearly distinguish between a conceptual misunderstanding or a mechanical error.

♡ ...

N Nikhil Kandkur STAFF 7mth #984bbee

We would need to calculate the longest path in each stage in order to figure out the min clock period: the min clock period is the time it takes for the longest stage. Regfile read is not included because ID is not included in the stages we use in our critical path, and Regfile setup = t_{setup} , so either value could be used to signify setup time.

♡ ...

Anonymous Oyster 7mth #984babc

✓ Resolved

SU23-Final-Q3.11

Why would we need to skip to PC + 8 if we don't take the jump/branch?

♡ ...

Anonymous Penguin 7mth #984bbcc

If we don't take the jump/branch we would always need to skip to PC + 8 in order to move to the next instruction (since long jumps and long branches take up 8 bytes in IMEM, 4 from the instruction itself and 4 more from the immediate)

♡ 1 ...



Anonymous Rook 7mth #984babb

✓ Resolved

sp23-Final-Q4.4

I've seen a couple people ask similar questions regarding this one but not exactly with regards to what I'm not understanding. The solution using $i=4$ which is the fifth iteration, while the question asks to use the last iteration, which would be $i=5$. At $i = 4$, the data in the cache would be `arr[0]`, `arr[1]`, `arr[2]`, `arr[3]`, which leads to a hit rate of $7/12$ as we get five misses when we replace `arr[0]` with `arr[4]`, replace `arr[1]` with `arr[0]`, replace `arr[2]` with `arr[1]`, replace `arr[3]` with `arr[2]`, and replace `arr[0]` with `arr[3]`, as described in the solution. This leaves us with a data consisting of `arr[4]`, `arr[3]`, `arr[1]`, and `arr[2]` by the time we start the next and last iteration, which is the iteration the solution is asking us to solve for. My understanding would then be:

Data, LRU:

`arr[4]`, 0

`arr[3]`, 1 \rightarrow `arr[2]`

`arr[1]`, 2 \rightarrow `arr[3]`

`arr[2]`, 3 \rightarrow `arr[5]`

Read `arr[5]`. This evicts `arr[2]`, resulting in a miss.

Read `arr[0]`. This results in a hit

Write to `arr[5]`. This results in a hit.

Read `arr[5]`. This results in a hit.

Read `arr[1]`. This results in a hit.

Write to `arr[5]`. This results in a hit.

Read `arr[5]`. This results in a hit.

Read `arr[2]`. This evicts `arr[3]`, resulting in a miss.

Write to `arr[5]`. This results in a hit.

Read `arr[5]`. This results in a hit.

Read `arr[3]`. This evicts `arr[1]`, resulting in a miss.

Write to `arr[5]`. This results in a hit

This would give us a hit rate of $9/12$, why is this wrong? Am I not adjusting the LRU numbers correctly? Just to confirm, the LRU numbers update whenever we access something or only when we replace something in the cache? Thanks!

♡ 1 ...



Andrew Liu STAFF 7mth #984bedc

The reason is that each block is 16B, meaning that it holds 4 ints per block. Therefore, the cache will look something like

`a[0]` to `a[3]`

`b[0]` to `b[1]` plus some other data

`results[0]` to `results[2]` plus some other data

unused block

♡ ...



Anonymous Dragonfly 7mth #984baae

✓ Resolved

sp23-Final-Q2.1 and 2.2

Q2.2: Why do we not include the Regfile Setup here as part of the writeback stage (after the last mux)?

Q2.3: I saw from other answers that ImmGen was moved to be part of the EX stage making 2.2's answer ImmGen. Since ImmGen is not part of the EX stage this semester, would our corresponding answer be DMEM or not possible?

♡ ...

 N **Nikhil Kandkur** STAFF 7mth #984bbef

We could use regfile setup, but that happens to be the same as t_setup, and there is only one setup value needed following the writeback stage. For 2.3, I'd argue not possible since we cannot parallelize any operations within our datapath further.

♡ 1 ...

 **Anonymous Dragonfly** 7mth #984bbfa

Is it always the cause in the datapath that regfile set up is the same as t setup?


♡ ...

 N **Nikhil Kandkur** STAFF 7mth #984bbfb

↩ Replying to Anonymous Dragonfly

Not necessarily, and in the case where they are not the same, I believe you would use the regfile setup since that is the only setup time needed for leaving the writeback stage.

♡ ...

 **Anonymous Penguin** 7mth #984baad ✓ Resolved

SP23-Final-Q2.1

Why didn't we need to consider regfile setup

Q2 IF Only ID Pipelined Better**(10 points)**

In Project 3, we implemented a RISC-V CPU with two stages; stage 1 included IF and stage 2 included ID/EX/MEM/WB. For this question, imagine instead that we implement a two-stage pipeline with a different split; stage 1 will include IF/ID and stage 2 will include EX/MEM/WB (IF/ID/EX/MEM/WB are defined equivalently to the pipelined CPU on the reference card).

For Q2.1 and Q2.2, assume the following delays for each component. Any component not listed is assumed to have a negligible delay.

Component	Delay
$\tau_{\text{clk-to-q}}$	35ps
τ_{setup}	20ps
Mux	75ps
Regfile Setup	20ps
Regfile Read	175ps
Immediate Generator	150ps
Branch Comparator	200ps
ALU	200ps
Memory Read	300ps

Q2.1 (3 points) What is the minimum clock period of this circuit, in picoseconds, to achieve correct behavior?

Solution: 855ps ($\tau_{\text{clk-to-q}}$ (35) + Immediate Generator (150) + Mux (75) + ALU (200) + Memory Read (300) + Mux (75) + τ_{setup} (20)).

The critical path occurs in stage 2 for a load instruction.

Grading: Partial credit was given for errors that showed conceptual understanding of what the critical path is, but excluded or included an extra component's timing. We did not give partial credit for excluding some components since we cannot clearly distinguish between a conceptual misunderstanding or a mechanical error.

♡ ...

S **Skyler Chan** 7mth #984bbbf

I believe it's because for a load instruction, we're not writing anything to the regfile (only reading), so we don't need to wait for the regfile to "setup" (time to reach rising edge of the clock)

♡ ...

Anonymous Zebra 7mth #984affe ✓ Resolved

For question 6, isn't the task of sending a message to the manager in the main loop, according to the concept of the lecture, the worker node does not perform any task outside of the loop

Q6 Mixed Messages

(7 points)

A new data center has to perform $2N$ tasks using its C cores. These tasks, indexed 0 to $2N - 1$ inclusive, are independent, except that each task $N + i$ must be done after task i has been completed. We decide to use the manager-worker approach with one process per core, where the manager keeps track of a queue `task_queue` containing all unassigned tasks that can be started. We'll use process 0 as the manager.

Throughout this question, you must minimize the resources consumed by the data center:

- Do not send unnecessary messages.
- Terminate unused processes as soon as possible.

Assume that each process enters the main loop simultaneously, and that processes will not crash during execution. We implement this task with the following messages:

- `EXECUTE(x)`, where $0 \leq x < 2N$.
- `EXIT`
- `DONE(x)`, where $-1 \leq x < 2N$. `DONE(-1)` indicates that a worker is ready but has not performed a task.

A worker should have the following behavior:

Q6.1 (1 point) Before the main loop...

- Perform task 0
- Cleanup and exit process
- Send the `DONE(-1)` message to process 0
- Do nothing

Solution: Since `DONE(-1)` is defined to indicate that a worker is ready but has not performed a task, we send this message when a worker is about to start.


```
CS 61C
Set up
While True:
    Send to the manager "I'm ready for more work"
    Receive message from manager
    If message is "Here's more work":
        Do the work
    Else if message is "All work done":
        break
```

♡ 1 ...

 **Andrew Liu** STAFF 7mth #984beda

This behavior is specified in the question and so its slightly different from the pseudocode seen in lecture.

♡ ...

 **Anonymous Mandrill** 7mth #984affd ✓ Resolved

For FA 23 Q10, could you construct an equivalent solution with the arrows almost in the other direction because you instead consider the first digit of states as meaning $N-1$ and second digit as meaning $N-2$. The input/output would also be switched then

♡ ...

 **Abhinav Vedati** STAFF 7mth #984bdea

Sure, the state labels by themselves don't have any function.

♡ ...

D **Daichi Watanabe** 7mth #984affc ✓ Resolved

Sp23-Final-Q7

By the end of the function, does every int array (of type `int*`) in the data variable have every element it needs **and** every element after it in the original array (with branches and jumps set to

0)? I don't see where we null-terminated each of our strings, because all we did was increment through the addresses of codecopy and set the element at the end of each array to 0.

♡ ...

T Tyler Yang STAFF 7mth #984babd

The original array does not need to be modified. The solution makes it such that each pointer within the data "array" will point to the first instruction after a branch instruction (in codecopy). Since we change all branch and jump instructions (and after the codecopy array at the very end) to 0, every single int* in the data variable will be correctly null terminated and also have the proper elements.

♡ ...

D Daichi Watanabe 7mth #984baf6

I'm sorry, I still don't understand. For example, suppose this was the input:

```
input = [add a0 t0 t1, add a0 t0 t1, beq x0 x0 pass, add a0 t0 t1, beq x0 x0 pass]
```

This is what the problem says will be in the output:

```
[  
  [add a0 t0 t1, add a0 t0 t1, 0],  
  [add a0 t0 t1, 0],  
  [0]  
]
```

But would it be that, or would the actual arrays have the following?

```
[  
  [add a0 t0 t1, add a0 t0 t1, 0, add a0 t0 t1, 0],  
  [add a0 t0 t1, 0],  
  [0]  
]
```

Because by the end of the function, the original (pre-incrementation) address of codecopy is pointing to an array of these values:

```
[add a0 t0 t1, add a0 t0 t1, 0, add a0 t0 t1, 0] (is that not right?)
```

If so, does changing a value in an integer array to 0 count as null-terminating it there? If so, this seems strange because I don't understand how integer arrays could then carry the value 0 somewhere in the middle without it meaning null-termination. For example, the array [1, 2, 3, 4, 0, 5, 6, 7, 8] would be null-terminated at [1, 2, 3, 4].

♡ ...

T Tyler Yang STAFF 7mth #984bbaa

↩ Replying to Daichi Watanabe

For this specific question, the instructions for result state Each of these arrays should be "null-terminated"; that is, the last element of each array should be the number 0, to signify the end of the array.

In practical purposes, a 0 in an integer array does not carry any meaning.

♡ 1 ...

D **Daichi Watanabe** 7mth #984bbac

↩ Replying to Tyler Yang

Thanks 🙏

♡ 1 ...



Anonymous Albatross 7mth #984affb

✓ Resolved

FA23-Final-Q5.2

Would it be ok to set this to 0 and then at 5.5 use the `vec_or` operation, I think this works as this would just count all the odd numbers and exclude the even numbers but want to make sure.

may only use up to one SIMD operation per blank.

```
1 bool more_even_simd(uint32_t* array, uint32_t n) {
2     vector counts = vec_setnum(0); // Q5.1
3     vector mask = vec_setnum(0); // Q5.2
4     for (uint32_t i = 0; i < n / 4 * 4; i+=4) { // Q5.3
5         vector temp = vec_load(array + i); // Q5.4
6         vector masked = vec_or(mask, temp); // Q5.5
7         counts = vec_add(counts, masked); // Q5.6
8     }
9     return (n / 2 > // Q5.7
10         counts); // Q5.8
11 }
```

♡ ...



A **Abhinav Vedati** STAFF 7mth #984bdeb

No, and let's examine a specific number to see why. To simplify we'll imagine our vectors are all length 1 for now (i.e. they are scalars). Suppose that we load in the number 4 from our array. This is even, and thus 0 should be added to our total count. However, if we bitwise-or 2 with 0 (0b10 | 0b00), we get 2 (0b10) as our result, which is then added to our total count. Similarly, if we load in the number 3 from our array, we should add 1 to our total count. However, we when bitwise-or 3 with 0 (0b11 | 0b00), we get 3 (0b11) as our result, which is then added to our total count. Bitwise-anding with 1 works because there are infinitely many implicit zeros to the left of the single 1, so all higher order bits are zeroed out during the bitwise-and, and all that's left is the least significant bit from the loaded in number.

♡ 1 ...

 **Anonymous Albatross** 7mth #984bdec

Ohh thanks, i keep forgetting that we extend out bits when doing bitwise operations.
Thanks for the explanation!

♡ 1 ...

A **Abhinav Vedati** STAFF 7mth #984bdef

↩ Replying to Anonymous Albatross

No worries!

♡ ...

 **Anonymous Badger** 7mth #984afef ✓ Resolved

FA23-Final-Q1.3


Will we be given whether it is a local or global hit time?

♡ ...

A **Abhinav Vedati** STAFF 7mth #984bded

Can you clarify what "it" is? When we ask for the "average memory access time of this system", we are asking for the overall hit time including possible accesses to the L1, L2, and DRAM with their associated probabilities. This is a "global" hit time in a sense. If you are referring to "the hit time for this L2 Cache", then that can be thought of as a "local" hit time that only applies to the L2 cache.

♡ ...

 **Anonymous Fox** 7mth #984afce ✓ Resolved

FA23-Final-Q3.1

Why is there a WB for instruction 8 in clock cycle 12?

♡ ...

A **Abhinav Vedati** STAFF 7mth #984bdfa

This is a typo, good catch!

♡ ...

 **Anonymous Meerkat** 7mth #984afcc ✓ Resolved

SP23-Final-Q5.2

Why is the last hex digit of the VPN = to the index of the corresponding PTE?

♡ ...

A **Anonymous Fish** 7mth #984bddc


Were you able to figure it out?

♡ ...

A **Abhinav Vedati** STAFF 7mth #984bdee

This is because the entire VPN indexes our single-level page table, and the other hex digits happen to be 0s.

♡ ...

 **Anonymous Albatross** 7mth #984afcb ✓ Resolved
FA23-Final-Q2.7


Why do we not need to add an ALU operation, don't we need to do $rs1 + imm * 4$ just like in sp23 Q1, since we are doing $*(rs1 + imm)$ which in pointer arithmetic would be $rs1 + imm * sizeof(instruction)$?

♡ ...

 **T Tyler Yang** STAFF 7mth #984afcd

I don't think this is question 3.1 of FA23 - could you clarify which question it is?

♡ ...

 **Anonymous Albatross** 7mth #984afcf
sorry just edited it!


♡ ...

 **T Tyler Yang** STAFF 7mth #984afda

↩ Replying to Anonymous Albatross

In FA23, we know $rs1 + imm$ is the correct address. In SP23 we were looking to grab indices of an integer array and hence why we needed to multiply the **index of the array** by 4.

♡ 1 ...

 **Anonymous Albatross** 7mth #984afde
↩ Replying to Tyler Yang

Oh so the we only need to that type of pointer arithmetic with arrays, but in this case since its not an array then we do not have to do that?


♡ ...

 **T Tyler Yang** STAFF 7mth #984afdf

↩ Replying to Anonymous Albatross


More specifically since it was an integer array, each element is 4 bytes.

♡ 1 ...

 **Anonymous Albatross** 7mth #984afec
↩ Replying to Tyler Yang

Thanks this really helped!

♡ 1 ...

 **Anonymous Rhinoceros** 7mth #984afbd ✓ Resolved
FA23-Final-Q3.1

For the solution explaining all of the instructions in the table, why is there WB written instead of an X on the last nop for line 8 on the 12th clock cycle? ~~Also does the reference-card datapath have the ability to use forwarding?~~

Q3.1 (3 points) If all hazards are resolved through stalling (no double pumping or forwarding paths), during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X	X				
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2								IF	ID	EX	MEM	WB		
9. xori t1 x0 6									IF	ID	EX	MEM	WB	

♡ ...

 **Andrew Liu** STAFF 7mth #984beea

That is a typo in the solutions — nice catch

♡ ...

A **Adrian Bao** 7mth #984afba ✓ Resolved

SP23-Final-Q2.1


Why do we not stop at DMEM when finding the max combinational path? It is a register and I am not sure why we consider the writeback mux after that. Thanks!

♡ ...

 **T Tyler Yang** STAFF 7mth #984bbad

DMEM is only a state element if we are writing to it. If we follow the datapath for a load instruction, we are only reading from memory so this does not count as an endpoint for the combinational path (hence why we can continue to the wb mux after).

♡ ...

 **Anonymous Cormorant** 7mth #984aefb ✓ Resolved

fall 23. q8. final

Can someone explain to me what does line 2-6 is doing? I understand that it is trying to compute $x \% 4$, but i don't understand what `aupic`, `jalr`, `t3` and `t7` is doing?

Please help!!

Thank you in advance!

Q8 Cumulative: Unconditional RISC**(9 points)**

Assume we have a function, f , that takes in a 32-bit unsigned integer, x , as an argument. $f(x)$ is defined as follows:

$$f(x) = \begin{cases} x + 9 & \text{if } x \% 4 == 0 \\ x * 2 & \text{if } x \% 4 == 1 \\ x & \text{if } x \% 4 == 2 \\ x // 8 & \text{if } x \% 4 == 3 \end{cases}$$

Jero wants to write this function in RISC-V, but he couldn't get his CS61CPU's branch instructions to work! As a result, you may use any RV32I instruction **except** branch instructions.

Write a function, f , which accepts one argument x in $a0$, and returns $f(x)$. You may assume that there is no overflow.

```

1 f:
2  andi t3 a0 3
   Q8.1
3  slli t3 t3 3
   Q8.2
4  auipc t7 0
   Q8.3
5  add t3 t3 t7
6  jalr x0 t3 12
   Q8.4
7  addi a0 a0 9
   Q8.5
8  j f_end
9  slli a0 a0 1
   Q8.6
10 j f_end
11 nop
   Q8.7
12 j f_end
13 srli a0 a0 3
   Q8.8
14 f_end:
15 ret

```

♡ 2 ...



Anonymous Kingfisher 7mth #984aefd

From what I understood, the jal instruction is there so that you can jump to the appropriate instruction (based on the output of $x \% 4$) and t3 stores the appropriate offset. auipc is used to retrieve the PC value so that you can use it along with the offset stored in t3 in your jal instruction to jump to the correct instruction.

♡ ...



Daichi Watanabe 7mth #984aefa

✓ Resolved

Sp23-Final-Q7

I'm okay with everything on here except for one thing; I don't quite understand why blank 7.13 is ***result = data;** instead of **result = &data;**

(Edit: also, is 7.7 supposed to say **i < num + 1** or **num + 1** like on the markscheme?)

```
1 // Returns true if instruction is a branch or jump instruction
2 bool isBranchJump(int instruction) {
3     return _____;
4 }
5
6 int splitCode(int* code, int n, int*** result) {
7     int num = 0; // total number of branches and jumps
8     for(int i = 0; _____; i++) {
9         num += _____;
10    }
11    int** data = malloc(_____);
12    int* codecopy = calloc(n+1, _____);
13    // Hint: You should not need any more memory allocations
14    memcpy(codecopy, code, _____);
15    for(int i = 0; _____; i++) {
16        data[i] = _____;
17        while(_____ && _____ != 0) {
18            _____;
19        }
20        _____;
21        codecopy++;
22    }
23    _____;
24    return _____;
25 }
```

♡ ...

E Erik Yang STAFF 7mth #984afaf
i < num + 1 is the right answer - there was a typo. result is a triple pointer that is passed into the function, so that memory already exists - you can just dereference that pointer.

♡ 1 ...

D Daichi Watanabe 7mth #984afee

Thanks 🙏

♡ ...

Anonymous Mink 7mth #984aeef ✓ Resolved

FA23-Final Q10 answer key - possible errors?

Shouldn't the highlighted section be 2^{12} ?

Q10.3 (2 points) What is the fewest number of states that an FSM solving this problem can have? Your answer must be an exact **integer**.

Solution: 4096. To get this answer, notice that to reference something from 12 inputs ago in an FSM, we must keep a record of that for at least 12 states. In other words, we must "encode" 12 bits of memory into our FSM states, requiring $2^{12} = 4096$ states.

Why is it 25ns? Shouldn't it be 25ps?

Q10.4 (2 points) What is the minimum clock period of any circuit that solves this problem (assuming the register is expanded to sufficiently many bits without increasing clk-to-q and setup times)?

Solution: 25ns. Since the circuit will look exactly the same as the solution above, but extended to 12 bits, (Input \rightarrow Reg[0], Reg[0] \rightarrow Reg[1], ..., Reg[11] \rightarrow Output) the minimum combinatorial path will be 0s, (either from Input to D via a splitter, or Q to Output via a splitter) so our minimum clock cycle equals $t_{\text{clk-to-q}} + 0 + t_{\text{setup}} = 25\text{ns}$.

Q10.2 (4 points) Fill in the circuit diagram below to implement this FSM. For full credit, your circuit must have the minimum possible clock period, assuming the following component delays:

$$t_{\text{AND gate}} = 12\text{ps}$$

$$t_{\text{OR gate}} = 15\text{ps}$$

$$t_{\text{NOT gate}} = 4\text{ps}$$

$$t_{\text{XOR gate}} = 31\text{ps}$$

$$t_{\text{Register clk-to-q}} = 10\text{ps}$$

$$t_{\text{Register setup}} = 15\text{ps}$$

$$t_{\text{Bit splitter}} = 0\text{ps}$$

$$t_{\text{Wire}} = 0\text{ps}$$

You may not use any other components. You may assume that the input and output connect directly to registers (for the purpose of determining the clock period), and that the register stores 2 bits. Your circuit does not need to "match" the states you use in your answer to Q10.1; it will be considered correct if its behavior matches the intended behavior described above.

♡ ...



E Erik Yang STAFF 7mth #984afad

Yup these look like typos -thanks for noting this!

♡ ...



Anonymous Cormorant 7mth #984aeea

✓ Resolved

fall23. final. Q4.1

What value did we plug in the index formula to get 3?

For Q4.1, assume that we have a 32-bit address space with a 32KiB, 8-way associative cache with a block size of 512B.

Q4.1 (3 points) Calculate the TIO bits with this cache setup.

Solution: 20/3/9

The offset equals $\log_2(\text{block size}) = 9$. Then, the index equals $\log_2\left(\frac{\text{cache size}}{\text{block size} \cdot \text{associativity}}\right) = 3$. Finally, the tag is equal to address length $-$ index $-$ offset = 20.

♡ ...



C Catherine Van Keuren STAFF 7mth #984aeee

You won't be tested on n-way associative caches since the weren't taught this semester, but you would plug in $\log_2((32\text{KiB} = 32 * 2^{10} = 2^{15}) / ((512\text{ B} = 2^9) * (8 = 2^3) = 2^{12})) = \log_2(2^3) = 3$

♡ ...



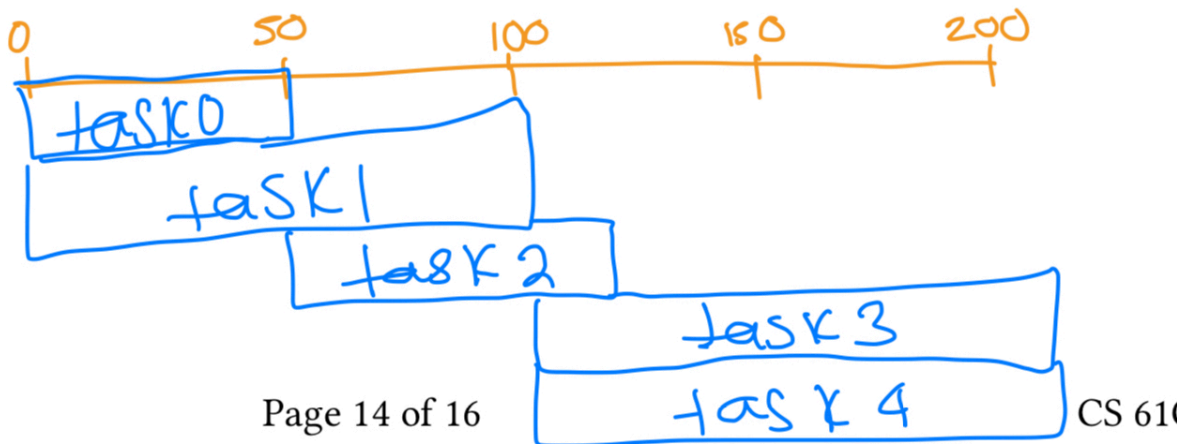
Anonymous Beaver 7mth #984aede

✓ Resolved

SP23-Final-Q8.3

Why is it 200 minutes? I'm currently getting 250 mins since you can only be as fast as your slowest performance. We can do task 0 and 1 in the first task and since I chose chipA (100 mins) and chipB (50 mins), it will take 100 mins for that portion. Then we perform the rest of the tasks: task 2 using chipA (100 mins), task 3, using chipB (150 mins), and task 4 using chipC (60 mins). So, in this section, doesn't it take 150 mins because that one is the slowest?

Reading the solutions, it looks like task 0, 1, and 2 get grouped together, but how can we do that if task 2 requires task 0 to be done first? Should I picture it kind of like this where as soon as task 0 is done, we run task 2 since it's not dependent on task 1 and won't add time since task 3 and task 4 need to still be completed?



♡ 1 ...



Erik Yang STAFF 7mth #984afac

Chip A does task 1 from $t=0$ to $t=100$, task 3 from $t=100$ to $t=200$ (task 0 is finished by $t=50$).

Chip B does task 0 from $t=0$ to $t=50$)

Chip C does task 2 from $t=50$ to $t=110$, task 4 from $t=110$ to $t=170$.

♡ ...



Anonymous Penguin 7mth #984bafa

Hi Erik, how do you recommend approaching these kinds of problems? I understand that here it's better to put B on tasks that require more memory operations and C on tasks that require more math operations, but other than that isn't it mostly just playing around with things to see what works? Is there a better strategy?

♡ 1 ...



Erik Yang STAFF 7mth #984bcfc

↩ Replying to Anonymous Penguin

I'd say some sort of greedy algorithm would be a good way to start

♡ ...



Anonymous Cormorant 7mth #984aedd

✓ Resolved

fall23. final. Q3.1

Why does line 5 and 6 is not included in the timing diagram? Also, I'm kinda confused, what is the answer to this question?

```

1  beq x0 x0 Label
2  addi x0 x0 3
3  addi x0 x0 1
4  addi x0 x0 4
5  addi x0 x0 1
6  addi x0 x0 5
Label:
7  addi t0 x0 9
8  addi t1 t0 2
9  xori t1 x0 6

```

Suppose that the IF stage of the `beq` on line 1 occurs during cycle 1.

Q3.1 (3 points) If all hazards are resolved through stalling (no double pumping or forwarding paths), during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X	X				
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2									IF	ID	EX	MEM	WB	
9. xori t1 x0 6										IF	ID	EX	MEM	WB



T Tyler Yang STAFF 7mth #984afaa

5 and 6 are not included in the timing diagram because we jump to Label before those instructions can enter into the pipeline. (Remember that PCsel will be determined in the MEM stage). Thus after instruction 1 completes MEM, then instruction 7 is ran. Considering the hazards as well as the timing diagram, `xori` on line 9 will execute its WB phase on cycle 14. Thus 14 would be the correct answer.



Anonymous Anteater 7mth #984afbe

Following up on this, why is line 8 repeated 3 times with a nop? Also, would the fact that the operations are being stored in x0 change this behavior in any way, since x0 is hardwired 0 no matter what?



T Tyler Yang STAFF 7mth #984afbf
 ↩ Replying to Anonymous Anteater
 The data hazard is resolved through stalling, so 8 is stalled 3 times.
 ♥ ...

Anonymous Dotterel 7mth #984befb
 ↩ Replying to Tyler Yang
 Hi, why do lines 3 and 4 not need stalling? Is it because the register is x0?
 ♥ ...

T Tyler Yang STAFF 7mth #984afca
 ↩ Replying to Anonymous Anteater
 As for the second part to your question.. I don't think so. The branch will be taken no matter what registers you substitute in for x0 so the following instructions would be nop'd regardless
 ♥ ...

Anonymous Woodpecker 7mth #984bccd
 ↩ Replying to Tyler Yang
 Hi, can you please clarify where the answer 14 cycles is coming from?
 ♥ ...

T Tyler Yang STAFF 7mth #984bcdd
 ↩ Replying to Anonymous Woodpecker
 In the timing diagram, instruction 9's WB stage happens on cycle 14. (It is the bottom right corner box).
 ♥ ...

Anonymous Cormorant 7mth #984afea
 In the lecture, for stalling, the ID of the next one happens at the same cycle as the WB of the previous one.

In the question above, the IF of line 8 happens in the same cycle of WB of line 7? WHY would this be the case?

CS 61C Data Hazards, Solution 1: Stalling

time →

	IF	ID	EX	MEM	WB
add s0, t0, t1	IMEM	Reg	ALU	DMEM	Reg
sub → nop	IMEM				
sub → nop	IMEM				
sub t2, s0, t0	IMEM	Reg	ALU	DMEM	Reg
or t6, s0, t3	IMEM	Reg	ALU	DMEM	Reg

s0 value

5	5	5	5	5/9	9	9	9	9
---	---	---	---	-----	---	---	---	---

Problem: Instruction depends on WB's RegFile write from previous instruction.

- Executing `sub` immediately after and reads old value of `s0`.

Solution 1: Stall pipeline:

- Requires extra pipeline state to prevent register writes on stalled stages.
- The correct instruction is stalled for two clock cycles.

Berkeley 22 Pipeline II Hazards (11) Yan, Yokoyama



T Tyler Yang STAFF 7mth #984baec

The question specifically states that there is no double pumping so the ID needs to happen after the WB.



Anonymous Cormorant 7mth #984aedb

✓ Resolved

f23. final. q1.4

What is the overhead? (definition or concept)

Q1.4 (2 points) You have a program that takes 20 seconds to run, but you've found a way to make 80% of the code 4 times faster at the cost of some overhead. After rerunning your program, it takes 13 seconds. What is the overhead (in seconds)?

Solution: 5 seconds

80% of the 20 second program would take 16 seconds, and would take 4 seconds if we make it 4 times faster. The remaining 20% of the program, or 4 seconds, remains the same. If there was no overhead, it would take 8 seconds (4 seconds from non-parallelizable section and 4 seconds from the parallelized section). Since the overall runtime was 13 seconds, this must mean the overhead is 5 seconds.

Grading: This problem was graded on an all-or-nothing basis.



Anonymous Mink 7mth #984aeec

overhead is the time it takes to speed up a program. when doing TLP, consider the time it takes to set up the threads, and then wait for all the threads to finish. in the case of DLP, im not too sure, unfortunately, perhaps like the time to set up the vectors? idk. and for PLP, time to set up the different processes and perhaps the finalizing of the work. basically, its the 'extra' time needed to speed up a program.



Anonymous Seahorse 7mth #984aeda

✓ Resolved

sp23-final-Q7

For line 3 I understand the idea that we are trying to compare the the left most bit of the opcode with the instruction to see if the instruction is branch / jump. However, I am not sure how does instruction & 64 give us an boolean value and how we use it as int at line 9

```

1 // Returns true if instruction is a branch or jump instruction
2 bool isBranchJump(int instruction) {
3     return _____;
4 }
5
6 int splitCode(int* code, int n, int*** result) {
7     int num = 0; // total number of branches and jumps
8     for(int i = 0; _____; i++) {
9         num += _____;

```

♡ ...



Karan Dalal 7mth #984aedf

C does not have native boolean values. We use 1 to represent True and 0 to represent False. When we do arithmetic operations with C bool, we're just using 1/0.

♡ ...



Tyler Yang STAFF 7mth #984afae

From the 61C reference card, since there is no `ecall` or `ebreak` in code, then jump and branch instructions are the only instructions that have a leading 1 in the opcode. Since all instruction types have the opcode being the 0-7 bits, we can perform an `&` (and) operation between instruction and `00000000000000000000000010000000` (which is 0 at every bit except the 6th bit since this determines whether the instruction is a branch or jump). In decimal, this number is 64. Notice that if the instruction is not a jump or branch, `instruction & 64` will be 0 (`00000000000000000000000000000000`), however, if it is a jump or branch, `instruction & 64` will be 64 (`000000000000000000000000000010000000`).

When C casts to a bool, all non-zero ints are true and 0 is cast as false.

♡ 1 ...



Anonymous Seahorse 7mth #984afbc

thank you

♡ ...



Anonymous Dragonfly 7mth #984bace

does jalr count as a jump instruction here even though it is I type?

♡ ...



Tyler Yang STAFF 7mth #984bacf

↩ Replying to Anonymous Dragonfly

Yes

♡ ...



Anonymous Penguin 7mth #984baea

↩ Replying to Tyler Yang

`!(instruction ^ 96)` should also work, right?

♡ ...




Tyler Yang STAFF 7mth #984baeb

↩ Replying to Anonymous Penguin

I don't think so. Any opcode with `1100011`, `!(instruction ^ 96)` would return false. (The `11` at the end when xor'd with `00` would be `11` and thus is non-zero and returns false when `!`'d).

♡ 1 ...

 **Anonymous Llama** 7mth #984baee

↩ Replying to Tyler Yang

+can 7.11 and 7.12 be switched if we remove the "!" from isBranchJump?

```
int* codecopy = calloc(n + num + 1, sizeof(int));
```

I'm also not sure if codecopy has enough memory with calloc? shouldn't it also need num since we are putting a 0 on the array num + 1 times?

♡ ...

T **Tyler Yang** STAFF 7mth #984baff


↩ Replying to Anonymous Llama

If you mean 7.9 and 7.10 - I do believe that you can switch them.

For your second question, we are technically not putting 0 onto the array num + 1 times - rather we are replacing an already existing branch/jump instruction with 0 in codecopy.

See the comment at [#984babd](#) for more context.

♡ ...

 **Anonymous Porpoise** 7mth #984aece ✓ Resolved

Do branch instructions update PCSel during the MEM stage? Related to question 3.

♡ ...

⌋ T **Tyler Yang** STAFF 7mth #984aeff

Could you specify which exam you are referring to?

♡ ...

⌋  **Anonymous Porpoise** 7mth #984afab

Answered in [#984afaa](#)

♡ ...

 **Anonymous Weasel** 7mth #984aecd ✓ Resolved

Sp23-Final-Q1,

For 1.1, how did we now that our ALU operation that we wanted is $rs1 + rs2 * 4$? And for 1.2, for the reasoning behind checking the "add a new output to regfile for a third reg value" (since we read 3 reg values), why doesn't this same reasoning hold for the previous question 1.1?

I guess in general, is there some ways to figure out for each different choice here given an instruction? I've seen a lot of past questions but am never really sure how to tackle them..

♡ ...

⌋ T **Tyler Yang** STAFF 7mth #984aeffe

For 1.1, it is helpful to notice the pseudocode. Notice that $rs1$ points to an **integer array**. This means that when we want to access the $rs2$ index, we must add $rs2 * 4$ to $rs1$ to retrieve the desired index of the array. From here, we can use the DMEM to retrieve the value that $rs2 * 4 + rs1$ points to.

For your second question, (I assume you mean checking 1.2), the reason why 1.1 does not have you check "add a new output to regfile for a third reg value" is because readArr sets the destination register rd equal to $rs2 * 4 + rs1$. The R-type instruction format already has the ability to use 2 registers to set the destination register (ie. it has $rs1$, $rs2$, and rd).

However, 1.2 requires one to take the value of rs3 and put it into the **address** of $rs1 + rs2 * 4$. This requires us to be able to read from rs1, rs2, and rs3. (rs1 and rs2 to calculate the pointer and rs3 for the value).

The general approach I take when approaching these questions is (1) figure out what ALU operation is needed. It might be easier to see if you can write out the instruction or pseudocode they provide into some sort of arithmetic equation (addition is usually the most common operation). (2) See if any of the current instruction types can handle the new instruction (ie. R-type instructions reads from 2 registers and outputs to a destination register, while I-type instructions uses an immediate and one register and outputs to a destination register. (3) Figure out if you need to use the DMEM or not (need to load or store to memory).

Hope this helps!

♡ 1 ...



Anonymous Weasel 7mth #984aeac ✓ Resolved

SP23-Final-Q7, in lines 11, 15, 24, why are we doing $num + 1$ instead of just num ? Also, when we do `memcpy`, why are we doing $sizeof(int) * n$ instead of just $sizeof(int)$?

Could someone also explain how we are just doing `codecopy++` and `*codecopy = 0` and what that means exactly since `codecopy` is itself an array of ints?

♡ ...



Jero Wang STAFF 7mth #984aecb

For $num + 1$ and `codecopy++`, see [#984aaef](#)

We do $sizeof(int) * n$ because `codecopy` and `code` is an array of n ints, and we want to copy the entire array.

♡ ...



Anonymous Albatross 7mth #984adfe ✓ Resolved

SP23-Final_Q1

Is the reason why we do $* 4$ is because words are 4 bytes long? Also, how does the assumption of integers being 32 bits affect this problem?

♡ ...



Jero Wang STAFF 7mth #984aeca

It's $* 4$ because we're indexing into an integer array, and the assumption states that integers are 32 bits for this question.

♡ 1 ...



Anonymous Albatross 7mth #984aecc

Oh I get now the first part thanks

However I'm still confused how integers being 32 bits affect this question, are we multiplying by 4 because its 4 bytes in 32 bits? So if the integers were 16 bits we would only multiply by 2?

♡ ...



Anonymous Cattle 7mth #984aeed

↩ Replying to Anonymous Albatross

Basically this question is about pointer arithmetic. Think of $rs1$ as $\text{int}^{32} * p$, and you want $p + rs2$, which evaluates to $p + \text{sizeof}(*p) * rs2$

♡ 1 ...



Anonymous Cormorant 7mth #984adde

✓ Resolved

Why are the answers to these two none of the above?

Wouldn't PCSel should be the return value of ALUOut?

Q2 Eddy Needs a Project 3 Extension

(13 points)

For this problem, assume that we're working with the single-cycle datapath presented on the reference card.

Suppose Eddy wants to support the following instruction:

baleq rd rs1 rs2 offset (branch and link if equal)

```
if (rs1 == rs2) {  
    rd = PC + 4  
    PC = PC + offset  
}
```

For each of the following control signals, indicate the value it should have for **baleq**. If the control signal is not constant, select "None of the above".

Q2.1 (1.5 points) PCSel

- | | |
|------------------------------|--|
| <input type="radio"/> PC + 4 | <input type="radio"/> Doesn't matter |
| <input type="radio"/> ALUOut | <input checked="" type="radio"/> None of the above |

Solution: The value of PCSel is determined by the output of the branch comparator, so it is not a constant, therefore "None of the above".

Q2.2 (1.5 points) RegWEn

- | | |
|--------------------------------------|--|
| <input type="radio"/> Write disabled | <input type="radio"/> Doesn't matter |
| <input type="radio"/> Write enabled | <input checked="" type="radio"/> None of the above |

♡ ...



Jero Wang STAFF 7mth #984adef

PCSel and RegWEn should only be 1 if $rs1 == rs2$, so it doesn't fall under any of the other 3 answers.

PCSel is choosing between PC+4 and ALU, it's not the input value to the PC register.

♡ ...



Anonymous Weasel 7mth #984adcf

✓ Resolved

Sp23-Final-Q8, when it mentions a "multicore machine" do we essentially mean multithreading or like multiprocessing?

♡ ...



Jero Wang STAFF 7mth #984adfa

I don't think it matters for the context of this question, though a multicore machine can support both multithreading and multiprocessing.

♡ ...



Anonymous Weasel 7mth #984adcd

✓ Resolved

Sp23-Final-Q3:

why do we do $\text{num_cols}/4 * 4$ instead of $\text{num_rows}/4 * 4$ or $(\text{num_cols} + \text{num_rows})/4 * 4$?

Also, for line 6, why don't we want the value `matrix[i][j]`? What's the significance of adding `j` to `matrix[i]`?

♡ ...



Jero Wang STAFF 7mth #984adfb

Since this is a 2D array (instead of a 1D array stored in row major order), we can only really optimize based on the inner dimension, as the memory blocks for each row may not be next to each other. Therefore, we can only do $\text{num_cols} / 4 * 4$.

`matrix[i] + j` is the address of `matrix[i][j]`, and we need the address here, instead of the actual value.

♡ ...



Anonymous Cattle 7mth #984adbf

✓ Resolved

Q3.3 (6 points) Write a sequence of at most two instructions or pseudoinstructions that are equivalent to the `j` loop instruction.

You must use a `jalr` instruction or `jalr` pseudoinstruction in at least one of the blanks. You may not use a `jal` instruction, branch instruction, or `jal` pseudoinstruction in any of the blanks.

1	_____
2	_____

Solution:

```
1 la t0, loop
2 jalr x0, t0, 0 # jr t0
```

Grading: +3 for correct `la` usage, +3 for correct `jalr` with offset 0 (given correct usage of `la`)

Explanation: `j` loop is just `jal x0, loop`. `jal` instructions are PC-relative so to translate it to an absolute address, you have to load the address in code for the `loop` label using the `la` (load address) instruction into any register. Then, you have to use `jalr` with no return address (since `j` does not save a return address) on the address loaded into a register with no offset to execute starting from `loop`.

SU22-Final-Q3:

I propose an alternative solution:

`auipc t0, 0` (this stores current PC on line 18 onto `t0`, which is equivalent to the address of that instruction)

`jalr x0, t0, -28` (This takes the PC back to the first line of the loop, which is 7 instructions above, and according to the spec each instruction is 4 bytes)

♡ ...



Jero Wang STAFF 7mth #984adfc

Please post in the appropriate thread!

yes, it works, see [#985cab](#)



Anonymous Chimpanzee 7mth #984adbe

✓ Resolved

I am currently looking at SU23 Q8, and it looks like this is a floating point question. However, the question is titled "quantum computing" and has something called "chunk-addressable addresses". is this question something that is in scope? are some parts in scope?



Jero Wang STAFF 7mth #984adfd

Yes, this question is in scope.



Anonymous Cattle 7mth #984adbd

✓ Resolved

```
1 // Returns true if instruction is a branch or jump instruction
2 bool isBranchJump(int instruction) {
3     return _____;
4 }
5
6 int splitCode(int* code, int n, int*** result) {
7     int num = 0; // total number of branches and jumps
8     for(int i = 0; _____; i++) {
9         num += _____;
10    }
11    int** data = malloc(_____);
12    int* codecopy = calloc(n+1, _____);
13    // Hint: You should not need any more memory allocations
14    memcpy(codecopy, code, _____);
15    for(int i = 0; _____; i++) {
16        data[i] = _____;
17        while(_____ && _____ != 0) {
18            _____;
19        }
20        _____;
21        codecopy++;
22    }
23    _____;
24    return _____;
25 }
```

The answer uses instruction & 0x86 to determine B and J types.

0x86 bitwise representation = 0b1000 0110.

No opcode is 1000 011? It looks like you need to do a flip. Why, aren't C variables processed the way they are passed in? Or any variable will always have 0th bit on the left and higher positions on the right? If so then the little-endianess is reversed.

I, B, J both return 1 when and with 110 0001 as opcode placement. It doesn't filter out I type

♡ ...



Jero Wang STAFF 7mth #984adff

Wait, where is & 0x86 from? I see & 64 on the solutions.

♡ ...



Anonymous Cattle 7mth #984aeaf

Oh my bad I misread

♡ ...



Anonymous Cattle 7mth #984adba

✓ Resolved

Q5.14 (1.5 points) What is the value of $p + 1$?

(A) 0x3163 6973

(B) 0x6F6F 6373

(C) 0x7363 6F6F

(D) 0x1000 0001

(E) 0x1000 0004

(F) None of the above

Solution: $p + 1$ evaluates to $p + \text{sizeof}(p) = 0x1000\ 0000 + 4 = 0x1000\ 0004$

Fa23-Final-Q5.14

shouldn't $p + 1$ evaluates to $p + \text{sizeof}(\text{the element } p \text{ points to}) * 1$? But in the answer is $P + 1 * \text{pointer size}$. I know that in this case $\text{utype32 size} = \text{pointer size}$ which makes this explanation correct, but in general when asked the value of a pointer + n, the answer should be $p + \text{sizeof}(\text{element pointed}) * n$ I think

♡ ...



Jero Wang STAFF 7mth #984aeab

Oh yeah, sorry, should be $p + \text{sizeof}(*p)$. Thanks for the catch and we've noted it on our end.

♡ ...



Anonymous Weasel 7mth #984adac

✓ Resolved

Sp23-Final-Q4.2:

I understand that since b and a have the same index, there is some sort of overriding going on here but if we are just accessing the values at some index for arr a and b and not necessarily setting that to be something else like we do for the result array, why do we still have this competition? Also, assuming that $b[0]$ and $a[0]$ result in a compulsory miss, would $b[1]$ and $a[1]$ result in a conflict miss since we are just looking at the place in the same index aka index 0?

Also, if instead say we didn't have array a and it was simply $b[j] * b[i+j]$, even though $b[0]$ would result in the compulsory miss, would $b[1]$, $b[2]$, and just $b[3]$ result in hits afterwards since its a 16 Byte block and we have 4 ints?

♡ ...

J **Jero Wang** STAFF 7mth #984aead

Yeah, $b[1]$ then $a[1]$ would result in a conflict miss since they map to the same block. I'm not sure what you mean by "still have this competition", could you elaborate?

Yes, $b[1]$ through $b[3]$ would be hits.

♡ ...

L **Anonymous Weasel** 7mth #984beab

I guess in terms of competition I meant whenever we do like $b[0] + a[0]$ since we aren't technically replacing these values but just accessing them, would we still run into this issue of overriding? Also, if instead we had a 4 Byte block, would $b[1]$ through $b[3]$ result in misses then?

♡ ...

Anonymous Weasel 7mth #984adaa ✓ Resolved

Sp23-Final-Q2.1:

I'm a bit confused how we know what to include in our min clock period? What does having this different split have to do with calculating this? Also, on the reference card, should we be looking at the first circuit diagram or second for this question?

♡ ...

L J **Jero Wang** STAFF 7mth #984aeae

The second (pipelined) diagram is probably better for this.

The min clock period is based on the longest CL path, which is affected when we change the split because the path between any two registers change (as we're removing/adding pipeline registers).

♡ ...

L **Anonymous Weasel** 7mth #984beab

how did we know to only choose MUX+ALU+memread+MUX for the combinational longest path? why not also like the regfile steup or banch comp?

♡ ...

T **Timothy Bergal** 7mth #984acff ✓ Resolved

FA23-Final-Q3.1

Q3.1 (3 points) If all hazards are resolved through stalling (no double pumping or forwarding paths), during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

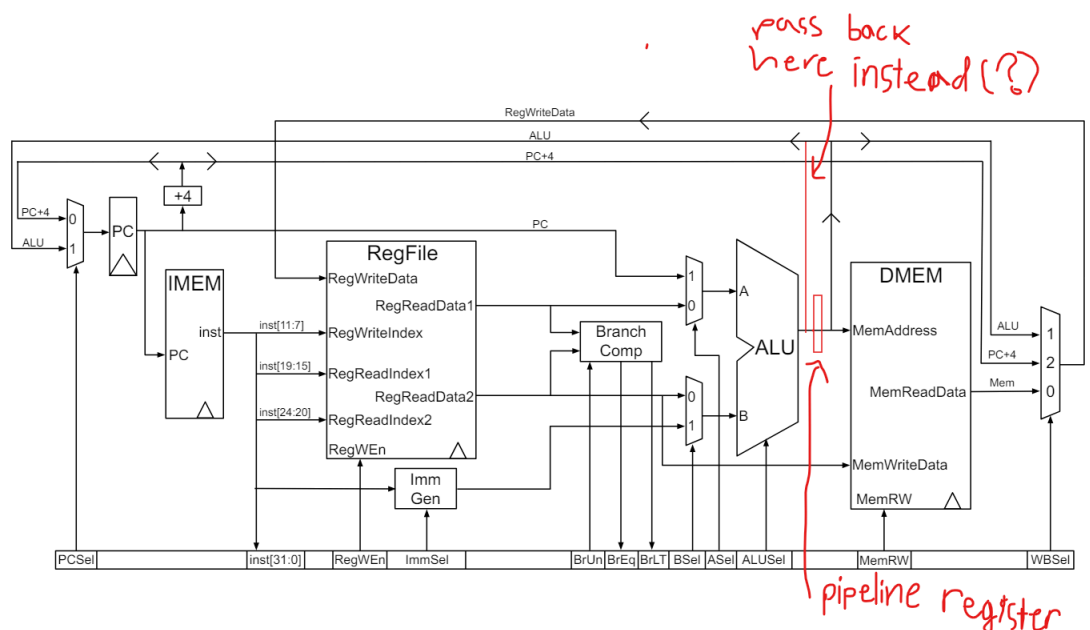
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X	X				
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2								IF	ID	EX	MEM	WB		
9. xori t1 x0 6									IF	ID	EX	MEM	WB	

Why we can't execute the 7th instruction (the one right after jump) right after the EX stage of the 1st instruction (we should know the branch result by then)?

♡ 1 ...

Timothy Bergal 7mth #984adbc

Edit: I think it makes sense to me know - the branch result is passed back to the PC mux at the start of the MEM cycle, hence instruction 7 begins its IF stage when instruction 1's write back stage is executing. However, why can't we just pass back the ALU result before pipeline register, saving us a cycle:



♡ 1 ...

Jero Wang STAFF 7mth #984adee

This is a design decision for the datapath, and it's just how it's taught for 61C. It can definitely save a cycle if you pass it before the output, though it might increase the clock period (which might be unideal, depending on what you're prioritizing).

♡ ...

Anonymous Seahorse 7mth #984afdc

↩ Replying to Jero Wang

is there a reason why line 5 and 6 is not on the chart?

♡ ...



Anonymous Penguin 7mth #984acea

✓ Resolved

SP23-Final-Q3

In Q3.11, if we casted the result to an integer we would still get full credit right? I forgot that the division automatically rounds down to the nearest integer.

In Q3.13, we should still get full credit if we wrote `start_row + chunk_size`, right?

♡ ...



Jero Wang STAFF 7mth #984aeba

Q3.11: Yes, however, please note operator precedence when casting in general (e.g. `(int) a / b` is not the same as `(int) (a / b)`), though doesn't matter here.

Q3.13: Yes, that works.

♡ ...



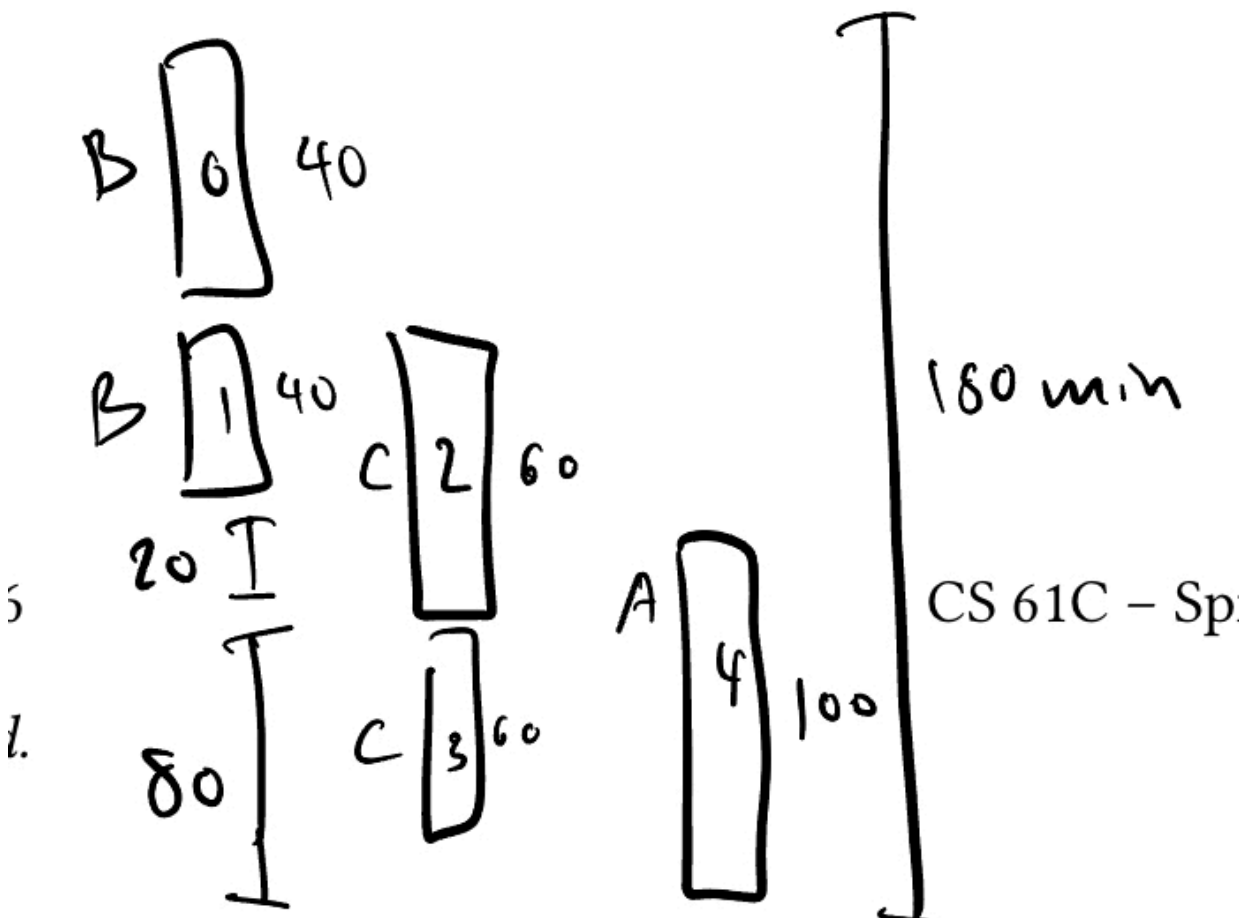
Anonymous Pheasant 7mth #984acdf

✓ Resolved

SP23-Final-Q8:

The solution says the minimum time is 200 min, but I'm not sure what's wrong with my 180 min implementation: A: 4 ; B: 0, 1, ; C: 2, 3

The order of events would visually look kind of like below. Is this valid?



♡ ...



Jero Wang STAFF 7mth #984aebb

CPU B for tasks 0 and 1 take 50 mins each (90 minutes of memory, 3x as fast, so 30 minutes; 10 minutes of math, 2x as long, so 20 minutes).

♡ ...



Anonymous Lobster 7mth #984acde

✓ Resolved

Solution: 7/12. The key insight here is to notice that once we start accessing `arr[4]` and greater, we begin to have to evict members. For ease of explanation, we consider the 5th iteration of the loop below. Listing out the 12 accesses, we have the below cache state at the start of the iteration:

Data	LRU
<code>arr[0]</code>	3
<code>arr[1]</code>	2
<code>arr[2]</code>	1
<code>arr[3]</code>	0

1. Read `arr[4]`. This evicts `arr[0]`, resulting in a miss.
2. Read `arr[0]`. This evicts `arr[1]`, resulting in a miss.
3. Write to `arr[4]`. This results in a hit.
4. Read `arr[4]`. This results in a hit.
5. Read `arr[1]`. This evicts `arr[2]`, resulting in a miss.
6. Write to `arr[4]`. This results in a hit.
7. Read `arr[4]`. This results in a hit.
8. Read `arr[2]`. This evicts `arr[3]`, resulting in a miss.
9. Write to `arr[4]`. This results in a hit.
10. Read `arr[4]`. This results in a hit.
11. Read `arr[3]`. This evicts `arr[0]`, resulting in a miss.
12. Write to `arr[4]`. This results in a hit.

Counting our hits, we have 7 hits and 5 misses, leading to our above hitrate.

how did we get the cache state at the start of the iteration?

♡ ...



Jero Wang STAFF 7mth #984aebc

In the fourth iteration ($i = 3$), access the elements in order (from 0 to 3), so the last accessed element was `arr[3]`, followed by `arr[2]` ... If you're uncertain, I'd recommend drawing out the first 4 iterations for practice!

In the future, please label your questions!

♡ ...



Anonymous Rhinoceros 7mth #984accf

✓ Resolved

SP23-Final-Q9

I don't really understand why we're looking at the mantissa bits and where the 30 and 31 in the solutions came from. Can't we just identify if it's representable using the significand (since we know that if its greater than 2^5 it will be infinity or Nan), thus I did $\sim(B\&C\&D\&E\&F\&G)$ because as long as the exponent values are not all 1s it is representable?

♡ ...



Justin Yokota STAFF 7mth #984bbde

You're going the wrong way; we want to determine if an unsigned number can be exactly representable as a float. Either way, some numbers can be represented as a float but not as an integer; for example, the number 0.5 is representable as a float, but not an integer.

♡ ...



Anonymous Rhinoceros 7mth #984accb

✓ Resolved

Sp23-Final-Q7

I'm pretty confused about the entire logic for this question. For 7.1, how did we arrive at instruction & 64? For the rest of the code why are we taking the sizeof(int) when allocating space? Aren't the instruction greater than just the size of one int?

Would greatly appreciate if someone could walk me through the thought process behind this code!

♡ ...



Candice Yang STAFF 7mth #984acfe

The bit that distinguishes branch and jump instructions from other instructions is the leftmost bit of their opcode.

Note that the program input is `int* code`.

♡ ...



Anonymous Weasel 7mth #984acbf

✓ Resolved

SP23-Final-Q6: are any parts of this question in scope?

♡ 1 ...



Candice Yang STAFF 7mth #984acfc

They are all out of scope. They are in the two optional lectures.

♡ 1 ...



Anonymous Spoonbill 7mth #984acbc

✓ Resolved

Fall 2023, Q 3

Is double pumping something we learned this year and if so, what does it refer to?

Q3.2 (3 points) **If we implement double pumping and all forwarding paths, during which cycle does the xori on line 9 execute its WB stage?**

♡ ...



Rohan Mathur STAFF 7mth #984acbd

Yes, double pumping is in scope. Check page 5 of [discussion 8](#).

[Double pumping is] when data is transferred along data buses at double the rate, by utilising both the rising and falling clock edges in a clock cycle

♡ ...



Anonymous Spoonbill 7mth #984acbe

Got it so it's when you write then-read in one cycle for RegFile?

♡ ...



Rohan Mathur STAFF 7mth #984acdb

← Replying to Anonymous Spoonbill
correct!



Anonymous Spoonbill 7mth #984acaf ✓ Resolved

Fall 2023, Q 3 When I originally solved this problem, I assumed that since branch is not taken (and we don't know if it's actually taken until the MEM stage of instruction1), we proceed with the code as normal. This means since there is data dependency between instruction 2-4 (x0) we would implement stalling for this. I'm wondering why we don't do this even though we proceed as if branch is not taken. For example, in the case that beq is actually not taken, wouldn't you have the stalling between instructions 2-4 to address the data dependency of x0? Or is it always the case that when deciding if ur taking a branch of not (till you reach MEM stage), you just proceed without any stalling?

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to t0 to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X	X				
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2									IF	ID	EX	MEM	WB	
9. xori t1 x0 6										IF	ID	EX	MEM	WB



Jero Wang STAFF 7mth #984aebd

I think in this case, we are assuming that x0 s don't cause data hazards (since they can never be changed). If that behavior is required, we'll specify whether x0 causes data hazards or not.



Anonymous Spoonbill 7mth #984aebe

Ahhh got it, so if we used another register instead of x0, would we still have to proceed as if there were data hazards even if we nop later once we realize we branch?



Anonymous Quetzal 7mth #984afbb

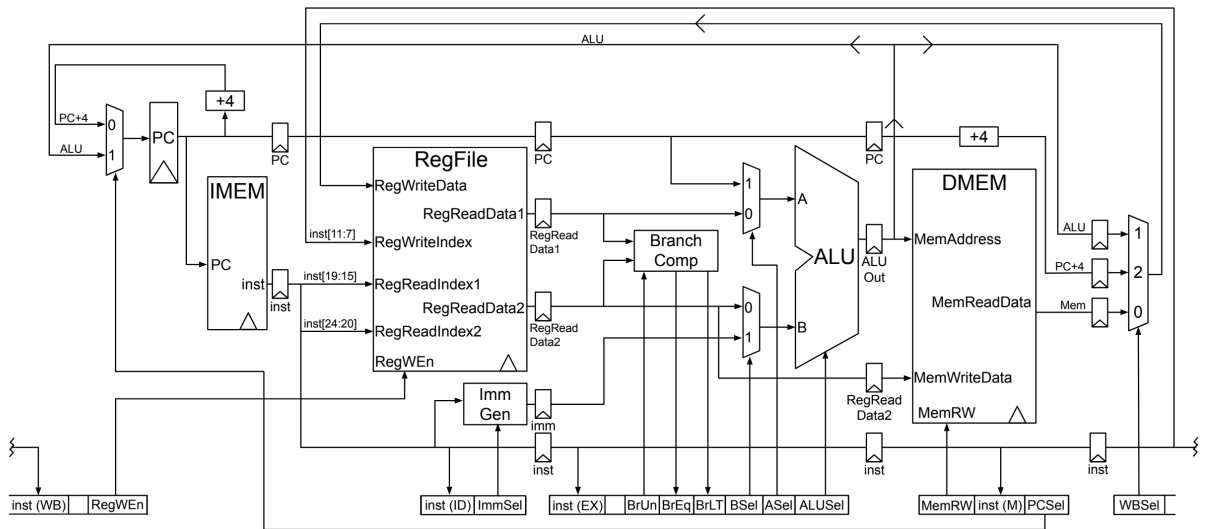
An additional question in this context: since as Spoonbill says, we don't know the branch is not taken until after the MEM phase, which is time unit 4, then why do we mark any "X"s in time unit columns 3 and 4? All these instructions must have normally progressed through the pipeline, and suddenly in time unit 5 we discovered that they were wrong. So the "X"s should only appear at time unit 5 or later (for the instructions that were wrongly speculatively executed)?



Anonymous Penguin 7mth #984acae ✓ Resolved

SP23-Final-Q2.1

Why is the immediate generator included in our clock period calculation? Isn't it part of the instruction decode stage? Like in the following diagram, shouldn't the second stage be everything after the second "column" of pipeline registers (PC, RegReadData1, RegReadData2, imm, inst) while the first stage is everything before?



♡ 2 ...



Candice Yang STAFF 7mth #984acfb 👁 Visible to staff only

I think this student is correct, can someone ack? This q splits F/D and X/M/B

♡ ...



Jedidiah Tsang STAFF 7mth #984acfd

In Spring 2023, the ImmGen was placed in the Ex stage.

♡ 4 ...



Anonymous Quetzal 7mth #984acac ✓ Resolved

Q1.4 (1 point) True or False: The linker initializes the stack with program arguments.

- True False

Q1.5 (1 point) True or False: During a thread context switch, the entries of the TLB get invalidated.

- True False

SU23-Final-Q1.5.

Is this answer correct? I had assumed that different threads operate in the same virtual address space, hence don't need the TLB's entries invalidated upon a thread context-switch.

♡ ...



Candice Yang STAFF 7mth #984aced 👁 Visible to staff only

I think this student is right, can someone ack

♡ ...



Jero Wang STAFF 7mth #984adec

Threads in the same process do operate within the same virtual address space. However, in 61C, we teach that a context switch will invalidate TLB (for simplicity), see lecture 35, slide 58, but it's a bit ambiguous whether this refers to process or thread context switch. This is more explicitly defined in Summer 2023, lecture 22, slide 10.

If we do decide to test on things like this, we'll ensure it's unambiguous for the current semester.

♡ 1 ...



Anonymous Llama 7mth #984acaa ✓ Resolved

SU23-Final-Q8

Q8.5 (1.5 points) 0xC61

Solution: N/A

Since bits 11 and 0 are set, it cannot fit into a floating point number with 8 mantissa bits, so it is not representable.

Q8.6 (1.5 points) 0x200

Solution: 0x0A00

Each chunk is 64 bytes, so that accounts for the lower 6 bits of the address, 0x00. The chunk address is the upper 6 bits of the address, is 0x08. We then must translate 0x08.00 to the floating point system described.

$0x08.00 = 0b1 \cdot 2^3$, so the exponent is 0b1010 after applying the bias, and the mantissa is all 0's. This gives us 0x0A00.

i don't understand 8.5 why it is not representable if those bits are set? and for 8.6, what is the purpose behind the lower and upper 6 bits since the question only provides the sign/exponent/mantissa bits?

♡ ...



Jero Wang STAFF 7mth #984aebf

Q8.5: Since we're representing addresses as 13-bit FP addresses, we need to convert this address to FP. Since our FP representation has 8 bits of mantissa, it means we can represent 8 bits of precision, but we have 13 bits of precision here (assume the most significant 1 is your implicit 1 for FP, how are you going to fit the next 12 bits in your 8-bit mantissa without losing precision?).

Q8.6: We use the upper/lower 6 bits split here because they represent the value before and after the floating point, since the integer part represents the chunk (upper 6 bits) and the fractional part represents the offset within the chunk (lower 6 bits).

♡ ...



Anonymous Penguin 7mth #984abfe

✓ Resolved

SP23-Final-Q1.1

Solution: The readArr instruction is similar to any other R-type instruction, except that the ALU operation performed must be $rs1 + rs2 * 4$. Hence, we need to add a new ALU operation, and there are no other changes needed to the standard datapath.

Grading: Each checkbox was graded as it's own true/false question, and selecting "None of the above" was treated as not selecting any of the other choices.

Why is readArr similar to an R type instruction, how do we know it isn't similar to a lw instruction? If rs1 points to an array, don't we need to access DMEM to retrieve the memory that is being pointed to? I was thinking rs2 acts as our "offset", so we should access memory at the address $rs1 + rs2 * 4$.

♡ ...



Candice Yang STAFF 7mth #984acfa

lw is an I-type instruction which only encodes rs1, rd, and imm. Note that we need to read rs2 instead of imm in readArr. R-type instructions encode rd, rs1, and rs2. Also note for rs2, we need to read it from regfile, and imm needs to be generated from imm generator. And yes, we need to access the DMEM because we are reading from memory.

♡ 1 ...



Anonymous Pigeon 7mth #984abfb ✓ Resolved

Q3.9 (2 points) What is the maximum number of bytes a bge instruction can branch by? Write your answer as a sum or difference of unique powers of 2 (e.g. $2^3 - 2^2 + 2^1$).

Solution: Branches have 12-bit immediates, along with an additional implicit 0. Since this immediate is interpreted as a two's complement number, this means that we can only represent a range from -2^{12} to $2^{12} - 2$ (since bit 0 is forced to be 0).

Thus the maximum number of bytes we can branch is 2^{12} (backwards).

Grading: The question was graded as all-or-nothing, except partial credit was awarded for accounting for only the forward branch.

SU-final-Q3.9

Why would it not be $2^{12} + 2^{11} + 2^{10} + \dots + 2^1$? I thought since the largest immediate we can have is if all of the top 12 bits are 1, then the max immediate we can have is the sum I wrote. Why is it just 2^{12} ?

♡ ...



Candice Yang STAFF 7mth #984acee

risc-v immediates are two's complement numbers. So if all of the top bits are 1 and the last bit is 0, to compute it's decimal value, we flip all bits which gives us $0b1$, and add 1 to it, giving us $0b10$, so decimal value is -2.

♡ 1 ...



Anonymous Llama 7mth #984abfa ✓ Resolved

Regardless of your above answers, assume that we have 20-bit physical memory addresses, and 24-bit virtual memory addresses. Assuming that physical pages are assigned sequentially and the following 3 virtual addresses have been accessed, in order:

Virtual Address	PPN
0xABCDEF	0
0xAABCC	1
0x202122	2

After accessing the previous three addresses, we access the following virtual addresses in order. For each access, fill out the corresponding physical address, and whether the access causes a page hit or a page fault. Assume that if a page fault occurs, then the next sequential physical page number is assigned to the virtual page number.

Q5.4 (1 point) 0xA01243

Page Hit Page Fault

Solution: The VPN is 0xA01 and the offset is 0x243. VPN 0xA01 has not been accessed before, so it is a page fault. Since it's a page fault, we use the next sequential PPN, which is 0x3, so the physical address is 0x03243.

SU23-Final-Q5

does this question also assume that when we get the next available physical page due to a page fault, we don't evict anything from the TLB that's already mapped in the chart, we just add to it?

♡ ...



Candice Yang STAFF 7mth #984acef

Yes, the problem statement says "Assume that if a page fault occurs, then the next sequential physical page number is assigned to the virtual page number."

♡ ...



Anonymous Sand Dollar 7mth #984abee ✓ Resolved

SP23-Final-Q4.1

how we do we know the index is 0b11. I understand we get the offset by making 7 into 4 bit binary and same with 3030 for the tag, but after we find that the index is two bits how do we get the actual index number?

♡ ...

 **Anonymous Penguin** 7mth #984abff

You convert the address into binary and partition the bits accordingly. So the offset is going to have 4 bits, meaning we take the first 4 bits in the address, the index is going to take the next two bits, and the tag takes the rest.

♡ ...

 **Anonymous Sand Dollar** 7mth #984acab

that makes so much sense thanks

♡ 1 ...

 **Anonymous Cattle** 7mth #984abeb

✓ Resolved

Q5.10 (2 points)

```
1 bool more_even_pragma(uint32_t* array, uint32_t n) {
2     uint32_t even_count = 0;
3     uint32_t odd_count = 0;
4     #pragma omp parallel
5     {
6         uint32_t evens;
7         uint32_t odds;
8         for (uint32_t i = 0; i < n; i++) {
9             if (array[i] % 2 == 0) {
10                evens++;
11            } else {
12                odds++;
13            }
14        }
15        #pragma omp critical
16        {
17            even_count = evens;
18            odd_count = odds;
19        }
20    }
21    return even_count > odd_count;
22 }
```

- Correct and faster than the naive `more_even`
- Correct and slower than the naive `more_even`
- Incorrect

Solution: There is no declaration of parallelizing the `for` loop via a `#pragma omp parallel for` directive. However, since each thread completes all of the work, the direct assignment on lines 17 and 18 give us the correct answer.

FA23-Final-Q5.10

I think this OpenMP implementation is incorrect:

in the critical section you assign the private variables evens and odds to the even_count and odd_count instead of adding them. So in the end your even_count and odd_count will be the copy of evens and odds of a single thread instead of the sum over all thread. Then logically the answer is wrong.

♡ ...



Jero Wang STAFF 7mth #984abed

The printed answer is correct.

At like 4, we start executing the following block on each thread. On each thread, we calculate how many evens and odds there are in the entire array (evens and odds should be initialized to 0; this was clarified during the exam). At the end, each thread will have the same (and correct) result in evens and odds , so one of the threads will update even_count and odd_count last, and it doesn't matter which.

♡ ...



Anonymous Cattle 7mth #984abea

✓ Resolved

Q4.4 (2.5 points) What is the hit rate for the **last** iteration of the **for** loop, using an **LRU** replacement policy?

Solution: 7/12. The key insight here is to notice that once we start accessing `arr[4]` and greater, we begin to have to evict members. For ease of explanation, we consider the 5th iteration of the loop below. Listing out the 12 accesses, we have the below cache state at the start of the iteration:

Data	LRU
<code>arr[0]</code>	3
<code>arr[1]</code>	2
<code>arr[2]</code>	1
<code>arr[3]</code>	0

1. Read `arr[4]`. This evicts `arr[0]`, resulting in a miss.
2. Read `arr[0]`. This evicts `arr[1]`, resulting in a miss.
3. Write to `arr[4]`. This results in a hit.
4. Read `arr[4]`. This results in a hit.
5. Read `arr[1]`. This evicts `arr[2]`, resulting in a miss.
6. Write to `arr[4]`. This results in a hit.
7. Read `arr[4]`. This results in a hit.
8. Read `arr[2]`. This evicts `arr[3]`, resulting in a miss.
9. Write to `arr[4]`. This results in a hit.
10. Read `arr[4]`. This results in a hit.
11. Read `arr[3]`. This evicts `arr[0]`, resulting in a miss.
12. Write to `arr[4]`. This results in a hit.

Counting our hits, we have 7 hits and 5 misses, leading to our above hitrate.

FA-23-Final Q4.4

The question asks for the last iteration hit rate, but the explanation says to consider the 5th. I understand that mathematically 5th iter and 6 iter have the same hit rate, but theoretically can

you always assume that once evictions happen at a certain iteration, hit rate will be the same in later iterations as well?

♡ ...



Jero Wang STAFF 7mth #984abec

No, it would depend on the exact for loop we're working with.

♡ ...



Anonymous Dove 7mth #984aaec

✓ Resolved

Su23 Final, 3.9 and 3.10:

After implementing the processor, AJ spends some time writing code for it. However, because the project he is working on has a really large codebase, he encounters the branch and jump range limitations.

Q3.9 (2 points) What is the maximum number of bytes a bge instruction can branch by? Write your answer as a sum or difference of unique powers of 2 (e.g. $2^3 - 2^2 + 2^1$).

Q3.10 (2 points) What is the maximum number of bytes a jal instruction can jump by? Write your answer as a sum or difference of unique powers of 2 (e.g. $2^3 - 2^2 + 2^1$).

Why isn't it the maximum 12 bit number represented by 2s complement for 3.9 and then max 20 bit number represented by 2s complement for 3.10, since its asking the max number of bytes it can branch by?

♡ ...



Jero Wang STAFF 7mth #984abdd

We need to subtract by 1 in order to get an even number, since we have an implicit 0 at the end, which makes representing odd offsets impossible.

♡ ...



Anonymous Walrus 7mth #984aadd

✓ Resolved

SU23-FINAL-Q3.12

Why does jump has control hazard? I think only the branch operation has to make predicitions.

♡ ...



Jero Wang STAFF 7mth #984abdc

The destination address is not known until EX occurs, so we still need to stall until that calculation finishes.

♡ ...



Anonymous Hedgehog 7mth #984aadb

✓ Resolved

Sp23 Final Q5

Which method should we be using to evict blocks in the TLB? The instructions never specify LRU, MRU, FIFO, LIFO, etc.

♡ ...

J Jero Wang STAFF 7mth #984abdb

It does not matter for this question, but it will be specified if it does affect the answer.

♡ ...

J Anonymous Hedgehog 7mth #984acda

Why does it not matter? For example in 5.2, we do a TLB miss and should therefore evict a block to make space for the VPN 000002. This affects the rest of the questions as it can change whether or not there is a hit right?

Edit: Oh it doesn't matter since the rest of the problem's VPN are all misses regardless of how we evict

♡ ...

J Jero Wang STAFF 7mth #984addc

↩ Replying to Anonymous Hedgehog

Yeah, it doesn't matter here due to the specific pattern of accesses.

♡ ...

Anonymous Walrus 7mth #984aabb ✓ Resolved

SU23-Q2 Is hamming code in scope?

♡ ...

J Jedidiah Tsang STAFF 7mth #984aabe

Nope

♡ ...

T Timothy Bergal 7mth #984aaab ✓ Resolved

Q 9

- Originally, the `orion` function works by performing a bitwise OR on the input and Orion's favorite number, and returning 1 (true) if the result is 0, and 0 (false) otherwise.

The below is the compiled (RISC-V RV32I) code of Orion's `orion` function:

```

orion: # Input is received in a0, and the result is outputted in a0
    li a1 ORNUM # Orion's favorite number omitted
    or a0 a0 a1
    bne a0 x0 FalseCase
    addi a0 x0 0
    jr ra
FalseCase:
    addi a0 x0 1
    jr ra

```

Shouldn't `orion` return 0 in the false case and 1 in the true case, as per the problem description (here, 1 is returned in the false case and 0 is returned in the true case)?

♡ ...

J Jedidiah Tsang STAFF 7mth #984aaac

Yeah, I think this was one of the clarifications we issued last semester. Good catch!

♡ ...

J Jedidiah Tsang STAFF 7mth #984aacc

Also please tag your questions appropriately in the future, thanks!

♡ ...

T Timothy Bergal 7mth #984aadc

← Replying to Jedidiah Tsang

Ok, I will update my question! Thanks for the reply!

♡ ...



Anonymous Elk 7mth #984aaaa

✓ Resolved

Sp23 Final Q2.1

since the load instruction write back to regfile and write back wouldn't be pipelined, why wouldn't regfile setup be present instead of the register setup in the answer?

Q2 IF Only ID Pipelined Better

(10 points)

In Project 3, we implemented a RISC-V CPU with two stages; stage 1 included IF and stage 2 included ID/EX/MEM/WB. For this question, imagine instead that we implement a two-stage pipeline with a different split; stage 1 will include IF/ID and stage 2 will include EX/MEM/WB (IF/ID/EX/MEM/WB are defined equivalently to the pipelined CPU on the reference card).

For Q2.1 and Q2.2, assume the following delays for each component. Any component not listed is assumed to have a negligible delay.

Component	Delay
$\tau_{\text{clk-to-q}}$	35ps
τ_{setup}	20ps
Mux	75ps
Regfile Setup	20ps
Regfile Read	175ps
Immediate Generator	150ps
Branch Comparator	200ps
ALU	200ps
Memory Read	300ps

Q2.1 (3 points) What is the minimum clock period of this circuit, in picoseconds, to achieve correct behavior?

Solution: 855ps ($\tau_{\text{clk-to-q}}$ (35) + Immediate Generator (150) + Mux (75) + ALU (200) + Memory Read (300) + Mux (75) + τ_{setup} (20)).

The critical path occurs in stage 2 for a load instruction.

♡ ...



J Jedidiah Tsang STAFF 7mth #984aaad

Hmmm...I thought about this for a bit, and this should probably be regfile setup (doesn't end up mattering cuz they're both 20ps), but the correct last operator should be the regfile setup.

♡ ...



Anonymous Barracuda 7mth #984fef

✓ Resolved

Sp23 Final Q3.12

Is it assumed that in SIMD, thread_num is 1-indexed instead of 0-indexed?

```
Q3.12: thread_num * chunk_size
Q3.13: (thread_num + 1) * chunk_size
```



Anonymous Barracuda 7mth #984ffa

in other words, there's no thread #0, but the first thread is thread #1?



Jedidiah Tsang STAFF 7mth #984aaaf

Threads are 0-indexed in SIMD. Consequently, your loop iterations start from $i=0$ (so you can access the 0th row of the matrix).



Anonymous Barracuda 7mth #984adcb

↩ Replying to Jedidiah Tsang

I see, but say we are in the "first thread", thread_num would return 1 and not 0, correct?



Jedidiah Tsang STAFF 7mth #984adea

↩ Replying to Anonymous Barracuda

We should assume that "first thread" would refer to the 0th thread



Anonymous Barracuda 7mth #984afeb

↩ Replying to Jedidiah Tsang

In that case wouldn't this return 0 for 3.12 (which isn't what we want), for say the first thread?



Jedidiah Tsang STAFF 7mth #984affa

↩ Replying to Anonymous Barracuda

Why don't we want 3.12 to return 0 for thread 0?



Anonymous Barracuda 7mth #984baca

↩ Replying to Jedidiah Tsang

I was assuming we wouldn't want our start_row to be equal to 0, but I'm probably wrong



Jedidiah Tsang STAFF 7mth #984bafc

↩ Replying to Anonymous Barracuda

I would probably try coding this out with a naive, non-parallelized solution. Since we're computing the average of our matrix, we would want to iterate through every element in our matrix, starting from $i = 0$. That doesn't change as we parallelize our solution.

♡ ...

 **Anonymous Barracuda** 7mth #984bbbe

↩ Replying to Jedidiah Tsang

I see, is it because we want it to return 0 since we'd want our row to start at 0 (considering the matrix is 0-indexed)?

♡ ...

 **Jedidiah Tsang** STAFF 7mth #984bbca

↩ Replying to Anonymous Barracuda

yup!

♡ ...


 **Anonymous Stork** 7mth #984fee

✓ Resolved

Fa23 Final Q10

Can someone share some tips or intuition in constructing complex FSMs like the one in this question?

♡ ...

 **Jedidiah Tsang** STAFF 7mth #984aaba

I think labeling the individual states goes a long way here - in this case, since we're tracking the recent two timesteps, we can have four distinct states representing the recent two bits that we've seen (00, 01, 10, and 11). From here, I would personally work off of the example shown in the problem statement, and trace how my output changes as we go from 00 --> 01, or 00 --> 10, or 00 --> 00.

♡ ...

 **Anonymous Bear** 7mth #984fed

✓ Resolved

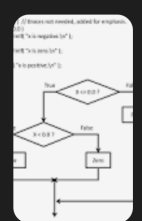
Spr 23 Q7.1

The return type for the function isBranchJump is boolean, so I am confused on how the answer "return (instruction & 64)" would return a boolean. Wouldn't it make more sense to do (instruction & 64) == 64?

♡ ...

 **Anonymous Pigeon** 7mth #984fff

C does not have boolean data types, and normally uses integers for boolean testing. Zero is used to represent false, and One is used to represent true. For interpretation, Zero is interpreted as false and anything non-zero is interpreted as true.



♡ 1 ...

 **Anonymous Spoonbill** 7mth #984fec

✓ Resolved

Fa 2023 Q 5, does omp parallel for split the for loop across the array so the data race doesn't actually matter? or am I incorrect to think this? Each thread will be looking at it's own set of values in the array so there will be no impact in the even/odd_count val?

Q5.9 (2 points)

```
1 bool more_even_pragma(uint32_t* array, uint32_t n) {
2     uint32_t even_count = 0;
3     uint32_t odd_count = 0;
4     #pragma omp parallel for
5     for (uint32_t i = 0; i < n; i++) {
6         if (array[i] % 2 == 0) {
7             even_count++;
8         } else {
9             odd_count++;
10        }
11    }
12    return even_count > odd_count;
13 }
```

- Correct and faster than the naive `more_even`
- Correct and slower than the naive `more_even`
- Incorrect

Solution: There is a race condition through multiple threads accessing `even_count` and `odd_count` without critical sections or reductions.



Jedidiah Tsang STAFF 7mth #984aabc

The issue is that although different threads handle different iterations of the for loop, `even_count` and `odd_count` are shared variables (since they're declared outside of the `#pragma omp parallel for` compiler directive. Consequently, if you view `even_count++` as three `risc_v` instructions (loading in `even_count`, incrementing `even_count`, then storing it back in), you can still have a data race if a thread context switches before it's able to store the incremented value.

♡ 1 ...



Anonymous Spoonbill 7mth #984abef

Ah that makes sense, thanks!

♡ ...



Anonymous Spoonbill 7mth #984feb

✓ Resolved

Fa 2023 Q 5, would there be a way to do this with `vec_xor` instead of `vec_and`?

Fill in the blanks below to finish the implementation of `more_even_simd`. Assume that your code for this subpart is only required to work on inputs where `n` (the length of `array`) is a multiple of 4. You may only use up to one SIMD operation per blank.

```
1 bool more_even_simd(uint32_t* array, uint32_t n) {
2     vector counts = vec_setnum( 0 );
3     vector mask = vec_setnum(1);
4     for (uint32_t i = 0; i < n / 4 * 4; i+=4) {
5         vector temp = vec_load(array + i);
6         vector masked = vec_and(temp, mask);
7         counts = vec_add(counts, masked);
8     }
9     return (n / 2) >
10         (vec_sum(counts));
11 }
```

♡ ...



Jediah Tsang STAFF 7mth #984aabd

I think we constrain the solution space such that xor wouldn't be possible, but feel free to suggest a solution and I can look into it :3

♡ ...





Anonymous Spoonbill 7mth #984fde

✓ Resolved

Fa 2023,

Can someone explain with an example why this circuit works? I don't understand how the inputs, outputs are getting proposed and what the splitter is doing.

 **Anonymous Stork** 7mth #984aeaa
↩ Replying to Anonymous Spoonbill
wondering the same ^
♡ 1 ...

 **Anonymous Sheep** 7mth #984fdb ✓ Resolved
SU23 Q8.5-8.7

I don't understand these problems. Lets take 8.7 for example. Changing 0x7D8 into binary:
0b0111 1110 1000

How are they getting the lower 6 bits (101000) to my understanding to equal 0x18?


Q8.7 (1.5 points) 0x7D8


Solution:

Each chunk is 64 bytes, so that accounts for the lower 6 bits of the address, 0x18. The chunk address is the upper 6 bits of the address, is 0x1F. We then must translate 0b1 1111.01 1000 to the floating point system described.

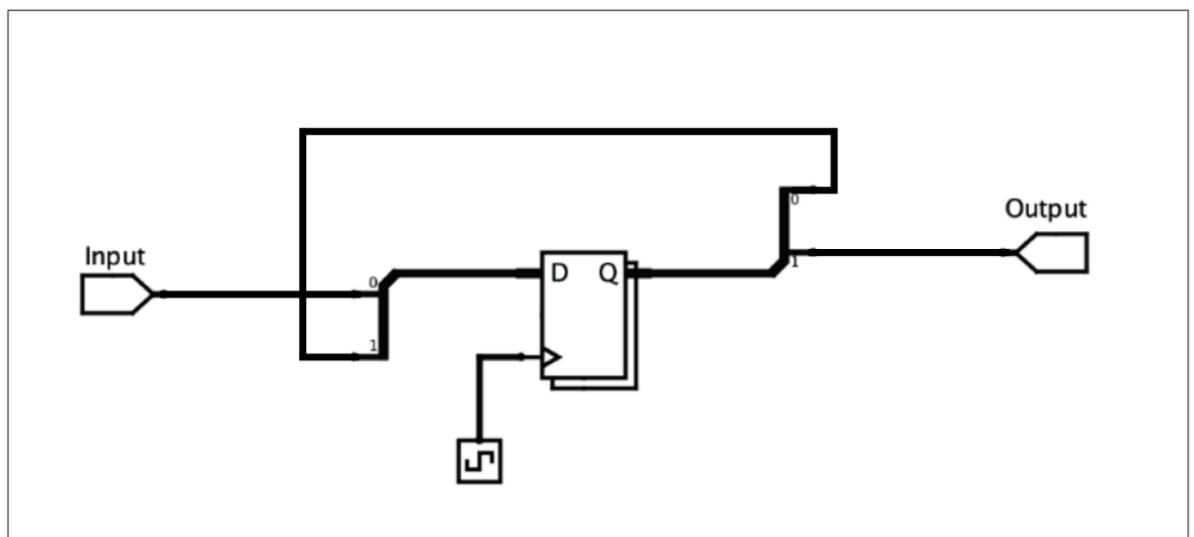
$0b11111.011000 = 0b1.1111011 \cdot 2^4$, so the exponent is 0b1011 after applying the bias, and the mantissa is 0xF6. This gives us 0x0BF6.

♡ ...


 **Jero Wang** STAFF 7mth #984abda
0x7D8 is 0b 0111 1101 1000 , so the lower 6 bits are 0b01 1000 , which is 0x18 in hex.
♡ ...


 **Anonymous Seahorse** 7mth #984fda ✓ Resolved
Fall 23 - Final - Q10.2


I have the correct FSM, but I do not know how to translate it to circuit diagram. How to we get this circuit diagram from the FSM?



♡ 1 ...

 **Anonymous Spoonbill** 7mth #984fdf
Wondering the same ^
♡ ...

 **Jedidiah Tsang** STAFF 7mth #984aaca
[#984aabf](#)
♡ ...

 **Anonymous Spoonbill** 7mth #984fcf ✓ Resolved
Fall 2023

I'm a bit confused on what ret is doing in this case. When you do jalr x0 t3 12, doesn't rd = PC + 4, which in our case is x0. So when we ret does it return back to line 7 (line right after the jalr line) or is it just returning the value in a0?

Q8 Cumulative: Unconditional RISC**(9 points)**

Assume we have a function, f , that takes in a 32-bit unsigned integer, x , as an argument. $f(x)$ is defined as follows:

$$f(x) = \begin{cases} x + 9 & \text{if } x \% 4 == 0 \\ x * 2 & \text{if } x \% 4 == 1 \\ x & \text{if } x \% 4 == 2 \\ x // 8 & \text{if } x \% 4 == 3 \end{cases}$$

Jero wants to write this function in RISC-V, but he couldn't get his CS61CPU's branch instructions to work! As a result, you may use any RV32I instruction **except** branch instructions.

Write a function, f , which accepts one argument x in $a0$, and returns $f(x)$. You may assume that there is no overflow.

```

1 f:
2  andi t3 a0 3
   Q8.1
3  slli t3 t3 3
   Q8.2
4  auipc t7 0
   Q8.3
5  add t3 t3 t7
6  jalr x0 t3 12
   Q8.4
7  addi a0 a0 9
   Q8.5
8  j f_end
9  slli a0 a0 1
   Q8.6
10 j f_end
11 nop
   Q8.7
12 j f_end
13 srli a0 a0 3
   Q8.8
14 f_end:
15 ret

```

♡ ...

J **Jedidiah Tsang** STAFF 7mth #984aacb

From the ref card, `ret` is a pseudoinstruction for $PC = ra$. `jalr` sets rd to $PC + 4$, so since rd is $x0$, it's an unconditional jump. However, this doesn't change the value of ra so PC can return to whatever function called f safely.

♡ ...

J **Anonymous Spoonbill** 7mth #984abfc

Got it, but if we set rd in `jalr` to ra or $x1$ then it would return back to the line after `jalr` when `ret` is called? The way I'm understanding it is that $x0$ can't be modified so the return address is actually not stored anywhere when we call `jalr` so it returns back to function that called f .

♡ ...

J **Jero Wang** STAFF 7mth #984adda

← Replying to Anonymous Spoonbill

If we have `jalr ra, ra, 0`, we jump to the current address stored in `ra`, and then store the `PC+4` (using `PC` before jump) to `ra`. If you have another `ret`, it would jump to the line after the `jalr`.

♡ ...



Anonymous Llama 7mth #984fcb

✓ Resolved

SU23-Final-Q2

is this also a valid solution? i'm using `t1` instead of keeping track of the parity in `LSB`, i'm using it as a counter for the number of 1 bits and then at the end i do `srl` to divide by 2 to see if it equals 0 or 1 depending on if `t1` is an even or odd number?

```
li t1 0
```

```
loop:
```

```
    srl a0, a0, 1
```

```
    addi t1, t1, 1
```

```
    bne a0, x0, loop
```

```
    srl a0, t1, 1
```

```
jr ra
```

```
1 calculate_parity:
2   li t1 0           # t1 holds current parity value
                       # in least significant bit
3 loop:
4   xor t1, t1, a0    # add also acceptable
                       # adds the current least significant bit
                       # in a0 to the parity value in t1

5   srl a0, a0, 1     # shift to the next bit in a0
                       # it's important here that this is
                       # a logical shift, since using a
                       # arithmetic shift would possibly
                       # put us in an infinite loop
                       # if bit 31 is set

6   bne a0, x0, loop  # loop if there are more than 1 bit left in a0
                       # 0 bits will never affect parity

7   andi a0, t1, 1    # we only want the one parity bit
                       # in bit 0 of t1
8   jr ra
```

♡ ...



Jero Wang STAFF 7mth #984abcf

This adds up the number of bits right of the `MSB`, which is not necessarily equivalent to the parity, sorry.

♡ ...



Anonymous Sheep 7mth #984fca

✓ Resolved

How do we get values from 0 through 63 by having 4 exponent bits? Having a bit of trouble connecting the dots.

Q8.1 (2 points) If we had 4KiB of memory, with chunk addresses 0, 1, 2, etc., what is the minimum number of exponent bits in our floating point memory address required to access every byte, assuming that we use a standard bias?

Solution: Given 4KiB memory and 64B chunks, we have 64 chunks total. Therefore, we need 4 exponent bits to be able to represent values 0 through 63. Given 4 exponent bits, the largest non-infinity/NaN exponent value (pre-bias) is $2^4 - 2$, which, after applying the standard bias of -7, is 7, which allows us to represent values up to 63.

If we used 3 exponent bits, the largest exponent value after applying the standard bias of -3 is 3, which cannot represent the value 63.

♡ ...

J Jero Wang STAFF 7mth #984abce

4 exponent bits mean we have a standard bias of -7, which means the largest exponent we can have (without going into NaNs/Infinities) is 7, so the maximum value we can represent with just the exponent is $2^7 = 128$, which is greater than 63.

♡ ...

Anonymous Sheep 7mth #984accd

How do they get from 4 to -7?

♡ ...

J Jero Wang STAFF 7mth #984acdc

← Replying to Anonymous Sheep

It's the standard bias equation from the refcard

♡ ...

Anonymous Sheep 7mth #984fbe

✓ Resolved

SU23 6.2

Are we expected to know this?

♡ 1 ...

J Jedidiah Tsang STAFF 7mth #984aacd

Yes, reductions are in scope (they were in lab 7)

♡ 1 ...

Anonymous Barracuda 7mth #984fbd

✓ Resolved

SP23-Final-Q8.3

Q8.3 (3 points) Using one chip was still taking too long, so you buy both Chip B and Chip C from the store, and connect them to Chip A in a new multicore machine with negligible overhead. Using all three chips, what is the minimum amount of time required to complete this set of tasks? Each task must be completed entirely on one chip.

Please also provide the list of tasks each chip will complete, in order of completion, or write "None" if a chip does not complete any task. For example, if you decide to have Chip A complete all of the tasks, your answer should be "0, 1, 2, 3, 4" for Chip A, and "None" for Chip B and Chip C.

Solution: 200 minutes. There are many task lists. For example: Chip A completes tasks 1 and 3, Chip B completes task 0, Chip C completes tasks 2 and 4.

Grading: Partial credit was awarded with a task list that would run in 200 minutes but the wrong time was provided, or a task list that runs in <250 minutes and has the correct time.


Why is it that it's 200 minutes?

♡ ...


Anonymous Snake 7mth #984fcc

For this problem, try to find combinations of tasks where you maximize parallelism (multiple chips doing work at the same time) and chip specialization (the chip that is good at memory operations doing the tasks with that are more demanding on memory).


♡ ...

 **Anonymous Barracuda** 7mth #984fce
Thanks I didn't realize that we can use parallelism here - does multicore machine imply parallelism is used?


♡ ...

 **Jedidiah Tsang** STAFF 7mth #984aace
↩ Replying to Anonymous Barracuda
Yeah, with multiple cores, it would be a waste to only have one process running at a time since you can now handle PLP.


♡ ...

 **Anonymous Barracuda** 7mth #984adca
↩ Replying to Jedidiah Tsang
Thanks, also sorry what is PLP?

♡ ...

 **Jedidiah Tsang** STAFF 7mth #984addf
↩ Replying to Anonymous Barracuda
Process level parallelism

♡ ...


 **Anonymous Barracuda** 7mth #984fbb ✓ Resolved
SP-23-Final-Q7

Isn't int* (a pointer, which is 4 bytes) and int (4 bytes) the same amount of bytes, so does it really matter if it was int or int* in sizeof()?


Q7.4: sizeof(int*) * (num + 1)

Q7.5: sizeof(int)


♡ ...

 **Jedidiah Tsang** STAFF 7mth #984aacf
You can't necessarily assume that int s are always 4 bytes, and similarly, you can't always guarantee that pointers store 32 bit addresses (you could have 64 bit addresses).


♡ ...

 **Anonymous Barracuda** 7mth #984adcc
I see, so in that case we can't assume how many bytes a pointer or int can be?

♡ ...

 **Jedidiah Tsang** STAFF 7mth #984adeb
↩ Replying to Anonymous Barracuda
correct

♡ 1 ...

 **Anonymous Mantis** 7mth #984fba ✓ Resolved

SP23-Final-Q2.1

When calculating the critical path, why are they using a load instruction? Why isn't the critical path being calculated using all of the components that fall in EX/MEM/WB (like branch comparator)?

Q2.1 (3 points) What is the minimum clock period of this circuit, in picoseconds, to achieve correct behavior?

Solution: 855ps ($\tau_{\text{clk-to-q}}$ (35) + Immediate Generator (150) + Mux (75) + ALU (200) + Memory Read (300) + Mux (75) + τ_{setup} (20)).

The critical path occurs in stage 2 for a load instruction.

Grading: Partial credit was given for errors that showed conceptual understanding of what the critical path is, but excluded or included an extra component's timing. We did not give partial credit for excluding some components since we cannot clearly distinguish between a conceptual misunderstanding or a mechanical error.


♡ ...

 **Jedidiah Tsang** STAFF 7mth #984aada

The branch comparator runs at the same time that the outputs from the regfile are passed into the immgen + amux/bmux + ALU +...

As a result, considering the largest combinational logic delay, the path that lw takes is going to be much longer than any branch instruction. See the review session for more insight (I'll upload them in the morning).

♡ ...

 **Anonymous Sheep** 7mth #984fad ✓ Resolved

SU23 Q4.2 and 4.3

Can someone explain whats going on?

Regardless of your answer to the above question, assume that CoryBot has a direct-mapped trache with TIO breakdown of 7:1:2, while SodaBot has a direct-mapped (binary) cache with a TIO breakdown of 7:1:2.

Q4.2 (2 points) For each block in their trache, CoryBot stores the tag and 1 additional trit of metadata (invalid/valid/dirty). What is the total number of trits used by the trache? Express your answer in terms of powers of 2 and 3.

Solution: $3 \cdot (2^3 + 3^3)$

Each entry must contain 7 trits for the tag, 1 trit for metadata, and 3^2 trytes ($3 \cdot 3^2 = 3^3$ trits) of data. There are 3^1 entries total, for a total of $3 \cdot (2^3 + 3^3)$ trits.

Q4.3 (2 points) For each block in their cache, SodaBot stores the tag and 2 additional bits of metadata (valid bit, dirty bit). What is the total number of bits used by the cache? Express your answer in terms of powers of 2 and 3.

Solution: $2 \cdot (3^2 + 2^5)$

Each entry must contain 7 bits for the tag, 2 bits for the metadata, and 2^2 bytes ($2^2 \cdot 2^3 = 2^5$ bits) for the data. There are 2^1 entries total, for a total of $2 \cdot (3^2 + 2^5)$ bits.

♡ 1 ...



Jero Wang STAFF 7mth #984abcd
#984aafc

Feel free to follow up if you have questions about the explanation or the solutions!



Anonymous Bear 7mth #984fac

✓ Resolved

Q2.1 (3 points) What is the minimum clock period of this circuit, in picoseconds, to achieve correct behavior?

Solution: $855\text{ps} (\tau_{\text{clk-to-q}} (35) + \text{Immediate Generator} (150) + \text{Mux} (75) + \text{ALU} (200) + \text{Memory Read} (300) + \text{Mux} (75) + \tau_{\text{setup}} (20))$.

The critical path occurs in stage 2 for a load instruction.

Grading: Partial credit was given for errors that showed conceptual understanding of what the critical path is, but excluded or included an extra component's timing. We did not give partial credit for excluding some components since we cannot clearly distinguish between a conceptual misunderstanding or a mechanical error.

SP23-Final-Q2.1

Why is Immediate Generator included in the equation when in the pipelined circuit Imm Gen is placed before the pipeline register separating ID and EX?



Jero Wang STAFF 7mth #984abcc

Imm gen was moved to ID in Fall 2023 to align with lecture content, so it was in EX back in Spring 2023.

In the future, please you label your question with the correct label!



B

Ben Yu 7mth #984faa

✓ Resolved

SP23-Final-Q2.3

Q2.3 (3.5 points) In the CPU, which of the following values must have a pipeline register? Select all that apply.

Instruction

RegReadData2

MemReadData

Program Counter

Immediate

None of the above

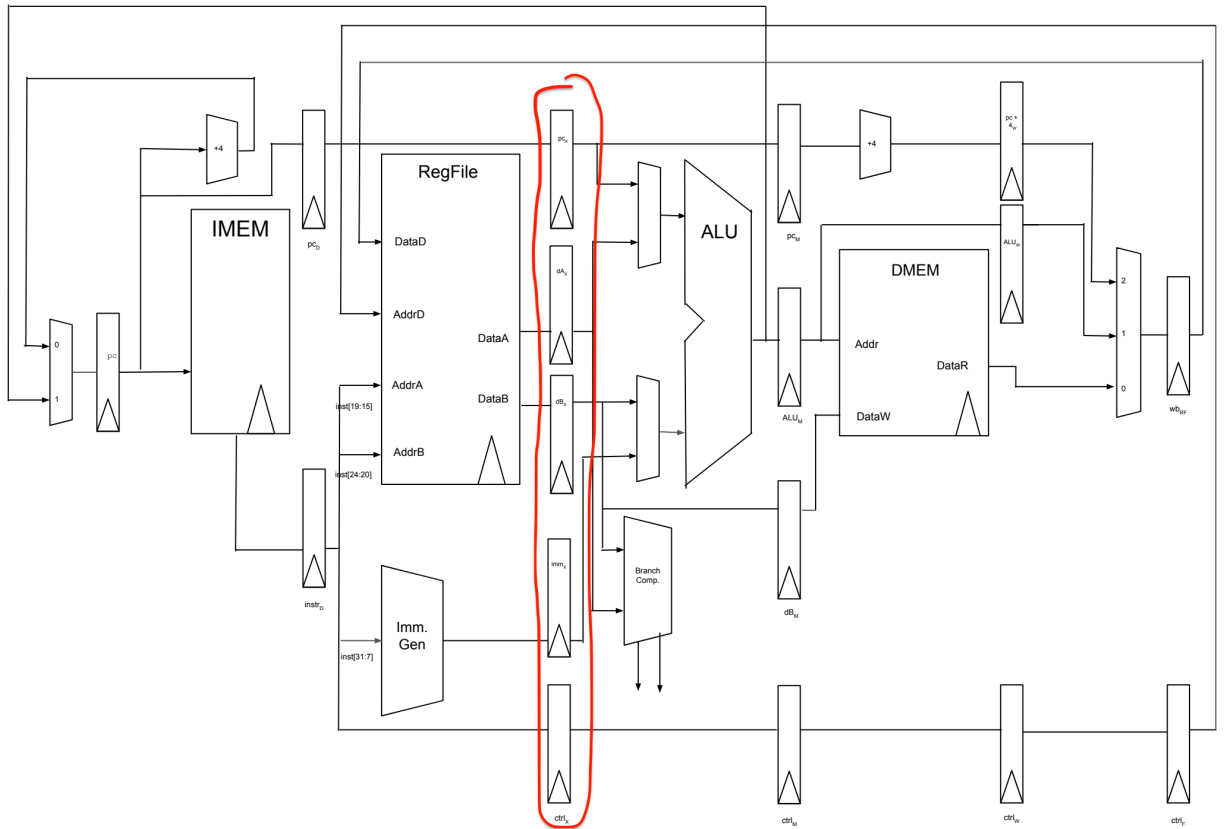
RegReadData1

ALUOut

Solution: The values that require a pipeline register are all values generated in stage 1, which are Instruction (from IMEM), Program Counter (from the PC register), RegReadData1 (from Regfile) and RegReadData2 (also from Regfile). The remaining three values are all outputs of components in the second stage of our pipeline.

Grading: Each checkbox was graded as it's own true/false question, and selecting "None of the above" was treated as not selecting any of the other choices.

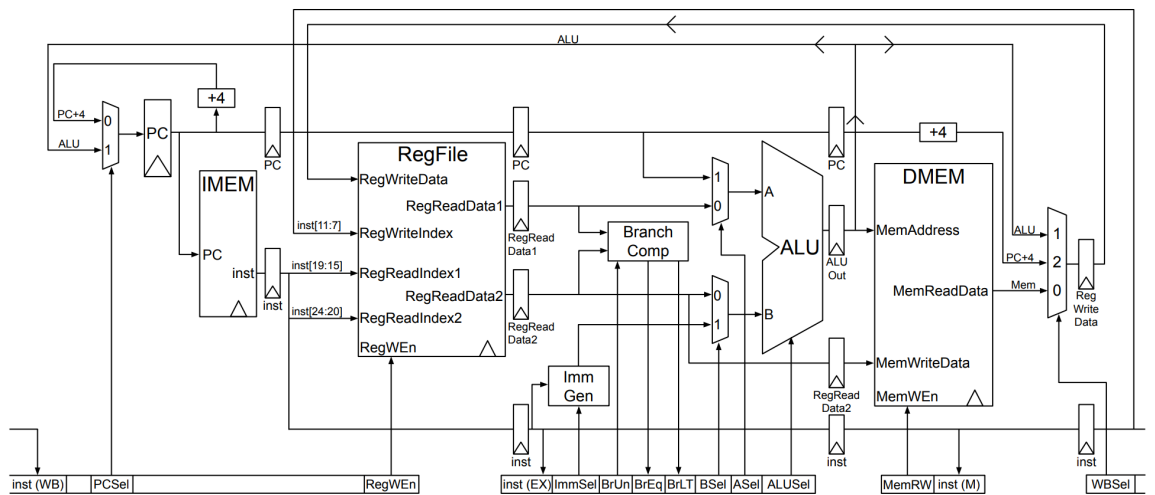
I assume the separation between the two stages have registers as in the image belowL so why Instruction is stored in pipeline register and immediate is not?



♡ ...

Anonymous Snake 7mth #984fab

hi, I think it's because for the sp23 final they had a different layout for the pipelined CPU:



from: <https://inst.eecs.berkeley.edu/~cs61c/sp23/pdfs/resources/reference-card.pdf>

in the above diagram the immediate is not needed to be pipelined because the Imm Gen is in stage 2 not 1.

♡ 1 ...

B Ben Yu 7mth #984fdd

Oh Thank you! That's solved one of my confusions. Does the bottommost register circled in red stores the instruction code?

♡ ...

J Jero Wang STAFF 7mth #984abcb

← Replying to Ben Yu

Yes

♡ ...

B **Ben Yu** 7mth #984aff

← Replying to Jero Wang

Got it

♡ ...

 **Anonymous Snake** 7mth #984eff ✓ Resolved

sp23-final-q2.4


I am a bit confused why a structural hazard wouldn't occur. Does reading from IMEM and reading/writing to DMEM cause a structural hazard, or are IMEM and DMEM considered separate resources?

♡ ...

J **Jero Wang** STAFF 7mth #984abca

No, in 61C, we treat IMEM and DMEM as two separate components.

♡ ...

 **Anonymous Sheep** 7mth #984efe ✓ Resolved

SU23 3.9

Why does it go up to $2^{12}-2$? Doesn't 0 in bit 0 only cause it to decrease by 1?

Solution: Branches have 12-bit immediates, along with an additional implicit 0. Since this immediate is interpreted as a two's complement number, this means that we can only represent a range from -2^{12} to $2^{12} - 2$ (since bit 0 is forced to be 0).

Thus the maximum number of bytes we can branch is 2^{12} (backwards).


Grading: The question was graded as all-or-nothing, except partial credit was awarded for accounting for only the forward branch.

♡ ...

J **Jero Wang** STAFF 7mth #984abbf

The maximum value of a 12-bit signed number is $2^{12} - 1$, and since LSB is 0, we must find the largest even number smaller than the max, which is $2^{12} - 2$.

♡ ...

 **Anonymous Sheep** 7mth #984efd ✓ Resolved

SU23 2.4

Could we have used addi or add instead of andi?

♡ ...

J **Jero Wang** STAFF 7mth #984abbe

No, you have to use andi here to get rid of the upper 31 bits.


♡ ...

 **Anonymous Armadillo** 7mth #984efb ✓ Resolved


SU23-Final-Q3.7

Why is MemRW set to read for the jal instruction? Is it always set to read for anything that is not a store instruction?


♡ ...

 **Anonymous Armadillo** 7mth #984efc
nvm, MemRW should never be *

♡ 1 ...

 **Anonymous Albatross** 7mth #984aefc
Why is that the case?

♡ ...

 **Anonymous Penguin** 7mth #984bbcb
↩ Replying to Anonymous Albatross

Because we don't want jal to write to a location in memory, it's only meant to write back to rd and update PC.


♡ ...

K **Keanu Chandra** 7mth #984efa ✓ Resolved


SU23-FINAL-Q3.12

Why does long jump have a data hazard because a regular jump function doesn't write back to rd...


♡ ...

 **Anonymous Llama** 7mth #984ffb
a regular jump instruction does write back to rd. you can see on the reference card: $rd = PC + 4$

♡ ...

 **Anonymous Swallow** 7mth #984acce
But that doesn't make sense because there is no rd... you just have j label. Unless it's including jal ra and the others.


♡ ...

 **Anonymous Sheep** 7mth #984ede ✓ Resolved


SP23 7.1

Why is the answer instruction & 64?


♡ ...

 **Jero Wang** STAFF 7mth #984abbd
Bit 6 (0-index) is always 1 for jumps and branches, and 0 for all other instructions (excluding ecalls and ebreaks), so by ANDing it with 64 (2^6), we can get that bit 6. If bit 6 is 1, the result is a non-zero number and therefore truthy, and if bit 6 is 0, the result is 0 and therefore falsy.

♡ ...


 **Anonymous Sheep** 7mth #984accc
Are there other generalizations that you can help me out with? Like what about R types, I types... etc?

♡ ...

 **Jero Wang** STAFF 7mth #984acdd
↩ Replying to Anonymous Sheep


Sorry, we don't. This isn't really worth memorizing in my opinion, I'd recommend looking for similarities on the refcard during the exam.


♡ ...

 **Anonymous Sheep** 7mth #984edb ✓ Resolved
SP23 6.4 and 6.5

Are these in scope?


♡ ...


 **Jero Wang** STAFF 7mth #984abbc
[#984ad](#)
♡ ...

 **Anonymous Sheep** 7mth #984eda ✓ Resolved
SP23 Q6.2

What is data shuffling?


♡ ...


 **Jero Wang** STAFF 7mth #984aafb
[#984aadf](#) [#984ad](#)
♡ ...


 **Anonymous Sheep** 7mth #984ecd ✓ Resolved
SP23 Q4.2

How do we know the index of these blocks conflict?

♡ ...

 **Jero Wang** STAFF 7mth #984abbb
Since our cache is a direct-mapped 64B cache with 16B blocks and 16-bit addresses, we have 4 bits of offset, 2 bits of index, and 10 bits of tag. Since $a = 0x1000$ and $b = 0x2000$, their index bits (bits 4-5) are the same for each iteration of the for loop.
♡ ...

 **Anonymous Beaver** 7mth #984adab
Why is it fine for results though?
♡ ...

 **Jero Wang** STAFF 7mth #984aded
↩ Replying to Anonymous Beaver
Results is at $0x3030$, which has a different index (bits 4-5 are $0x11$ instead of $0x00$).
♡ ...

B **Ben Yu** 7mth #984ecb ✓ Resolved
SU23-Final-Q4.2 & 4.3

How is the number of trits / bits determine for the data and entries part, given that it is a 9 block table with 27 trytes in each block (at least for Cory Bot)?

CoryBot has a tryte-addressable memory space with 10-trit memory addresses, and their CPU has a direct-mapped trache with 9 blocks that hold 27 trytes each.

Q4.1 (3 points) Calculate the TIO trits in this setup.

Solution: 5:2:3

Offset: 27 trytes requires $\log_3 27 = 3$ trits.

Index: 9 blocks requires $\log_3 9 = 2$ trits.

Tag: The addresses are 10 trits, which leaves us with 5 tag trits.

Regardless of your answer to the above question, assume that CoryBot has a direct-mapped trache with TIO breakdown of 7:1:2, while SodaBot has a direct-mapped (binary) cache with a TIO breakdown of 7:1:2.

Q4.2 (2 points) For each block in their trache, CoryBot stores the tag and 1 additional trit of metadata (invalid/valid/dirty). What is the total number of trits used by the trache? Express your answer in terms of powers of 2 and 3.

Solution: $3 \cdot (2^3 + 3^3)$

Each entry must contain 7 trits for the tag, 1 trit for metadata, and 3^2 trytes ($3 \cdot 3^2 = 3^3$ trits) of data. There are 3^1 entries total, for a total of $3 \cdot (2^3 + 3^3)$ trits.

Q4.3 (2 points) For each block in their cache, SodaBot stores the tag and 2 additional bits of metadata (valid bit, dirty bit). What is the total number of bits used by the cache? Express your answer in terms of powers of 2 and 3.

Solution: $2 \cdot (3^2 + 2^5)$

Each entry must contain 7 bits for the tag, 2 bits for the metadata, and 2^2 bytes ($2^2 \cdot 2^3 = 2^5$ bits) for the data. There are 2^1 entries total, for a total of $2 \cdot (3^2 + 2^5)$ bits.

♡ ...

 **Anonymous Llama** 7mth #984ffc

i have the same question

♡ ...

 **Jero Wang** STAFF 7mth #984abba

Before Q4.2, we change the cache have different properties (T:I:O is now 7:1:2).

[#984aafc](#)

♡ ...

 **B Ben Yu** 7mth #984baaa

For 4.2, does the answer use 2^3 because that's the closest power to the 7 trits?

♡ ...

 **B Ben Yu** 7mth #984baac

↩ Replying to Ben Yu

Oh no 2^3 is for tag + metadata, and the 3 entries comes from the index. But why the index itself (which counts as a trit) doesn't count in this case?

♡ ...

 **Anonymous Lobster** 7mth #984eca

✓ Resolved

Q10 Cumulative: Art Class

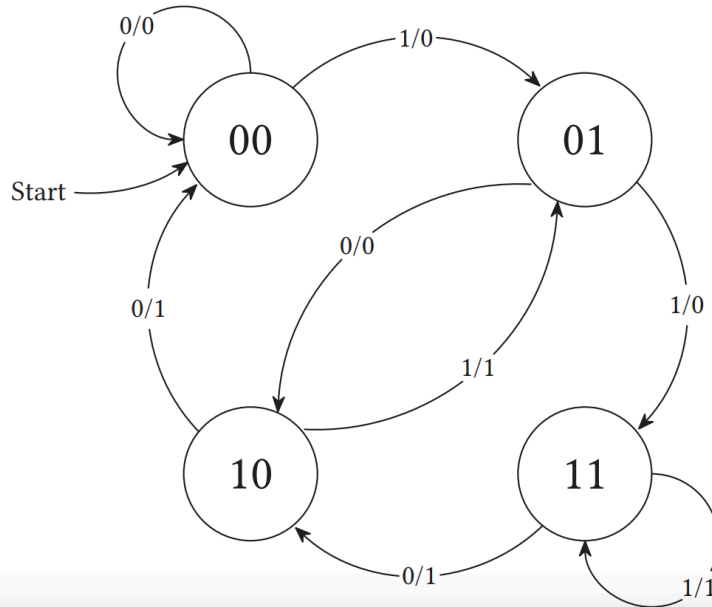
(16 points)

You wish to create an FSM whose output at time step N is 0 for the first two time steps, and the input of time step $N - 2$ otherwise.

For example, if the input to this FSM was 0b01 1011 1011 1000 1001,

the output should be 0b00 0110 1110 1110 0010.

Q10.1 (8 points) Complete the FSM below. You may not add additional states. Note that you must also label the state transitions we have provided for you.



can someone explain how this fsm was drawn? i don't understand it.

♡ ...



Anonymous Oyster 7mth #984aaae **ENDORSED**

You can start off by thinking of the rightmost bit of each state as the most recent input and the left bit as the previous input, and then drawing arrows based what output you'd get, given your starting stage and the next bit you'll see.

♡ ...



Anonymous Armadillo 7mth #984ebf **Resolved**

FA23-Final-Q10.4

Why is the answer in ns, when the timings are given in ps?

♡ ...



Justin Yokota STAFF 7mth #984eeb **Visible to staff only**

... whoops

♡ ...



Jero Wang STAFF 7mth #984abaf **Visible to staff only**

we're too used to ns lol we never use ps

♡ ...



Justin Yokota STAFF 7mth #984eec

Looks like a typo on our end, sorry. You are correct that our answers are off by a factor of 1000.

♡ 1 ...



Anonymous Oyster 7mth #984ebe

✓ Resolved

Sp23-Final-Q7

For 7.4, why do we care about $num+1$ and not just num ? Is it because we're trying to account for counting the instructions before the start and after the end, like in the example?

For 7.7, why is it just $num+1$ and not $i < (num+1)$ since it's in a loop condition?

♡ ...



Sophie Xie STAFF 7mth #984edc

7.4: yes, you got it!

7.7: yep, you're right! (sorry about that)

♡ ...



Annika Liu 7mth #984ebd

✓ Resolved

Su23-Final-Q6

I have a question about the logic of the code, if the width is a number e.g. 11 where it cannot be wholly divided by 4, won't there be bugs? For instance, given an int * 1 2 3 4 8 7 6 4 3 2 1, the code will first compare the vectors 1 2 3 4 and 4 3 2 1 to find they are palindrome. But then the left pointer will move to 8 and the right pointer will move to 4, terminating the while loop and end up not comparing the 8 7 6 in the middle. How do we consider this in the code?

♡ 1 ...



Anonymous Kangaroo 7mth #984fdc

i have this exact same question and when i did the problem i thought the if case below was supposed to handle the middle values but it's not.

♡ ...



Jero Wang STAFF 7mth #984abae

Yeah, I think this question was broken and this was not caught during the exam itself.

For the sake of the question though, assume that N is a multiple of 4 please.

♡ ...



Anonymous Alligator 7mth #984bcbc

If change $left+=4$ to $left+=1$ and $right-=4$ to $right-=1$, is this change correct? I think we just do more repeating comparisons but the outcome should be correct.

♡ ...



Annika Liu 7mth #984ebc

✓ Resolved

Su23-Final-Q3.9

Why is the lower bound -2^{12} but not $-(2^{12} - 1)$ to account for the last bit forced to be 0? Thank you.

♡ ...



Anonymous Llama 7mth #984ece

+also curious about this

♡ ...



Anonymous Swallow 7mth #984eef

✳ ENDORSED

because that is an odd number and we cannot have odd numbers if the lowest order bit is forced to be a 0 because it will always be a multiple of 2.

♡ ...



Anonymous Hyena 7mth #984eba

✓ Resolved

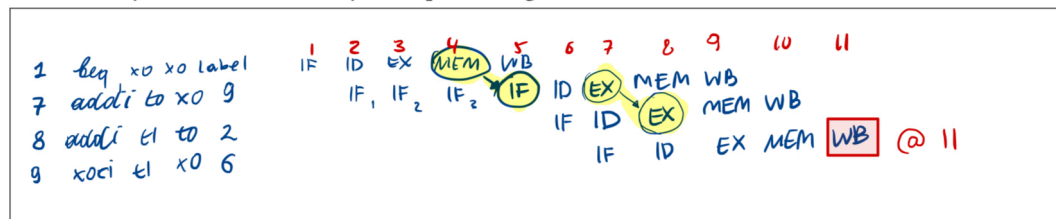
FA23-Final-Q3.2

Would this diagram be considered correct for this question? I'm confused about the difference between converting instructions that should not have been taken after a branch to "no-ops" (I thought this was pipeline flushing?) and stalling (which I thought was represented as in my diagram, with IF₁, IF₂, IF₃ until MEM can give ALU result to the PC register. Thank you!

Q3.2 (3 points) If we implement double pumping and all forwarding paths, during which cycle does the `xori` on line 9 execute its WB stage?

Cycle **11**

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.



♡ ...



Jero Wang STAFF 7mth #984abad

I don't think we have one set notation for pipeline diagrams, but we should still include lines 2-4 in the diagram since we don't actually know if the branch is taken or not, and we're speculatively executing those instructions. Otherwise, it matches the solutions.

If you're referring to the actual answer though, we were more looking for something like "Control hazard on line 1, data hazard between lines 7 and 8...".

♡ ...



Anonymous Cat 7mth #984eaf

✓ Resolved

FA23-Final-Q3.1

(3 points) If all hazards are resolved through stalling (no double pumping or forwarding paths), during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X	X				
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
9. xori t1 x0 6									IF	ID	EX	MEM	WB	

Instruction depends on write from previous instruction -> ID would read wrong value and then calculate wrong result

	1	2	3	4	5	6
add s0 t0 t1	IF	ID	EX	M	WB	
sub t2 s0 to -> no op		IF	no-op	no-op	no-op	no-op
sub t2 s0 to -> no op			no-op	no-op	no-op	no-op
sub t2 s0 to				IF	ID	EX
or t6 s0 t3					IF	ID

I have a couple questions about this problem.

1. If we are stalling how come the IF instructions are run in the next cycle as if we didn't stall at all? I thought stalling moved the IF stage to the number of cycles you stalled it for like in the discussion recording and for the inst. `addi t1 t0 2` where the `nop` is just pushing back the IF stage until it reaches the same cycle as the WB stage of the `addi t0 x0 9` inst.

On the topic of the stalling, I'm also a bit confused on why we pushed the `addi t1 t0 2` IF stage all the way to the WB stage, isn't it the ID stage that needs to be under the same cycle as the WB stage of the inst. we are stalling for.

2. Why aren't the instructions `addi x0 x0 1` and `addi x0 x0 5` executed at all?

3. Why isn't there a data hazard between all the instructions except 7 to 8? I thought we needed to make sure the ID of the inst. was called in the same cycle as the WB stage of the previous inst. that wrote back to a register. So I thought we needed to add 2 `nops` between each inst. that used a register being written to in the prev. inst. such as the example provided in discussion.

4. This may answer the question above but why is the hazard from lines 7 - 8 a control hazard, I don't see how it would cause problems with the flow of execution. Is it because we did not take the branch and now we have a different value in `x0` than what we should've had if we did take the branch? I thought the CPU "loses" the instructions it executed after the branch if it incorrectly predicted a branch would not branch so the value of `x0` would be the same.

♡ 3 ...

 **Anonymous Oyster** 7mth #984fbc

I am also confused about this question. If the branch gets taken then do we just nop all the previous instructions as if they were carried out, but without saving the results? Would I be correct in assuming that if we didn't take the branch, there would be a lot more stalls (i.e. the IF of each addi x0 x0 aligning with the previous's WB)? If we're carrying out these instructions anyway only to no-op them later after the WB of the first branch, then how do we know to start instruction 7 in cycle 5? Wouldn't we be way behind since we already stalled so much while carrying out instructions 2-4?

Also, what do the Xs signify here? Just stages that were never executed due to the hazard? Why's there a WB right after the three X's for the third nop of instruction 8?

Edit: I think I sort of get it; when branch predictions fail, the pipeline just gets flushed no matter what and we proceed with the program. So in this case, since we were wrong, whether or not we stalled on the addi instructions doesn't really matter since we continue running the code as we normally would once we take the branch.

♡ ...

 **Jero Wang** STAFF 7mth #984abac

X is just flushed pipeline stages/stalls due to hazards.

For your first question, yes, we nop the instructions the same cycle we resolve the branch, so that their results are not saved. Yes, technically, it would have nops between each of the adds if the branch is not taken, though it would depend on if the system has optimizations for x0 (since x0 doesn't actually cause data hazards).

Your edit sounds correct - it doesn't matter here because it's never actually executed.

♡ ...

 **Jero Wang** STAFF 7mth #984abab

1. This is more of a notation issue, the end result should be the same. We don't support double pumping, so we need to wait for the first instruction's WB to finish before we can read it (in ID), so the IF lines up with WB.

2. They are not executed because by that time, our branch reached WB and our branch prediction was incorrect, so we go to the correct instructions.

3. This only is true if the first instruction is writing to the register read by the second instruction. In our case, line 9 is not reading t0, so there's no hazard with line 8.

4. Where does it say it is a control hazard? It should be a data hazard between those lines.

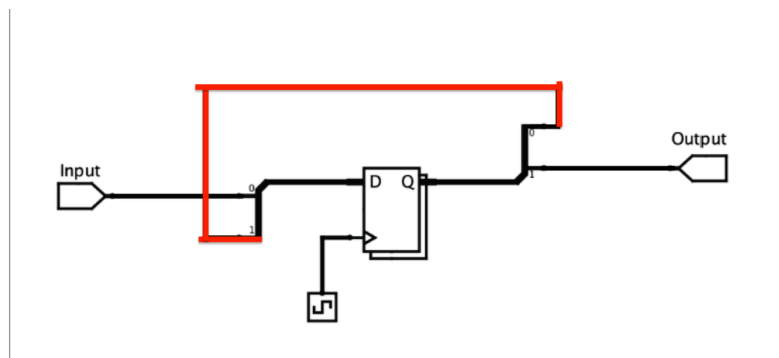
♡ ...

 **Anonymous Llama** 7mth #984eab

✓ Resolved

FA23-Final-Q10

i understand the input and output wires and how they relate to the register based on [#984dcf](#) but i'm not sure what this red wire that i've highlighted is doing? is it technically not needed since we're not using those bits for the input/output?



♡ ...



Justin Yokota STAFF 7mth #984eed

It's needed; bit 1 of the register is being driven by bit 0 of the register from the previous cycle. Without this path, you wouldn't have anything setting bit 1 of the register.

♡ ...



Anonymous Llama 7mth #984acad

why do we need to set bit 1 of the register? does that affect the output if we only care about the zeroth bit from the input? could you walk through some of the initial steps in the example provided and how it would go through the circuit?

♡ ...



Justin Yokota STAFF 7mth #984acca

↩ Replying to Anonymous Llama

The output is being driven by bit 1 of the register; if we don't set bit 1, the output would be undefined. Effectively, this circuit matches the behavior of the FSM; given a sequence of inputs, the register's internal value will change according to which state in the FSM you'd be in, and sends output according to the transition run.

Another way to think about this is that a 2-bit register is really just two flip-flops. Our goal is to delay a single-bit input by two cycles, which can be done by putting those flip-flops serially; this is attained by the given circuit (the "path" data takes through the flip-flops is a linear path).

♡ ...



Anonymous Llama 7mth #984dff ✓ Resolved

Q10.3 (2 points) What is the fewest number of states that an FSM solving this problem can have? Your answer must be an exact **integer**.

Solution: 4096. To get this answer, notice that to reference something from 12 inputs ago in an FSM, we must keep a record of that for at least 12 states. In other words, we must "encode" 12 bits of memory into our FSM states, requiring $2^{12} = 4096$ states.

FA23-Final-Q10

why is it 2^{12} ? is it because for every because 2^{12} is every possible combination of 12 prior bits?

♡ ...



Justin Yokota STAFF 7mth #984eee

Yes. You can show that each of the 2^{12} states yields a different output sequence if we start inputting all zeros, indicating that the states cannot be "collapsed".

♡ ...



Anonymous Llama 7mth #984dfe

✓ Resolved

FA23-Final-Q8

i understand why `andi t3 a0 3` is the same as `a0 % 4` but i was wondering what is the intuition / pattern behind how to derive this? $4 = 0b0100$ and $3 = 0b0011$ so it seems that if there is any one bit in the zeroth or first position, this guarantees that 4 would have a remainder the two one bits of three would give us what the remainder and zero out everything else. and since when dividing by 4, the possible remainders are 0, 1, 2, 3 so we use the highest one? is there a more straightforward way to think about this? thanks!

Q8 Cumulative: Unconditional RISC (9 points)

Assume we have a function, f , that takes in a 32-bit unsigned integer, x , as an argument. $f(x)$ is defined as follows:

$$f(x) = \begin{cases} x + 9 & \text{if } x \% 4 == 0 \\ x * 2 & \text{if } x \% 4 == 1 \\ x & \text{if } x \% 4 == 2 \\ x // 8 & \text{if } x \% 4 == 3 \end{cases}$$

Jero wants to write this function in RISC-V, but he couldn't get his CS61CPU's branch instructions to work! As a result, you may use any RV32I instruction **except** branch instructions.

Write a function, f , which accepts one argument x in `a0`, and returns $f(x)$. You may assume that there is no overflow.

```
1 f:
2  andi t3 a0 3
   Q8.1
```



Sophie Xie STAFF 7mth #984edf

yep, you got it! How I like to think about it is if you AND `0b11` with any number, say x , the output is going to be the last 2 bits of x , and the last 2 bits of x represents $x \% 4$.



Anonymous Llama 7mth #984dfb

✓ Resolved

```
1 bool more_even_simd(uint32_t* array, uint32_t n) {
2  vector counts = vec_setnum( 0 );
   Q5.1
3  vector mask = vec_setnum(1);
   Q5.2
4  for (uint32_t i = 0; i < n / 4 * 4; i+=4) {
   Q5.3
5      vector temp = vec_load(array + i);
   Q5.4
6      vector masked = vec_and(temp, mask);
   Q5.5
7      counts = vec_add(counts, masked);
   Q5.6
8  }
9  return (n / 2) >
   Q5.7
10     (vec_sum(counts));
   Q5.8
11 }
```

FA23-Final-Q5

why do we return $(n / 2)$ on the left side of the comparison? how does $n / 2$ represent the number of even number in the array? i put $n - \text{vec_sum}(\text{count})$ since $\text{vec_sum}(\text{counts})$ is the number of odd numbers there are



Daniel Wang 7mth #984eaa

ENDORSED

We can see this with an example! Say we have $n = 50$ elements total. If we have a majority of odd numbers, or $\text{vec_sum}(\text{counts}) \geq 25$, then we can return false. Otherwise, if $\text{vec_sum}(\text{counts}) < 25$, then we can return true. This is the exact same as returning $(n / 2) > (\text{vec_sum}(\text{counts}))$!

♡ ...



Anonymous Chimpanzee 7mth #984dfa

✓ Resolved

FA23 Q5.10

I am having trouble understanding how a data race will occur in this scenario leading to an incorrect answer. In this question, we are just adding the total on which values are even and which are odd. From my understanding, in this loop, filling in one index before another shouldn't impact the rest of the problem. Not sure if I understand this concept entirely correct but it seems that no matter which order the threads cover each "i", the even count and odd count should always be the same right?

♡ 1 ...



Anonymous Llama 7mth #984dfc

i have the same question

♡ 1 ...



Daniel Wang 7mth #984eac

Are you referring to Q5.9? I think the reason why is because `event_count` and `odd_count` are not private variables, but shared variables, meaning they are shared amongst all threads. This could cause threads to have an unsynchronized order of reading and writing variables.

For instance, if we have 2 threads, both could read the value of `event_count` as 0, find their respective `arr[i]` to be even, and simultaneously write the value of `event_count` as 1 back, resulting in `event_count` incorrectly being 1 instead of 2.

On a side note, we can fix this with a reduction! If you want to see what that looks like or have any questions, feel free to let me know.

♡ 1 ...



Anonymous Llama 7mth #984acbb

why would they simultaneously write back since with a context switch means one thread write at different times but just that it can be interlaced? even with unsynchronized reads and writes, the resulting counts should still be the same since each thread covers a different part of the for loop?

♡ ...



Anonymous Walrus 7mth #984def

✓ Resolved

Sp23 Q9

I am totally lost on this question. Can someone explain the part for mantissa? Why does it need to fit in 13 bits? And how to decide the values for DRES?

Solution: $\overline{A|B|C|(D\&(R|S))|(E\&S)}$

As noted in the hint, two distinct constraints exist, provided by the exponent and mantissa, respectively.

In order for an unsigned number to be representable, it needs to be small enough that its exponent fits within five bits. The maximum exponent of a 19 bit floating point number is $30 - 15 = 15$, taking into account that exponent 31 is reserved for infinities and NaNs. Including the implicit one and a maximized mantissa, our range is slightly less than double this exponent; thus we cannot represent any number that is 2^{16} or greater. This means that we cannot represent any number with any of bits ABC set to one.

In order for an unsigned number to be representable, it needs to be divisible by enough powers of two that its mantissa fits within thirteen bits. The smallest level of precision we can represent is $0b1.0000000000001$, which has a total of fourteen bits between the largest and smallest one bit. Thus, we cannot represent any number that has more than fourteen bits between the largest and smallest one bit. Since ABC must be zero by the above, we only have 16 bits that can be nonzero. Thus, we need at least two bits on the outside of this to be zero; one of DE , DS , or RS must be zero, or alternatively, none of the pairs DR , DS , ES can be both ones.

Putting this together, we get the equation $(A|B|C)$ for the exponent constraint and $((D\&R)|(D\&S)|(E\&S)) = ((D\&(R|S))|(E\&S))$ for the mantissa constraint. Putting this together and using De Morgan's law provides our final answer.

♡ 1 ...

 **Anonymous Cattle** 7mth #984ecc

exponent - 5 bits; mantissa (unsigned number) - $19 - 5 = 14$ bits

Under this setup, the "precision" bit is 13 bits away from the decimal point.

There will be a loss of precision for a number that has two digits more than 13 bits apart.

This leads to the above expressions involving D, R, E, S, since you need at least two bits on the side to be 0 to "squeeze" the other bits within a distance of 13.

♡ ...

W **Wil Rothman** 7mth #984dee ✓ Resolved

To verify-- if set-associative caches are out of scope this semester, then does that mean for any TIO question we see on the final, $I = 0$ or else it wouldn't be fully associative?

♡ ...

 **Jero Wang** STAFF 7mth #984abaa

Direct mapped caches (which are in scope) will have a nonzero I , but $I = 0$ means it is fully associative.

♡ ...

 **Anonymous Grouse** 7mth #984ded ✓ Resolved

Fa23-Final-Q4

for `arr[4] += arr[0]`, the solution's order is

1. read `arr[4]`

2. read `arr[0]`

3. write to `arr[4]`

Can I swap 1 and 2?

♡ ...



Jero Wang STAFF 7mth #984abde

The order is not specified by the C spec, but we will clarify the order for exams in the future.

For Fall 2023, we assumed the solution's order.



Anonymous Cattle 7mth #984deb

Resolved

Q5 The Lookup Box

(10 points)

Consider a system with a 32-bit virtual address space, 256B pages, and 16 MiB of DRAM as main memory. Provided below is the TLB and a portion of the page table. The TLB is fully associative and there is no data cache. The next free physical pages in main memory start at physical addresses 0x61DE00 and 0x61EF00, respectively.

Each PTE is 32 bits. Bit 31 is the valid bit, bit 30 is the dirty bit, bits 16 through 29 hold other metadata (not relevant for this question), and bits 0 through 15 hold the PPN.

Initial TLB State:

Tag (VPN)	PPN	Valid	Dirty
0x000000	0x23EF	1	0
0x000001	0xFFFF	0	0

Page Table:

Index	PTE
0x0	0x80AB23EF
0x1	0x80EE00C0
0x2	0x8123200A
0x3	0x3561CBA8
...	...
0xA	0xCAFFEEE0

SP23-Final-Q5. So in both the Page Table and TLB, the data are written as right-left 0-31st bit right? If so, when we take the PPN to put into the final answer, do we have to flip it to rearrange it in order of increasing bit number? Why or why not?



Jero Wang STAFF 7mth #984aaff

This is a 32-bit number, so the 0th bit is the rightmost bit (LSB) and the 31st bit is the leftmost bit (MSB). The final answer asks you for a hexadecimal number, so the LSB should still be on the right, so you do not need to flip it.





Anonymous Cattle 7mth #984dea

✓ Resolved

FA23-Final-Q9

My proposed solution is to flip bit 15 of the instruction or a0 a0 a1 to change it to or a0 a1 a1 . My solve_orion function is as follows:

```
uint32_t solve_orion(bool(*orion)(uint32_t)) {
    // Your code here
    uint32_t orout = orion(0); // Input doesn't matter
    if (!orout) return 0;
    return orout - 1;
}
```

This is not a solution on the guide, which I understand is a non-exhaustive list of possible correct answers. I decided to ask ChatGPT (*not cheating because this is for non-deliverable conceptual help*) and it told me that my solution is correct, but that *it would grade me incorrect because I did not follow the spirit of the problem*. While this response was kind of funny, it's also kind of concerning.

Out of curiosity, would course staff consider this a correct answer? If not, I would 100% be submitting a regrade request for this.

Note this is also a constant time, $O(1)$ solution as the runtime is independent of any input size.

♡ ...



Jedidiah Tsang STAFF 7mth #984dec

I don't think this gives you orion's number. This allows you to return 1 (since orion would always return true) but it wouldn't give you the favorite number

♡ ...



Anonymous Dunlin 7mth #984baaf

Is this in scope?

♡ ...



Jedidiah Tsang STAFF 7mth #984baba

↩ Replying to Anonymous Dunlin

yes

♡ ...



Anonymous Dragonfly 7mth #984ddf

✓ Resolved

SU23-Final-Q8

How do we know the standard bias is -7? Also, why does applying this suddenly make our max value 7?

♡ 2 ...




Jero Wang STAFF 7mth #984aafe

Standard bias (as listed on refcard is $-(2^{E-1} - 1)$ where E is the number of exponent bits, so $-(2^{4-1} - 1) = -7$.

The largest value that the exponents can be is $0b1111 = 15$, which after applying the bias, becomes 8. However, when the exponent bits are all 1s, it's infinity/NaN, so the largest non-infinity/NaN exponent is 7.

♡ ...

 **Anonymous Dragonfly** 7mth #984dde ✓ Resolved


For SU23-Final-Q7, is there some systematic way to go about doing these manually?

♡ 1 ...

 **Jero Wang** STAFF 7mth #984aafd

Usually a greedy algorithm works well, then making manual adjustments based on what you see as the bottleneck, and I don't think there's a systematic way of doing it (at least in polynomial time).

♡ 1 ...

 **Anonymous Dragonfly** 7mth #984ddd ✓ Resolved

SU23-Final-Q4.2 How do we know there are 3 entries? Similarly for 4.3, how do we know there are 2 entries and how did we calculate the bits for data?

♡ 1 ...

 **Jero Wang** STAFF 7mth #984aafc

Direct mapped means that the number of sets equals to the number of entries, and we have 1 index trit (for 4.2) or bit (for 4.3), so we have 3 and 2 entries respectively.

For data, it's a similar pattern - we have 2 offset trits/bits, which means 3^2 trytes and 2^2 bytes of data respectively, then we multiply by how many trits are in a tryte/bits in a byte to get the result.

♡ 1 ...

 **Anonymous Walrus** 7mth #984ddc ✓ Resolved

SP23-Final Is Q6 in scope?

♡ 1 ...

 **Jero Wang** STAFF 7mth #984aafa

[#984ad](#)

♡ 1 ...

 **Anonymous Cattle** 7mth #984daa ✓ Resolved

SP23-Final-Q2.1

Q2 IF Only ID Pipelined Better**(10 points)**

In Project 3, we implemented a RISC-V CPU with two stages; stage 1 included IF and stage 2 included ID/EX/MEM/WB. For this question, imagine instead that we implement a two-stage pipeline with a different split; stage 1 will include IF/ID and stage 2 will include EX/MEM/WB (IF/ID/EX/MEM/WB are defined equivalently to the pipelined CPU on the reference card).

For Q2.1 and Q2.2, assume the following delays for each component. Any component not listed is assumed to have a negligible delay.

Component	Delay
$\tau_{\text{clk-to-q}}$	35ps
τ_{setup}	20ps
Mux	75ps
Regfile Setup	20ps
Regfile Read	175ps
Immediate Generator	150ps
Branch Comparator	200ps
ALU	200ps
Memory Read	300ps

Q2.1 (3 points) What is the minimum clock period of this circuit, in picoseconds, to achieve correct behavior?

Solution: 855ps ($\tau_{\text{clk-to-q}}$ (35) + Immediate Generator (150) + Mux (75) + ALU (200) + Memory Read (300) + Mux (75) + τ_{setup} (20)).

The critical path occurs in stage 2 for a load instruction.

Grading: Partial credit was given for errors that showed conceptual understanding of what the critical path is, but excluded or included an extra component's timing. We did not give partial credit for excluding some components since we cannot clearly distinguish between a conceptual misunderstanding or a mechanical error.

In the EX stage, how to calculate the critical path of the BRANCH COMP path? My confusion is does the critical path involving BRANCH COMP ends there, or do we continue the path after it sends signal to the control logical all the way to PC Sel? If the path ends at BRANCH COMP we can basically disregard it right since it's just one component starting from EX.

♡ ...

S **Sophie Xie** STAFF 7mth #984edd

Yes, if the branch comparator is used, the time it takes for stage 2 to execute is going to be less than 855 ps (how long our critical path takes to execute). Thus, the branch comparator is not part of our critical path.

♡ ...

Anonymous Fox 7mth #984cff ✓ Resolved

SP23-Final-7.13

If you did: `result = &data`, is this equivalent to `*result = data`?

♡ ...

S **Su-Ann Ho** STAFF 7mth #984dda

No. `*result = data` goes to the location pointed to by `result` and places data at that address. Because the caller of the function and the function itself has the same `result` value that is not changed, the `data` can be accessed after the end of the function. By doing `result = &data`, you are changing the local variable `result` instead.

♡ ...



Anonymous Anteater 7mth #984cfd

✓ Resolved

FA23-Final-Q5.3

For step 4 of the solution, why does it specify $n / 4 * 4$? From my understanding, this essentially rounds n down to a multiple of 4 due to the floor division -- but the problem already specified that n was divisible by 4, so wouldn't just n also be correct (and simpler) here?

```

1 bool more_even_simd(uint32_t* array, uint32_t n) {
2   vector counts = vec_setnum( 0 );
3   vector mask = vec_setnum(1);
4   for (uint32_t i = 0; i < n / 4 * 4; i+=4) {
5     vector temp = vec_load(array + i);
6     vector masked = vec_and(temp, mask);
7     counts = vec_add(counts, masked);
8   }
9   return (n / 2) >
10      (vec_sum(counts));
11 }

```

♡ 1 ...



Catherine Van Keuren STAFF 7mth #984dcc

I looked up the rubric and n was considered a correct answer for this question.

♡ 4 ...



Anika Chandwani 7mth #984cef

✓ Resolved

FA23-Final-Q3.1

For this solution, shouldn't `addi t1 t0 2` start one cycle earlier since IF doesn't have anything to do with register values, so it should be able to start its cycle when the WB of the previous instruction reaches the ID of the next instruction?

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X					
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2									IF	ID	EX	MEM	WB	
9. xori t1 x0 6										IF	ID	EX	MEM	WB

♡ ...



Catherine Van Keuren STAFF 7mth #984dcd

Since `addi t0 x0 9` is currently in the IF stage, we have to wait until it has reached the ID phase before we can start the `addi t1 t0 2` instruction since they can't run on the same hardware at the same time.

♡ ...

A **Anika Chandwani** 7mth #984afdb
Wait, my question was why did we stall addi t1 t0 2, 3 times, shouldn't we only stall it 2 times because we only need the value of t0 in the ID stage of addi t1 t0 2?
♡ ...

Anonymous Cormorant 7mth #984afed
same question here!
♡ ...

T **Tyler Yang** STAFF 7mth #984bdbf
I'm not sure I follow - but we do have to stall 3 times because the ID stage has to come **after** the WB stage of the previous instruction (no double pumping is allowed in this specific question).
♡ ...

A **Anika Chandwani** 7mth #984bdca
↩ Replying to Tyler Yang
In lecture, the professor talks about how the regFile is write-then-read so then we only need 2 stalls. This question does not clarify the regFile, so how would we know if this is valid?

Solution 1: Stall pipeline:

- Requires extra pipeline state to prevent register writes on stalled stages.
- The correct instruction is stalled for two clock cycles.

The diagram shows a pipeline stage labeled 'WB' with a 'Reg' block. Below it is a box containing the number '9'. The name 'Yan, Yokota' is visible in the bottom right corner of the slide.

♡ ...

T **Tyler Yang** STAFF 7mth #984bdcb
↩ Replying to Anika Chandwani
Double pumping is defined as write-then-read. Since this question specifies no double pumping, we cannot assume write-then-read. Thus the write and read have to occur in 2 separate cycles.
♡ ...

A **Anika Chandwani** 7mth #984bdcd
↩ Replying to Tyler Yang

Ahh I see, is there a place in the lecture where we talk about double pumping? I can't seem to find it.

♡ ...

T **Tyler Yang** STAFF 7mth #984bdce

↩ Replying to Anika Chandwani

I'm actually not sure if it's covered in any lecture this year, but discussion 8, question 4.6 touches upon it.

♡ ...

A **Anika Chandwani** 7mth #984bdcf

↩ Replying to Tyler Yang

Thank you for the help!

♡ 1 ...

 **Anonymous Stork** 7mth #984cee ✓ Resolved


For sp23 Final Q2.3, why the other values in stage 2 are not required to have a pipeline register?

♡ ...

 A **Andrea Lou** STAFF 7mth #984cfb

The question defines the two pipeline stages to be IF/ID and EX/MEM/WB. You can think of this like proj3 CPU pipelining step, where all values that cross over between our two stages must be pipelined so that multiple instructions can run in parallel. The other values in stage 2 don't cross between different stages, so we don't need to save their value in a register.

♡ 1 ...

 **Anonymous Fox** 7mth #984cde ✓ Resolved

SP23-Final-Q5.4

Isn't the PTE at index 0x3 valid for the virtual address?

Page Table:

Index	PTE
0x0	0x80AB23EF
0x1	0x80EE00C0
0x2	0x8123200A
0x3	0x3561CBA8
...	...
0xA	0xCAFFEEE0

♡ 1 ...



A **Andrea Lou** STAFF 7mth #984cfa

index 3 has 0x3561CBA8. This can be translated to binary: 0b 0011 0101 ... where 0011 is the leading 3. Expanding it out helps to see that bit 31 is 0, so this entry is invalid.

♡ ...



Anonymous Fox 7mth #984cce

✓ Resolved

SP23-Final-Q4.2

Why is there thrashing in the accesses to a and b? Don't a and b have different tags and thus are located on different blocks?

For each set of accesses to a and b, the program experiences thrashing since the index of these blocks conflict. As a result, the accesses to a and b will always be misses (first two compulsory, then the rest are conflict).

♡ ...



J **Justin Yokota** STAFF 7mth #984cdc

Yes, but they share the same index. Because they share the same index, they'll end up competing for the one spot in our cache that could possible hold that type of block.

♡ 1 ...



Anonymous Cattle 7mth #984aeeb

According to this thread [#985aba](#) cache coherency is out of scope this sem, so to what extend do we have to know about thrashing?

♡ 3 ...



Annika Liu 7mth #984ccd

✓ Resolved

Sp23-Final-Q2

1. For Q2.1 and Q2.2, why is the Immediate Generator considered under stage 2 but not stage 1? Isn't it already parallel to the RegFile in the design of the circuit?

♡ 1 ...



Wesley Kai Zheng 7mth #984cdb

[#984aa](#)

♡ ...

D

Daichi Watanabe 7mth #984ccc

✓ Resolved

SP23-Final-Q9

I'm lost at the end of the first big paragraph in the solution. To recap, the number is:

- 0bABC DEFG HIJK LMNO PQRS

Since we're following all IEEE-754 conventions with a 5-bit exponent, the sign, exponent, and mantissa should be the following, respectively:

- A BC DEF G HIJK LMNO PQRS

The visible mantissa is 13 bits, but because of the implicit 1, it should be:

- 1G HIJK LMNO PQRS

So, in the largest case, the number to multiply $2^{(\text{exponent})}$ by should be:

- 1.1 1111 1111 1111 (with thirteen trailing ones after the radix point)

Unless I'm mistaken, the range represented by the maximized mantissa is **not** "slightly less than double this exponent" (presumably meaning $\text{double}(2^{15}) = 2^{16}$), but rather slightly less than 2^{17} :

- Highest num = $1.1\ 1111\ 1111\ 1111 * 2^{15} = 1111\ 1111\ 1111\ 1100 = 2^{17} - 4$

So, why is it the case that "we we cannot represent any number that is 2^{16} or greater"? Why is this number not 2^{17} ?

Q9 Cumulative: The Magnus Effect

(7 points)

Q9.1 (7 points) Write a Boolean expression that determines if a 19-bit unsigned integer can be expressed exactly as a 19-bit floating point number. For full credit, you may use at most 8 Boolean operators (|, &, ~).

Inputs: Bits A through S

Output: One bit. Output 1 if 0b ABC DEFG HIJK LMNO PQRS is an unsigned number that can be exactly represented as a 19-bit float which follows all IEEE-754 conventions, with 5 exponent bits (and a standard bias of -15). Output 0 otherwise.

Hint: both the exponent and the mantissa provide nontrivial constraints.

For partial credit, describe in English how you can determine if a 19-bit unsigned integer can be expressed exactly as a 19-bit floating point number (using the float representation described above).

Solution: $A|B|C|(D\&(R|S))|(E\&S)$

As noted in the hint, two distinct constraints exist, provided by the exponent and mantissa, respectively.

In order for an unsigned number to be representable, it needs to be small enough that its exponent fits within five bits. The maximum exponent of a 19 bit floating point number is $30 - 15 = 15$, taking into account that exponent 31 is reserved for infinities and NaNs. Including the implicit one and a maximized mantissa, our range is slightly less than double this exponent; thus we cannot represent any number that is 2^{16} or greater. This means that we cannot represent any number with any of bits ABC set to one.

In order for an unsigned number to be representable, it needs to be divisible by enough powers of two that its mantissa fits within thirteen bits. The smallest level of precision we can represent is 0b1.0000000000001, which has a total of fourteen bits between the largest and smallest one bit. Thus, we cannot represent any number that has more than fourteen bits between the largest and smallest one bit. Since ABC must be zero by the above, we only have 16 bits that can be nonzero. Thus, we need at least two bits on the outside of this to be zero; one of DE, DS, or RS must be zero, or alternatively, none of the pairs DR, DS, ES can be both ones.

Putting this together, we get the equation $(A|B|C)$ for the exponent constraint and $((D\&R)|(D\&S)|(E\&S)) = ((D\&(R|S))|(E\&S))$ for the mantissa constraint. Putting this together and using De Morgan's law provides our final answer.



Justin Yokota STAFF 7mth #984cdd

0b1111 1111 1111 1100 is $2^{16}-4$, not $2^{17}-4$. Also note that we're going the other way; 0bABCDEFGHIJKLMNOPS is an unsigned integer, and you're checking if it would be representable as a float.

1 ...



Daichi Watanabe 7mth #984dce

Thanks!

...



Anonymous Cat 7mth #984ccb

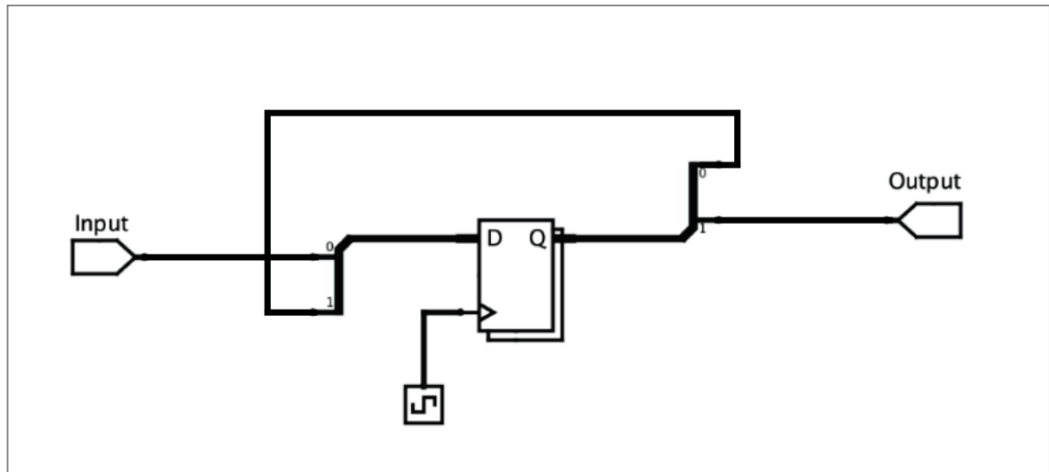
Resolved

Q10.2 (4 points) Fill in the circuit diagram below to implement this FSM. For full credit, your circuit must have the minimum possible clock period, assuming the following component delays:

$$\begin{aligned} t_{\text{AND gate}} &= 12\text{ps} \\ t_{\text{OR gate}} &= 15\text{ps} \\ t_{\text{NOT gate}} &= 4\text{ps} \\ t_{\text{XOR gate}} &= 31\text{ps} \end{aligned}$$

$$\begin{aligned} t_{\text{Register clk-to-q}} &= 10\text{ps} \\ t_{\text{Register setup}} &= 15\text{ps} \\ t_{\text{Bit splitter}} &= 0\text{ps} \\ t_{\text{Wire}} &= 0\text{ps} \end{aligned}$$

You may not use any other components. You may assume that the input and output connect directly to registers (for the purpose of determining the clock period), and that the register stores 2 bits. Your circuit does not need to "match" the states you use in your answer to Q10.1; it will be considered correct if its behavior matches the intended behavior described above.

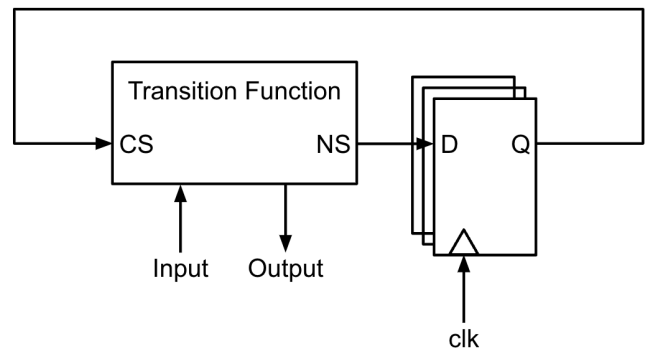


FSM: Circuit Format

S 61C

Spring

- To implement an FSM in a circuit:
 - Use a combinational logic block to receive input and current state, and send output and next state
 - Save the state in a register
- Acts as a very small datapath
- Similarly, any circuit with state elements can be considered an FSM (by combining all state elements into a single mega-register)



Fa23-Final-Q10.2

i'm honestly just really confused on why we aren't using logic gates for this solution. Can you explain the reasoning behind the solution? In addition where can I find more resources on how to implement FSM circuits because the only example i'm seeing right now is in Lecture 24 and its not that descriptive.

 **Su-Ann Ho** STAFF 7mth #984dcf

For the solution, we want to have the minimum possible clock period, meaning we want to use the fewest logic gates possible (as each gate contains a delay). This FSM is designed to output the second-most recently seen input, $N-2$. A key point to remember is that the register contains the two-bit State from the FSM (the 2-bit values in the circles).

In the FSM, we have a two-bit state, where the 0th bit represents the most recently seen input, and the 1st bit represents the second-most recently seen input. Therefore, if the output we are expecting is the second-most recently seen input, it solely depends on the 1st bit of the State. This State, in the circuit, is held by the register.

When the clock ticks, a new input comes in, meaning the previous most recently seen bit (0th bit of the output) should become the second-most recently seen bit (1st bit to the input), and the most recently seen bit is the new input. Because there is no need to do combinational logic for any of the inputs or outputs, the solution does not depend on any logic gates.

Some additional examples of FSM circuits can be found in [Fall 2019 Final Q5c](#), [Summer 2018 Final Q7 Part 2](#), and [Summer 2018 Midterm 2 Q1](#).

♡ 1 ...

 **Anonymous Cat** 7mth #984ddb

So if we had an FSM that needed to output the $N-10$ bit would we need to have a 10-wire splitter connected to D of the register?

Also, correct me if I'm wrong but given an input, the register stores the state before the current input (like the 2-bit value in the circle before the current circle).

♡ ...

 **Anonymous Zebra** 7mth #984aecf

In this question, why is index 1 connected to the output and 0 connected to the register, can it be index 0 connected to the output?

♡ ...

 **Anonymous Barracuda** 7mth #984bdaa

↩ Replying to Anonymous Zebra

same q


♡ ...

 **Anonymous Penguin** 7mth #984bdda

↩ Replying to Anonymous Zebra

In a bitstring, the bits are numbered in increasing order from right to left. So in our states, the 0th bit is the right number while the 1st bit is the left. We can't have them flipped.

♡ ...

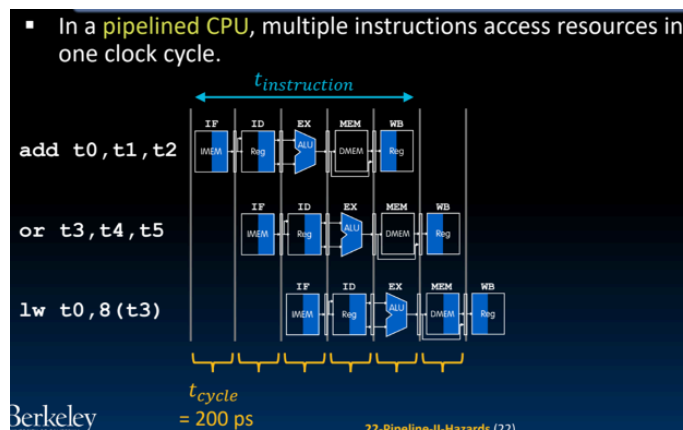
 **Anonymous Walrus** 7mth #984cbf ✓ Resolved

Sp23 Q2.3 Why do all values generated in stage1 need a pipeline register? What exactly is it? And I am still confused about the main differences between pipelined and non-pipelined CPU.

♡ 1 ...

 **Anonymous Walrus** 7mth #984cca

In this example, is it a five-stage pipelined CPU?



♡ ...

A **Andrea Lou** STAFF 7mth #984cfc

Yes, that is the five stage pipelined datapath. We pipeline in order to parallelize instructions, so multiple instructions can be working on different parts of the pipeline instead of waiting for one instruction to completely finish before executing another. See [#984cfb](#)

♡ ...

Anonymous Albatross 7mth #984cbe ✓ Resolved

FA23-Final-Q2

For 2.5, how come the BrUn does not matter? Would making it signed not mess up the register addresses?

♡ ...

A **Andrea Lou** STAFF 7mth #984cfe

The baleq instruction only compares $rs1 == rs2$. Normally, BrUn tells us if we are to make a signed comparison or not, but in this case we only care if the two values are equal or not. This means we just compare the bits, if they are the exact same then $rs1 == rs2$ is true, ignoring representation.

♡ ...

Anonymous Stork 7mth #984cbd ✓ Resolved

Hi, is Sp23 Q8 in scope?

♡ ...

M **Myrah Shah** STAFF 7mth #984cec

Yes, this would be similar to Homework 10.3 Q1 about optimizing task ordering.

♡ 1 ...

Anonymous Kangaroo 7mth #984caf ✓ Resolved

SP23-Final-Q7

Would this be a valid alternative solution for isBranchJump?

```
return (instruction & 96) == 96
```

My logic was that I wanted to check that the most significant bits of the opcode were 1s, since the opcodes for the jumps and branches all had 11 as the two most significant bits. With the and, if the instruction matches in those two spots, it should become exactly 96, which is why I then check the equality.

I understand the given solution was basically just checking if the 6th bit matched (1000000), but is checking the 5th bit as well okay?

♡ ...

A **Andrea Lou** STAFF 7mth #984dab
Yes, that should be fine

♡ ...

Anonymous Fox 7mth #984cae ✓ Resolved
SP23-Final-Q3

What is the logic behind the chunk_size?

♡ 1 ...

A **Andrea Lou** STAFF 7mth #984dac
We want to partition the total work we have to do between multiple threads to speed up the process. In this case we have num_rows amount of work to distribute between num_threads threads, so the chunk size (or how much work a single thread does) is just num_rows/num_threads

♡ ...

Anonymous Weasel 7mth #984cac ✓ Resolved
Fa23-Final-Q8:

are we allowed to use the t7 register even though it's not explicitly on our reference card? also, how do the lines 4-6 allow for us to execute either just line 7, 9, 11, or 13 based on our condition?

♡ ...

A **Andrea Lou** STAFF 7mth #984dbc
Please just use the 32 registers defined on the reference card. I think there was a clarification on that later in the exam, sorry about that.

First, line 1 has `andi t3 a0 3`, which ands the input `a0` with 3 to essentially get `a0 % 4`. This is then shifted left by 3 bits, or multiplied by 8. Then later on `auipc t7 0` does `rd = PC + imm`, so `t7` would have the value `PC + 0`, or just `PC`. This then allows us to "branch" to a location by jumping to `t3 + t7`, which is equal to $8 * (a0 \% 4) + PC$ (also plus the additional 12 bytes or 3 instructions to get us from line 4 to line 7 in order to set our PC properly)

♡ 1 ...

Anonymous Weasel 7mth #984bff ✓ Resolved
Fa23-Final-Q10, why doesn't each `Reg[0]`, `Reg[1]`, `Reg[2]`... and so on not take up any of the time

for our clock period?

♡ ...

A **Andrea Lou** STAFF 7mth #984dca
We calculate critical path from register to register, so once you go from `reg[0]` to `reg[1]` your calculation ends. The formula is `clktoq + setup + longestCL path`, but there are no combinational logic components so it is simply `clktoq + setup`

♡ ...



Anonymous Llama 7mth #984bfe

✓ Resolved

in sp23, 7.7

what is `num + 1` even doing here? I'm generally confused about what `codecopy` is doing here?

in one line we increment `codecopy++`; and in the other line we set its value to 0 (`*codecopy = 0`)?

♡ ...



Jero Wang STAFF 7mth #984aaef

We add an extra entry into `codecopy` compared to `code` because the final entry in our results needs to end with an element with a value of `0`. We therefore need to adjust the `for` loop accordingly to ensure that the last element is processed correctly.

`codecopy` is an array of RISC-V instructions, copied from the input. For every branch/jump we see, we replace that instruction with a `0`, essentially splitting this array of instructions into a bunch of 0-terminated subarrays. Then, we point each element of data to one of these subarrays.

We increment `codecopy` to traverse through this array and look for branches/jumps, and we set it to 0 when we find one as described above.

♡ 1 ...



Anonymous Llama 7mth #984bfc

✓ Resolved

Solution: $\overline{A|B|C|(D\&R|S)|(E\&S)}$

As noted in the hint, two distinct constraints exist, provided by the exponent and mantissa, respectively.

In order for an unsigned number to be representable, it needs to be small enough that its exponent fits within five bits. The maximum exponent of a 19 bit floating point number is $30 - 15 = 15$, taking into account that exponent 31 is reserved for infinities and NaNs. Including the implicit one and a maximized mantissa, our range is slightly less than double this exponent; thus we cannot represent any number that is 2^{16} or greater. This means that we cannot represent any number with any of bits ABC set to one.

In order for an unsigned number to be representable, it needs to be divisible by enough powers of two that its mantissa fits within thirteen bits. The smallest level of precision we can represent is $0b1.0000000000001$, which has a total of fourteen bits between the largest and smallest one bit. Thus, we cannot represent any number that has more than fourteen bits between the largest and smallest one bit. Since ABC must be zero by the above, we only have 16 bits that can be nonzero. Thus, we need at least two bits on the outside of this to be zero; one of DE , DS , or RS must be zero, or alternatively, none of the pairs DR , DS , ES can be both ones.

Putting this together, we get the equation $\overline{A|B|C}$ for the exponent constraint and $\overline{(D\&R)|(D\&S)|(E\&S)} = \overline{(D\&(R|S)|(E\&S)}$ for the mantissa constraint. Putting this together and using De Morgan's law provides our final answer.

SP23-Final-Q9

I understood how they got 15 as the maximum possible true exponent, but what does the implicit one mean to get 2^{16} ? is this calculating the value for denorms?

also, i'm still a bit confused on how they got the expression for the mantissa constraint

♡ ...



Justin Yokota STAFF 7mth #984dbf

$0b1.111\dots$ is slightly less than 2. $2 * 2^{15} = 2^{16}$

♡ ...



Anonymous Llama 7mth #984bfb

✓ Resolved

SP23-Final-Q7 i kind of forgot how double pointers work and confused why we assign `data[i] = codecopy` which is a pointer in line 7.8 because `data` is a double pointer. but why are double pointers equivalent to an array of pointers? so `data` points to an array (since arrays are similar to pointers) but every element in this array points to a pointer `codecopy` ? if we assign `data[i] = codecopy` and since we increment `codecopy`, essentially each index of the array `data` is pointing to a particular segment in `codecopy` which is a contiguous section in memory?

Solution:

Q7.1: instruction & 64

Q7.2: `i < n`

Q7.3: `isBranchJump(code[i])`

Q7.4: `sizeof(int*) * (num + 1)`

Q7.5: `sizeof(int)`

Q7.6: `sizeof(int) * n`

Q7.7: `num + 1`

Q7.8: `codecopy`


```
6 int splitCode(int* code, int n, int*** result) {
7     int num = 0; // total number of branches and jumps

8     for(int i = 0; _____; i++) {
9         num += _____;
10    }
11    int** data = malloc(_____);

12    int* codecopy = calloc(n+1, _____);
13    // Hint: You should not need any more memory allocations
14    memcpy(codecopy, code, _____);

15    for(int i = 0; _____; i++) {
16        data[i] = _____;
```

♡ ...

 **Anonymous Mandrill** 7mth #984fbf
i was confused about this as well

♡ ...

 **Jero Wang** STAFF 7mth #984aaee

As you mentioned, pointers are essentially arrays. If we have a `int **`, it means we have a pointer to `int *s`, or an array of `int *s`, which itself is an array of `int s`.

Yes, each element of `data` points to somewhere within the `codecopy` array, which itself is a contiguous section in memory.

♡ ...

 **Anonymous Boar** 7mth #984bfa

✓ Resolved

Q10.2 (4 points) Fill in the circuit diagram below to implement this FSM. For full credit, your circuit must have the minimum possible clock period, assuming the following component delays:

$$t_{\text{AND gate}} = 12\text{ps}$$

$$t_{\text{Register clk-to-q}} = 10\text{ps}$$

$$t_{\text{OR gate}} = 15\text{ps}$$

$$t_{\text{Register setup}} = 15\text{ps}$$

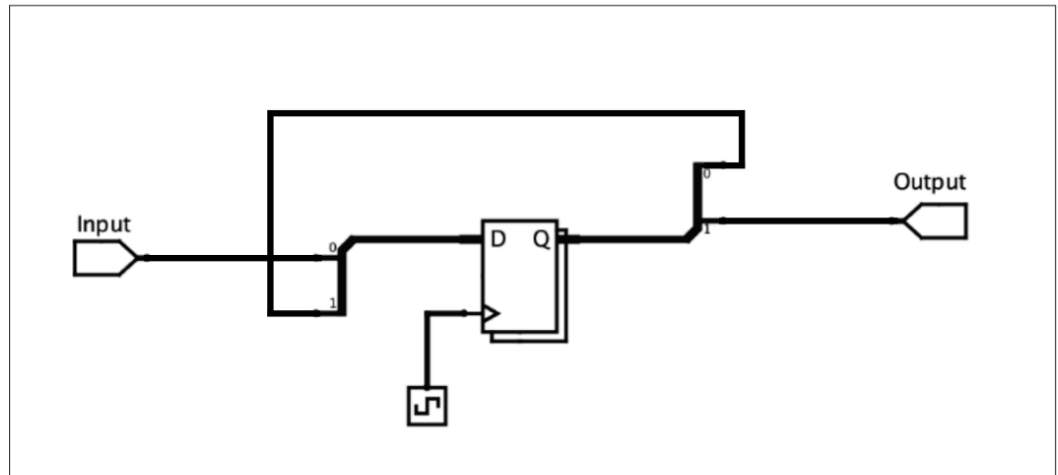
$$t_{\text{NOT gate}} = 4\text{ps}$$

$$t_{\text{Bit splitter}} = 0\text{ps}$$

$$t_{\text{XOR gate}} = 31\text{ps}$$

$$t_{\text{Wire}} = 0\text{ps}$$

You may not use any other components. You may assume that the input and output connect directly to registers (for the purpose of determining the clock period), and that the register stores 2 bits. Your circuit does not need to "match" the states you use in your answer to Q10.1; it will be considered correct if its behavior matches the intended behavior described above.



Unsatisfied with just delaying the input by 2 cycles, you decide to create an FSM that delays the input by 12 cycles (and outputs 0 for the first 12 cycles).

For example, if the input to the new FSM was 0b01 1011 1011 1000 1001,
you should output 0b00 0000 0000 0001 1011.

Q10.3 (2 points) What is the fewest number of states that an FSM solving this problem can have? Your answer must be an exact **integer**.

Solution: 4096. To get this answer, notice that to reference something from 12 inputs ago in an FSM, we must keep a record of that for at least 12 states. In other words, we must "encode" 12 bits of memory into our FSM states, requiring $2^{12} = 4096$ states.

idk how this works either; can someone explain

♡ 1 ...



Jero Wang STAFF 7mth #984aaed

To delay the input by 12 cycles, we need to store the last 12 inputs, which has a total of 4096 combinations, so we need that number of states.

In the future, please also tag your questions!

♡ ...



Anonymous Llama 7mth #984bed

✓ Resolved

```
1 // Returns true if instruction is a branch or jump instruction
2 bool isBranchJump(int instruction) {
3     return _____;
4 }
```

Q7.1

SP23-Final-Q7

when do we interpret true as 1? when i saw the comment that it returns true, i thought we had to get to a boolean expression that evaluates to 'true' or 'false' rather than 1 or 0, which makes it easier for adding later on

♡ ...



Justin Yokota STAFF 7mth #984cbc

"true" is defined as 1 internally, and "false" is defined as 0. This is actually in the form of two define statements in stdbool.h

♡ ...



Anonymous Boar 7mth #984bec

✓ Resolved

Q8 Cumulative: Unconditional RISC

(9 points)

Assume we have a function, f , that takes in a 32-bit unsigned integer, x , as an argument. $f(x)$ is defined as follows:

$$f(x) = \begin{cases} x + 9 & \text{if } x \% 4 == 0 \\ x * 2 & \text{if } x \% 4 == 1 \\ x & \text{if } x \% 4 == 2 \\ x // 8 & \text{if } x \% 4 == 3 \end{cases}$$

Jero wants to write this function in RISC-V, but he couldn't get his CS61CPU's branch instructions to work! As a result, you may use any RV32I instruction **except** branch instructions.

Write a function, f , which accepts one argument x in $a0$, and returns $f(x)$. You may assume that there is no overflow.

```
1 f:
2  andi t3 a0 3
   q8.1
3  slli t3 t3 3
   q8.2
4  auipc t7 0
   q8.3
5  add t3 t3 t7
6  jalr x0 t3 12
   q8.4
7  addi a0 a0 9
   q8.5
8  j f_end
9  slli a0 a0 1
   q8.6
10 j f_end
11 nop
   q8.7
12 j f_end
13 srli a0 a0 3
   q8.8
14 f_end:
15 ret
```



can someone explain what this code is doing line-by-line; I'm not really clear on how `auipc`, `jalr`, and `lui` instructions work

♡ ...

 **Anonymous Quetzal** 7mth #984bef

Same. I don't understand the use of the `no-op`

♡ 1 ...

 **Daniel Wang** 7mth #984cbb  **ENDORSED**

Before approaching this problem, keep in mind that we are trying to emulate the branch instruction, which checks for a condition and jumps to a certain address depending on whether the condition is satisfied or not.

In this case, `jalr` is used to

1. **jump** to a certain line (or address) determined by the ALU output $rs1 + imm$.
2. **link** (save) the next instruction on the next line in a register `rd`. This essentially saves the return address in `rd` if we want to return back to the line after the `jalr` call.

Therefore, we can think of Line 6 as our "branch instruction," using the output of $x \% 4$ to "jump" to the corresponding line that performs the corresponding operation on x (i.e. if $x \% 4 == 1$, do $x * 2$, etc.). This means that lines before Line 6 must be used to determine $x \% 4$ and set up the conditions for the `jalr` call. Additionally, we can see that Lines 7, 9, 11, and 13 are all of our "cases".

Line 2: How can we get $x \% 4$? When we approach arithmetic calculations in RISC-V, it can often be useful to try representing them with binary operations. Let's try some cases with binary numbers! $4 \% 4 = 0$, where $4 = 0b100$ and $0 = 0b000$. $5 \% 4 = 1$, where $5 = 0b101$ and $1 = 0b001$. $6 \% 4 = 2$, where $6 = 0b110$ and $2 = 0b010$. $7 \% 4 = 3$, where $7 = 0b111$ and $3 = 0b011$. This is where the pattern begins to form, the bottom two bits of the output of $x \% 4$ are equivalent to the bottom two bits of the input x ! In order to model this with a binary operation, try ANDing a bit with 1. $1 \& 1 = 1$ and $0 \& 1 = 0$. Another pattern forms! ANDing a bit with 1 will always return itself. Since we want just the bottom two bits, we can AND the input x with $0b11 = 3$. This is equivalent to `andi t3 a0 3`, where we save this output to a temporary register.

Line 3: Each case is separated by 2 lines (Line 7 to 9, 9 to 11, 11 to 13) or 8 bytes (remember that each line or instruction address is offset by 4 bytes from the next one in a 32-bit architecture like the RISC-V we use). One thing to note is that when we add to the PC counter, we are always adding in terms of bytes. Therefore, if we wanted to "jump" from one case to the next, we need to offset the PC by a value of 8 bytes. Currently, `t3` holds 0, 1, 2, or 3, which are all spaced out by 1 byte individually. In order to space them out by 8 bytes, we can simply multiply all by 8 (0, 8, 16, 24) and store this in `t3`. This is equivalent to `slli t3 t3 3`.

Line 4: In order to jump properly, we need to jump starting as close as we can to our cases. This is because `jalr` requires that we add an immediate to a register and set our PC to that, meaning we want that register to represent which case we jump to. Thus, we want to increase `t3`, and we can do this by adding an instruction address before our cases to it. A trick here is that we can store the current line or PC into a register `rd` by doing `auipc rd 0`. So, for now, we can store the current PC to a temporary register, equivalent to `auipc t7 0`.

Line 5: `add t3 t3 t7` adds the previous PC or instruction address on Line 4 to `t3`. We can

abstractly think of this by using cases. Say t_3 is 0, and t_7 is relatively speaking 4 (by Line 4). Then, t_3 will hold the instruction address of Line 4. If t_3 is 8, then t_3 will hold the instruction address of Line 6, since 8 bytes corresponds to 2 lines. If t_3 is 16, then t_3 will hold the instruction address of Line 8. If t_3 is 24, then t_3 will hold the instruction address of Line 10. Notice that Lines 4, 6, 8, and 10 are each 3 lines away from Lines 7, 9, 11, and 13, the original cases we eventually want to jump to.

Line 6: Thus, we can add 3 lines or 12 bytes to t_3 to reach each case! `jalr rd t3 12` will accomplish this by adding 3 lines to t_3 , but what do we set `rd` to? In this case, we actually don't care about the return address, since we are branching and not returning to a previous instruction after the branch. Thus, we can set it to `x0`, as `x0` ignores writes and always holds the value of 0. This is equivalent to `jalr x0 t3 12`.

The rest of the lines, specifically Lines 7, 9, 11, and 13, represent the operations performed on x . For Line 11, the `nop` is used since we simply return x itself in the case where $x \% 4 == 3$. Therefore, we don't modify `a0` and do nothing on this line.


If we want to confirm this implementation works, we can try some values out. $x = 4$ should return 13. Before Line 6, t_3 holds a value representing instruction Line 4, and after the `jalr` call, we set the PC to instruction Line 7, adding 9 to x , and terminating by jumping to `f_end` and returning with `ret`. Feel free to try some more!

Let me know if you have any questions on anything, this is definitely a complex problem and difficult to understand.

♡ 20 ...



 **Anonymous Chimpanzee** 7mth #984dfd
amazing explanation... thank you!

♡ 1 ...

 **Anonymous Spoonbill** 7mth #984fcd
Can you explain what `ret` does?

The reference sheet says it does `PC = ra`. Does it return to something in the function or terminate the function (i.e. what is `ra` set to?)



♡ ...

 **Daniel Wang** 7mth #984fea  **ENDORSED**
↩ Replying to Anonymous Spoonbill

I believe this jumps to the return address or `ra`, effectively leaving the function and returning to the address directly after the function call (see `jal*`). If any of this is unclear though, feel free to let me know and I can try writing an example!

*`jal` sets the `ra` to be the address or line directly after the line of the function call (`PC + 4`, where `PC` is the line of the function call and `+ 4` is the line after it). So when we call `ret`, we essentially are going back to this line, which we save inside a register specifically designated to save the return address, `ra`.

♡ ...

 **Anonymous Llama** 7mth #984beb  **Resolved**

SP23-Final-Q5.4

since the next free physical page has a physical address starting at 0x61DE00, we take the top four hex numbers since this is the PPN but we add it to the offset given by the VPN which is 60 because this is the offset (somewhere on that physical page) that we need to actually translate the virtual address?

Q5.4 (2.5 points) Virtual Address: 0x00000360

- TLB hit TLB miss, no page fault TLB miss, page fault

Physical Address:

Solution: Physical Address: 0x61DE60


The VPN is 0x000003 and the offset is 0x60. The VPN does not exist in the TLB, so we look for the corresponding PTE. There are no valid PTEs for this VPN, therefore, it is a page fault. The next available page starts at 0x61DE00, so our PPN becomes 0x61DE. The physical address is 0x61DE60.

♡ ...

 Jero Wang STAFF 7mth #984aaeb

Yep

♡ ...

 Anonymous Gaur 7mth #984bea ✓ Resolved

SP23-Final-Q6.2

Q6.2 (1 point) After a single machine M finishes its assigned portion of a map task in a MapReduce cluster, which of the following can happen immediately, regardless of the overall program state? Select all that apply.

- M can shut down without risking the success of the overall computation
- M can be assigned a new map task
- Data shuffling of the workload M just finished can begin
- The reduce task for the workload M just finished can begin
- None of the above


What is data shuffling?

♡ ...

 Jero Wang STAFF 7mth #984aadf

MapReduce is out of scope, but data shuffling refers to splitting the dataset randomly across multiple computers.

♡ ...

 Anonymous Gaur 7mth #984bdf ✓ Resolved

SP23-Final-Q6

Q6.1 (1 point) The OS running on a cluster of computers in a datacenter allows a single machine to read and write the memory and local disk of a remote machine in the same rack or array. According to lecture, which of the following are true? Select all that apply. ("farther away" means that the distance the data travels increases in steps, first to our local machine, then to a machine in our same rack, then to a machine in our same array.)

- As our CPU sends data to DRAM farther away, bandwidth increases
- As our CPU sends data to Disk farther away, bandwidth increases
- As our CPU sends data to DRAM farther away, latency increases
- As our CPU sends data to Disk farther away, latency increases
- We have higher latency to DRAM on an Array computer than to our own disk
- We have higher bandwidth to DRAM on an Array computer than to our own disk
- None of the above

I know this is out of scope, but I want to understand what is the question asking.

For answer choices 3 and 4, as our CPU sends data to DRAM farther away, the speed will decrease so the latency increases. (is this correct?)

For answer choices 1 and 2, does the bandwidth means the amount of data that we can send and receive each time? So the bandwidth doesn't change when we send data farther away.

And I'm not sure about 5 and 6.

♡ ...

J Jero Wang STAFF 7mth #984aaea

3 and 4: yes

1 and 2: yes

5 and 6: this is extremely dependent on the technology used, but in Dan's slides, it's because network latency and bandwidth is limited compared to our own disk, but that's not necessarily true in the world anymore.

♡ ...

Anonymous Crocodile 7mth #984bde ✓ Resolved

SP23-Final-Q9

Are we including the leading bit in the mantissa? I initially thought that the leading 1 was always assumed and we would not include that in the mantissa - so we would be able to represent a number with 15 bits (14 in mantissa + 1 assumed leading bit) between the largest and smallest one bit

♡ ...

J Jero Wang STAFF 7mth #984abdf

We have 13 bits of mantissa in this question - 19 bits total, 1 sign bit, 5 exponent bits. With the implicit 1, there are 14 bits overall.

♡ ...

 **Anonymous Crocodile** 7mth #984acec

I thought we wouldn't have a sign bit because we are working with unsigned numbers in this problem - should we always assume a sign bit regardless?

♡ ...

 **Jero Wang** STAFF 7mth #984addd

↩ Replying to Anonymous Crocodile

IEEE-754 states that there will be always be a sign bit, so we need a sign bit here as well since we're following IEEE-754 convention here.

♡ 1 ...

 **Anonymous Llama** 7mth #984bdc

✓ Resolved

SP23-Final-Q3

```
1 double matrix_average(double** matrix, int num_rows, int num_cols) {
2     double global_sum = 0.0;
3     vector sum_vec = _____;
4     for (int i = 0; i < num_rows; i++) {
5         for (int j = 0; j < _____; _____) {
6             vector values = _____;
7             sum_vec = _____;
8         }
9         for (int j = _____; j < _____; _____) {
10            global_sum += matrix[i][j];
11        }
12    }
13    global_sum += _____;
14    return global_sum / (num_rows * num_cols);
15 }
```

Solution:

Q3.1: `vec_set0()`

Q3.2: `num_cols / 4 * 4`

Q3.3: `j += 4`

Q3.4: `vec_load(matrix[i] + j)`

Q3.5: `vec_add(values, sum_vec)`

Q3.6: `num_cols / 4 * 4`

Q3.7: `num_cols`

Q3.8: `j += 1`

Q3.9: `vec_sum(sum_vec)`

I was wondering why 3.4 `vec_load(matrix[i] + j)` works because `matrix[i]` does `*(array + (i * sizeof(int)))` which evaluates to a value since it's dereferenced but then we add `j`, how does that result in an address that we need to pass into `vec_load`?

♡ ...

 **Anonymous Quetzal** 7mth #984bdd

`matrix` is a double pointer, so `matrix[i]` is an address not a "value"

♡ ...

 **Anonymous Llama** 7mth #984ffe

why is it not possible to do `matrix[i][j]`?

♡ ...

 **Jero Wang** STAFF 7mth #984aade

↩ Replying to Anonymous Llama

`matrix[i][j]` is equivalent to `*(matrix[i] + j)`. In this question, we're looking for the address, not the value itself.

♡ ...



Anonymous Gaur 7mth #984bdb

✓ Resolved

SP23-Final-Q3

Solution:

Q3.10: `#pragma omp parallel reduction(+: global_sum)`

Q3.11: `num_rows / num_threads`

Q3.12: `thread_num * chunk_size`

Q3.13: `(thread_num + 1) * chunk_size`

Q3.14: ~~`#pragma omp critical`~~

Q3.15: `global_sum += row_sum`

will this work?

♡ ...



Anonymous Spoonbill 7mth #984cdf **ENDORSED**

No. It is stated in the question that you cannot use the reduction keyword.

♡ ...



Anonymous Hedgehog 7mth #984bda

✓ Resolved

Fa23-Final-Q10

For 10.2, how does this circuit work? I see that we are sending the zeroth bit back to the input but I'm not sure I understand how it represents the FSM.

♡ 2 ...



Justin Yokota STAFF 7mth #984eea

The register keeps track of the internal state; at each step, the state gets updated to its new value, and an output is sent to the out pin.

♡ ...



Anonymous Weasel 7mth #984bcd

✓ Resolved

Fa23-Final-Q6

6.4: if we have a total of $2n$ tasks, why wouldn't we want our `task_queue` to hold all $2n$ tasks? if we only initialize it to have N tasks, doesn't this mean we won't process all $2n$ needed tasks?

6.7: if allow a process `p` to exit, what if we need it again for a later task?

♡ 1 ...



Justin Yokota STAFF 7mth #984cba

6.4: Since each of the later half of tasks require one specific earlier task to complete, we can't immediately start task $n+1$ necessarily. It is entirely feasible for task 1 to be delayed for some reason, such that task $n+1$ is slated to start before task 1 finishes.

6.7: The way this ends up working guarantees that if a process p exits, we won't need it again. This is because by the time we start sending exit commands, we necessarily have fewer tasks unstarted remaining than the number of threads we have available.

♡ ...



Anonymous Weasel 7mth #984ccf

thanks so for 6.4 we can only add a task $N + 1$ if N finishes since if we instead added tasks 0 through $2N-1$, task 0 could take a long time in which we process all tasks 1 through N (which shouldn't happen since N should only be processed if task 0 finishes) right?

♡ ...



Justin Yokota STAFF 7mth #984cda

↩ Replying to Anonymous Weasel
 $N+1$ if task 1 finishes, but yes.

♡ ...



Anonymous Weasel 7mth #984bcc

✓ Resolved

Fa23-Final-Q5,

Can someone explain why we are AND'ing together `temp` and `mask` and what exactly `counts` holds here?

Q5.10, if we just inserted "for" right after the original `#pragma omp paralell`, would this be "correct and faster than the naive"?

♡ ...



Daniel Wang 7mth #984bce

Notice that any odd number in binary form must have its least significant byte be 1 and that any even number must have it be 0. For example, $5 = 0b101$, $11 = 0b1011$, $12 = 0b1100$, $6 = 0b110$. This is because the LSB of a binary form is including or omitting $2^0 = 1$, while the rest of the bits represent powers of 2 (which are all even).

`temp` represents elements of the array while `mask` is used to apply to or "mask" `temp`. `mask` holds four 1s, where AND'ing 1 with an even number produces 0, and AND'ing 1 with an odd number produces 1. Therefore, an element of `counts` will have 0 if its corresponding element `temp` is even, and 1 if odd. This is why we `vec_sum(counts)` after iterating through all numbers, as this will represent the total number or count of odd numbers we have!

For Q5.10, I believe that's right, since we'll be able to parallelize the `for` loop and achieve a speedup (overhead would be minimal here compared to the magnitude of the speedup).

Edit: Lines 17 and 18 would need to be changed to `even_count += evens` and `odd_count += odds`, see below

♡ 2 ...



Anonymous Hedgehog 7mth #984bee

For Q5.10, if there was a "for" right after the original `#pragma omp parallel`, I think we also have to do `even_count += evens` and `odd_count += odds` to achieve speedup right? Since `even_count = evens` and `odd_count = odds` would incorrectly set the global variables to a local variable of each thread which are just parts of the overall loop?

♡ ...



Daniel Wang 7mth #984ceb

↩ Replying to Anonymous Hedgehog

Yes! That makes sense, since we would need to add the individual threads results to a global `even_count` and `odd_count` in the critical section. Thank you for the correction.

♡ ...



Anonymous Walrus 7mth #984eae

Why "AND'ing 1 with an even number produces 0, and AND'ing 1 with an odd number produces 1"? For example, if AND'ing 1 with 6 (0b110), shouldn't we still get 0b110? Do we just look at the least significant bit and why?

Edit: does 1 here only means 0b 00..01?

♡ ...



Daniel Wang 7mth #984ecf

↩ Replying to Anonymous Walrus

Your edit is right! It should just be the value 1 in binary form, or `0b1`.

♡ ...



Anonymous Weasel 7mth #984bbf

✓ Resolved

Fa23-Final-Q4

what is the significance of saying that we have a 16 byte cache with 4B blocks? Does this mean that when we first do `arr[0]` and have our compulsory miss, we can't load in `arr[1]`, `arr[2]`, `arr[3]`? I was trying to refer back to the homework where we did this and I think I remember that when we loaded in some number into our cache, we also technically loaded in the next 4 numbers? Why don't we do that in this case?

Q4.4, do we update the LRU only when there's a miss and eviction here or also when we have a hit? And if it's asking for the **last** iteration, why don't we just calculate it for when `i = 6` instead of `i = 4` like in the solution?

♡ 1 ...



Jero Wang STAFF 7mth #984dcb

Since each block is 4B, we can only store one `int32_t` in each block, so therefore, when we access `a[0]`, we only actually get `a[0]` and nothing else. If the block size was larger, we'd be loading the following elements.

When there's a hit, you should also update your LRU counter to indicate that the entry that was just accessed was most recently accessed. We showed the example of `i = 4` for simplicity, as all iterations past that point are the same format ("The key insight here is to notice that once we start accessing `arr[4]` and greater, we begin to have to evict members. For ease of explanation, we consider the 5th iteration of the loop below."). Calculating it for `i = 6` would require going through `i = 4` and `i = 5` individually.

♡ ...



Anonymous Barracuda 7mth #984bcdf

For say the line `arr[5] = arr[5] + arr[0]`, do you read right to left or left to right?

♡ ...



Anonymous Quetzal 7mth #984bad

✓ Resolved

FA23-Final-Q2:

Why is it necessary for the ALU to compute $PC + \text{offset}$ always? If the branch is not taken, why do we still need to calculate this? Or is this done before the conditional is checked? I noticed the branch comp is before the ALU in the datapath. Is this why?

Q2.2 (1.5 points) RegWEn

- | | |
|--------------------------------------|--|
| <input type="radio"/> Write disabled | <input type="radio"/> Doesn't matter |
| <input type="radio"/> Write enabled | <input checked="" type="radio"/> None of the above |

Solution: The value of RegWEn is determined by the output of the branch comparator (since we only update `rd` if the branch is taken), so it is not a constant, therefore "None of the above".

Q2.3 (1.5 points) ASel

- | | |
|--|---|
| <input type="radio"/> <code>rs1</code> | <input type="radio"/> Doesn't matter |
| <input checked="" type="radio"/> <code>PC</code> | <input type="radio"/> None of the above |

Solution: Since the ALU needs to compute $PC + \text{offset}$, ASel must be `PC`. This is identical to branch instructions.

♡ 1 ...



Jero Wang STAFF 7mth #984dbe

There's no signal to the ALU regarding if the branch is taken or not, so we must always compute it. This is similar to how for branches, we always compute $PC + \text{imm}$, regardless of if the branch is taken or not.

♡ 1 ...



Anonymous Weasel 7mth #984baa

✓ Resolved

Fa23-Final-Q3:

How do we know exactly if a control hazard will occur? is it only when we always take the branch?

Also, the solutions don't seem to answer the question for how many cycles it would take to execute line 9 and also how many stalls we need. Where can we find that answer?

♡ ...

E Eddy Byun STAFF 7mth #984bba

Note the problem description states that the CPU will always predict that branches are not taken. This means that if a branch does happen, we have to stall since our CPU assumed we'd take the next instruction.

For 3.1, it takes 14 cycles and 3.2, it takes 11 cycles.

For 3.1, both the control and data hazards require 3 cycles of stalling.

For 3.2, the control hazard requires 3 cycles of stalling (there's no data hazard in 3.2)

♡ ...

Anonymous Weasel 7mth #984bbc

so when the solution notes that "the branch will always be taken" but the problem description says that we predict that branches will not be taken, what does this exactly mean in terms of how we execute this? Since line 1 is true, don't we always have to jump to label and never run lines 2-6?

also, why are some stages in the cycle for noops X'd out while some like IF and WB still written out?

♡ ...

E Eddy Byun STAFF 7mth #984bbe

↩ Replying to Anonymous Weasel

The distinction is that the CPU will predict that all branches are not taken. For this specific RISC-V code, the branch is always taken `beq x0 x0 Label`, but the CPU will still predict that the branch is not taken.

Line 8 of the solutions for 3.1 should have an X at cycle 12 - sorry that's a typo

♡ ...

Anonymous Sardine 7mth #984caa

↩ Replying to Eddy Byun

Will we ever have a situation where the CPU predicts that branches are taken? If we do, what will that look like?

♡ ...

Wesley Kai Zheng 7mth #984cea

I am slightly confused as to why 3.1 is not 15. If we are not double pumping, in order to get the correct address to fetch instructions after taking the branch, do we need to wait until the next cycle after the `beq` WB stage has ended? Otherwise, how can we guarantee that we would get the correct PC address from which we fetch instructions? If that is the case, should we not stall for another cycle?

♡ ...

Anonymous Weasel 7mth #984aff ✓ Resolved

Fa23-Final-Q2.7

Why don't we need to add another WriteData and WriteIndex input to the RegFile and also allow the ALU to send out more than 1 output and update any relevant selector logic? Is there a general rule to follow with these kinds of questions?

♡ ...

E Eddy Byun STAFF 7mth #984bae

We're only writing to 1 register rd , so we don't need an additional WriteData and WriteIndex.

The ALU is calculating one address ($rs1+imm$), so we don't need the ALU to output more than 1 thing.

Unfortunately, there aren't any concrete general rules to follow, but for starters, I would recommend taking a look at the datapath (we provide one in the ref sheet!), and see where we may need changes in the datapath and control signals.

For this problem, notice that we have an immediate, $rs1$, $rs2$, and an rd - there aren't any instructions in our current datapath that has 4 of these, so we know we probably need a new instruction type and immediate generator.

The instruction is also doing a simultaneous load from and load to DMEM, which is also not currently supported in our datapath, so we probably need to allow for this behavior.

♡ ...

 **Anonymous Weasel** 7mth #984bbb

Thank you! what is exactly happening (or what does it mean) when we do $*(rs1+imm)$ here? Also, are we not writing to $*(rs1+imm)$ in the second line?

Also, is there an example of a case when we would add another write data and write index input to the regfile?

♡ ...


 **Jero Wang** STAFF 7mth #984dbd

↩ Replying to Anonymous Weasel

$*(rs1 + imm)$ refers to accessing the bytes located at address $rs1 + imm$. We are writing to that address in the second line, so we need to change DMEM. Since this is not writing to a register, we don't need to make changes to the write ports of the regfile.

An example would be an instruction that needs to write to two registers in the same cycle.

♡ ...

 **Anonymous Weasel** 7mth #984afe ✓ Resolved

Fa23-Final-Q2:

Since the ALU needs to compute $PC + offset$, wouldn't we want the PCSel to take in ALUOut?

Also, how do we know for sure if we want BrUn to be signed or unsigned?

♡ ...

 **Eddy Byun** STAFF 7mth #984bac

The PC should only be updated to the ALUOut when $rs1 == rs2$ otherwise, we should take $PC + 4$. As a result, the PCSel is not going to be constant.

In this case, it doesn't matter if we do a signed or unsigned comparison because we're checking for equality ($rs1 == rs2$). Lecture 20 Slide 19 may be helpful for this!


♡ ...

 **Anonymous Chimpanzee** 7mth #984afd ✓ Resolved


SU23-Final-Q5.10

Are two-level hierarchical page tables in scope? In lecture 35, it was marked as [Extra].

♡ 1 ...

 **Eddy Byun** STAFF 7mth #984bab
two-level page tables are out of scope

♡ 1 ...

 **Anonymous Wombat** 7mth #984afc ✓ Resolved

SU23-Final-Q2 pt 1 I am not sure how this works. How is xor effective here and how is it just getting the least significant bit instead of the entire number at a0? Also how is add also acceptable?

Part 1: Implement `calculate_parity`, a RISC-V function that takes in a 4-byte input in `a0` and calculates the parity of its bits, returning in `a0` either 1 for odd parity or 0 for even parity. For example:

Input: 0b 0000 0000 0000 0000 0000 0010 1101 0111	Output: 1
There are seven 1 bits (an odd number), so this number has odd parity, so the return value is 1.	
Input: 0b 0000 0000 0000 0000 0000 0010 1010 1101	Output: 0
There are six 1 bits (an even number), so this number has even parity, so the return value is 0.	

`calculate_parity` must follow calling convention.

```
1 calculate_parity:
2     li t1 0           # t1 holds current parity value
                       # in least significant bit
3 loop:
4     xor t1 t1 a0      # add also acceptable
                       # adds the current least significant bit
                       # in a0 to the parity value in t1


5     srli a0 a0 1     # shift to the next bit in a0
                       # it's important here that this is
                       # a logical shift, since using a
                       # arithmetic shift would possibly
                       # put us in an infinite loop
                       # if bit 31 is set

6     bne a0 x0 loop   # loop if there are more than 1 bit left in a0
                       # 0 bits will never affect parity


7     andi a0 t1 1     # we only want the one parity bit
                       # in bit 0 of t1

8 jr ra
```

♡ ...

 **Anonymous Gnat** 7mth #984cab
Isn't parity out of scope?

♡ ...

 **Jero Wang** STAFF 7mth #984dba
ECC is not in scope, though bitwise operations and RISC-V are in scope.

♡ ...

 **Jero Wang** STAFF 7mth #984dbb

xor gets the entire number here, but we don't care about the upper 31 bits, and those bits are removed in line 7 with the andi instruction.

♡ ...



Anonymous Cat 7mth #984aef

✓ Resolved

Fa23-Final-Q4.4

Q4.4 (2.5 points) What is the hit rate for the **last** iteration of the **for** loop, using an **LRU** replacement policy?

For this question why is the last iteration 4 instead of 5? Also is there a faster way to calculate the hit rate intuitively rather than just write out the LRU table for every iteration?

Q4.4 (2.5 points) What is the hit rate for the **last** iteration of the **for** loop, using an **LRU** replacement policy?

Solution: 7/12. The key insight here is to notice that once we start accessing `arr[4]` and greater, we begin to have to evict members. For ease of explanation, we consider the 5th iteration of the loop below. Listing out the 12 accesses, we have the below cache state at the start of the iteration:

Data	LRU
<code>arr[0]</code>	3
<code>arr[1]</code>	2
<code>arr[2]</code>	1
<code>arr[3]</code>	0

1. Read `arr[4]`. This evicts `arr[0]`, resulting in a miss.
2. Read `arr[0]`. This evicts `arr[1]`, resulting in a miss.
3. Write to `arr[4]`. This results in a hit.
4. Read `arr[4]`. This results in a hit.
5. Read `arr[1]`. This evicts `arr[2]`, resulting in a miss.
6. Write to `arr[4]`. This results in a hit.
7. Read `arr[4]`. This results in a hit.
8. Read `arr[2]`. This evicts `arr[3]`, resulting in a miss.
9. Write to `arr[4]`. This results in a hit.
10. Read `arr[4]`. This results in a hit.
11. Read `arr[3]`. This evicts `arr[0]`, resulting in a miss.
12. Write to `arr[4]`. This results in a hit.

Counting our hits, we have 7 hits and 5 misses, leading to our above hitrate.

♡ ...



Andrew Liu STAFF 7mth #984afa

Each iteration is 12 total accesses. Remember that `a += b` is in total 3 accesses (read `a`, read `b`, write `a`)

The intuition for this problem is mostly to find a pattern and then extrapolate it — in this case, the hit rate for all iterations where $i > 3$ have the same hit rate

♡ ...



Anonymous Weasel 7mth #984aed ✓ Resolved

Fa23-Final-Q1.3: I know that we are supposed to multiple the L2 miss rate by "main memory access" but I thought that this value was usually the number of times we access the main memory but in this case we made it equal to the hit time instead. How do we know when to use that instead?

♡ ...



Daniel Wang 7mth #984bcf

I could be wrong and I want to check my understanding as well, but I'm not sure what you're referring to by the number of times to access main memory. Do you mean the number of clock cycles? That is also another way of measuring hit time. I think when calculating AMAT hit times are always used.

♡ ...



Jero Wang STAFF 7mth #984daf

Are you referring to the third line of the math in the solutions? I believe that's a typo and it should be MR instead of HR.

♡ ...



Anonymous Wallaby 7mth #984aeb ✓ Resolved

su23-final-q5

For 5.4-5.9 how do we know the VPN is supposed to be 12 bits? Am I missing something in the instructions?

♡ 1 ...



Andrew Liu STAFF 7mth #984afb

The VPN is the number of bits in a virtual address minus the number of bits in the page offset. Given the virtual memory space and page size you should be able to calculate it.

Hint: n bits can count 2^n things. How can you use this to determine the length of a memory address given a virtual memory space? How about the number of bits in a page offset?

Hint 2: We desire to count the number of bytes to determine these lengths.

♡ ...



Anonymous Wallaby 7mth #984bbd

I understand how to do Q5.1-Q5.3 correctly, but when it tells us to assume different lengths for Q5.4 and onward does that not mean we should assume different sizes for VPN, PPN, offset etc? Or should we continue to assume we have 12 bits of offset?

♡ 1 ...

J **Jero Wang** STAFF 7mth #984dae

← Replying to Anonymous Wallaby

Yeah, we're still working with 4KiB pages for the rest of the question. This was clarified during the exam

| For Q5, assume a 4KiB page size for the entire question.

♡ 2 ...

 **Anonymous Pigeon** 7mth #984ade ✓ Resolved

fa23-final -Q8:

i do not understand lines 4-6 for the solution. i understand for lines 2-3 we are setting t3 to $x\%4$. i get that lines 7-13 are the different "branches" that the program can take depending on the value of t3, but i dont get the jumping 12 bytes + t3 since t3 can only be 0-3. if you jump 13 bytes it would only jump 1 more line? i know i'm missing something but i cant tell what

♡ ...

 **Anonymous Wallaby** 7mth #984aec

In line 3 when we slli by 3 thats the same as multiplying by 8. Since each "branch" is two instructions aka 8 bytes to skip. The 12 is from the difference from the state of PC at the auipc call and the jalr call. Then we just go a multiple of 8 based on which scenario we are in.

So like if $x \% 4 = 1$, when we slli by 3 we get 8. The auipc gives 12 since we are at line 4 but technically 3 as we ignore labels. Then when we add everything together its: 12byte (PC) + 8byte ($x*4$ slli by 3) + 12byte (imm) = 32byte which is line 9.

♡ ...

 **Anonymous Llama** 7mth #984ace ✓ Resolved

sp23 final 2.2


I thought in 2.1 we included the immediate generator even though its a component from stage 1 b.c we need to uses the imm gen values for the alu. However in 2.2, it says we can just omit the imm generator from 2.2. Little confused about this?

♡ ...

 **E Eddy Byun** STAFF 7mth #984acf

[#984aa](#)

♡ ...

 **Anonymous Goldfish** 7mth #984acc ✓ Resolved

SP23-Final-Q5.4

Why is there no valid PTE for the VPN of 0x00003? Why is the answer not 0xCBA860?

Page Table:

Index	PTE
0x0	0x80AB23EF
0x1	0x80EE00C0
0x2	0x8123200A
0x3	0x3561CBA8
...	...
0xA	0xCAFFEEE0

Initial TLB State:

Tag (VPN)	PPN	Valid	Dirty
0x000000	0x23EF	1	0
0x000001	0xFFFF	0	0

Q5.4 (2.5 points) Virtual Address: 0x00000360

- TLB hit
 TLB miss, no page fault
 TLB miss, page fault

Physical Address:

Solution: Physical Address: 0x61DE60


The VPN is 0x000003 and the offset is 0x60. The VPN does not exist in the TLB, so we look for the corresponding PTE. There are no valid PTEs for this VPN, therefore, it is a page fault. The next available page starts at 0x61DE00, so our PPN becomes 0x61DE. The physical address is 0x61DE60.

♡ ...

 **E** [Eddy Byun](#) STAFF 7mth #984ada

The valid bit is 0 for the entry at Index 0x3!

♡ 1 ...

 [Anonymous Wallaby](#) 7mth #984abe ✓ Resolved

SP23-Final-Q9

Could I get a thorougher explanation on why bits ABC must be zero? Thanks!

♡ ...

 **J** [Justin Yokota](#) STAFF 7mth #984abf

If those bits were nonzero, the unsigned number would exceed the maximum possible non-infinite float in this format.

♡ ...

 [Anonymous Wallaby](#) 7mth #984aca

Oh I was thinking of the number as a float rep not as the actual integer for some reason lol. Got it now thanks

♡ ...

 [Anonymous Moose](#) 7mth #984abd ✓ Resolved

Is Su23 Q2 hamming codes out of scope?

♡ ...

 **C** [Catherine Van Keuren](#) STAFF 7mth #984acb


Yes, hamming codes weren't covered this semester


♡ 1 ...


 [Anonymous Zebra](#) 7mth #984aac ✓ Resolved

Is Fa23-Final-Q10 in the scope of the final exam?

♡ ...

 **E** **Eddy Byun** STAFF 7mth #984aae
Yep, it's in scope
♡ ...


 **Anonymous Zebra** 7mth #984aaa ✓ Resolved
FA23-Final-Q4.1Is 8-way associative cache going to be in the scope of the class?
♡ 1 ...


 **M** **Myrah Shah** STAFF 7mth #984aab
From above:


As a note, some questions will be out of scope because set-associative caches are out of scope this semester.

But note: [#986bc](#), you should be able to solve similar problems for the types of caches covered.

♡ ...


 **Anonymous Zebra** 7mth #984da ✓ Resolved
Is the concept of overhead out of scope?
♡ ...


 **E** **Eddy Byun** STAFF 7mth #984db
Overhead is in scope
♡ ...


 **Anonymous Fox** 7mth #984cf ✓ Resolved
FA23-Final-Q3.1

Why are instructions 2-4 loaded in, but not 5-6?

♡ ...

 **E** **Eddy Byun** STAFF 7mth #984dc
Instructions 2-4 are loaded in because the PC doesn't get updated to the part of the program we want to branch to until Instruction 1 reaches the MEM stage. As a result, we load in instruction 2,3, and 4. Instructions 5 and 6 don't get loaded in because the PC has been updated to the part of IMEM we branched to.
♡ ...

 **Anonymous Zebra** 7mth #984dd
Why does instruction 2 stall at cycle 3 since the first instruction does not write anything back to register?
♡ ...

 **Anonymous Zebra** 7mth #984ff
← Replying to Anonymous Zebra

Why does the 7-line instruction start executing in the fifth cycle instead of the fourth cycle, given that during the third cycle, the Branch Comp is executed for the first line and the destination is known by the fourth cycle? Do we need to wait for three cycles if the instruction is J?

♡ ...

 **Anonymous Moose** 7mth #984abb

↩ Replying to Anonymous Zebra

But after cycle 3 right branch comparator should have completed and we would know not to take the branch right

♡ ...

E **Eddy Byun** STAFF 7mth #984adc

↩ Replying to Anonymous Moose

At cycle 3, the branch comparator gets executed for the `beq x0 x0 Label` instruction, and we do take the branch.

♡ ...

E **Eddy Byun** STAFF 7mth #984adb

↩ Replying to Anonymous Zebra

[#984ab](#) - we calculate where we're branching/jumping to in the EX stage, but we actually write this to the PC in the MEM stage

♡ ...

E **Eddy Byun** STAFF 7mth #984add

↩ Replying to Anonymous Zebra

We need to turn the second instruction into a nop because in Instruction 1, we take the branch, meaning the next instruction we execute should be the instruction at `Label`

♡ ...

 **Anonymous Penguin** 7mth #984bddb

↩ Replying to Eddy Byun

What does the process of turning into an instruction into a nop look like, is it just like since we know the value of `BrEq` at the beginning of the EX stage, we can use pipelining to then send a nop input into IMEM into the next instructions? Also is this why the instructions are turned only into nops after IF, because IMEM has to decode the nop signal first? And in the 8th line of the table in the solution, why is the WB stage of `addi` still present and not X'ed out?

♡ ...

 **Anonymous Zebra** 7mth #984ce

✓ Resolved

FA23-Final-Q1.2: I don't really understand the concept of overhead. How does it relate to multithreading and Amdahl's Law?"

♡ ...



Justin Yokota STAFF 7mth #984acd

Overhead is any additional time cost incurred by an optimization; the concept is borrowed from business management, where overhead refers to the fixed costs incurred just by running a business (like rent, maintenance costs, etc.). Any program overhead puts a minimum on the time it takes to run our code (regardless of input size), and therefore eats into our maximum speedup; this can be quantified through Amdahl's law, though it's easier in my opinion to treat the math more as a word problem rather than an equation. Multithreading is just one of many possible optimizations, but is notable for being one of the first ones we discuss that incurs a nontrivial overhead cost. Setting up a multithreaded environment costs thousands of clock cycles, so small problems are often not worth it to parallelize.



Anonymous Fox 7mth #984cd

✓ Resolved

FA23-Final-Q2.7

Are atomic loads in scope?



Eddy Byun STAFF 7mth #984de

Atomics are in scope. I'll note that this question is dealing with changes to the datapath and not so much atomics.



Anonymous Crocodile 7mth #984cc

✓ Resolved

SP23-Final-Q7

I understand that we are trying to determine whether an instruction is jump/branch by its opcode (so this is checking whether we have 1 in the 7th bit position). However, wouldn't this return true for ebreak/ecall as well?

Q7.1: instruction & 64



Eddy Byun STAFF 7mth #984df

Note the first bullet point! You may assume that all instructions are valid RISC-V base instructions, and that there are no pseudoinstructions, ecalls, or ebreaks.



Anonymous Gaur 7mth #984cb

✓ Resolved

FA23-Final-Q5

Q5.10 (2 points)

```
1 bool more_even_pragma(uint32_t* array, uint32_t n) {
2     uint32_t even_count = 0;
3     uint32_t odd_count = 0;
4     #pragma omp parallel
5     {
6         uint32_t evens;
7         uint32_t odds;
8         for (uint32_t i = 0; i < n; i++) {
9             if (array[i] % 2 == 0) {
10                evens++;
11            } else {
12                odds++;
13            }
14        }
15        #pragma omp critical
16        {
17            even_count = evens;
18            odd_count = odds;
19        }
20    }
21    return even_count > odd_count;
22 }
```

Why would this be Correct and slower than the naive **more_even**?

Each thread has its own private variable evens but it was not initialized so isn't it contain some garbage value?

♡ ...

 E **Eddy Byun** STAFF 7mth #984ea

Each thread is running the entire for loop; the for loop is not divided evenly among threads (not we don't have a #pragma omp for OR pragma omp parallel for)

♡ ...

 **Anonymous Gaur** 7mth #984eb

I understand that each thread is running the entire for loop, my confusion is the private variable for each thread uint32_t evens; uint32_t odds; did not initialize to 0. From what I remember in C, the declaration of a variable won't set it to default value, instead it contains some garbage value. Why this code would work?

♡ ...

E **Eddy Byun** STAFF 7mth #984ee

↩ Replying to Anonymous Gaur

Ah sorry, apologies. We had a clarification for this (take a look at page 2 of the solutions)

Lines 6 and 7 should say
uint32_t evens = 0;
uint32_t odds = 0;

♡ 1 ...

 **Anonymous Crocodile** 7mth #984ca ✓ Resolved

SP23-Final-Q4

Is this question in scope? What does "thrashing" mean in this context and why do results not cause it?

Q4.2 (2.5 points) What is the overall hit rate for a call to `convolve_1d`? No need to simplify the fraction.

Solution: $\frac{2}{15}$

There are a total of three iterations of the outer loop, and two iterations of the inner loop per iteration of the outer loop. For each iteration of the inner loop, there are two memory accesses: reading `b[j]` and reading `a[i + j]`. Each outer loop has an additional memory access: writing to `results[i]`. In total, there are 5 accesses per iteration of the outer loop, and 15 accesses overall.

For each set of accesses to `a` and `b`, the program experiences thrashing since the index of these blocks conflict. As a result, the accesses to `a` and `b` will always be misses (first two compulsory, then the rest are conflict).

The accesses to `results` do not cause any thrashing, so out of the three accesses (one per iteration of the outer loop), there is one compulsory miss and two hits, giving us a hit rate of $\frac{2}{15}$.

♡ 1 ...



E Eddy Byun STAFF 7mth #984aaf

Yes, this question is in scope.

Thrashing is a term we used when we described a cache where entries would be brought in and evicted back and forth.

The T:I:O breakdown for array `a` is

T: 0b0001 0000 00

I: 0b00

O: 0b0000

The T:I:O breakdown for array `b` is

T: 0b0010 0000 00

I: 0b00

O: 0b0000

The T:I:O breakdown for array `results` is

T: 0b0011 0000 00

I: 0b11

O: 0b0000

Note that for this code, the array accesses to `a` and `b` will go into cache entry at index 0, but `results` is at cache index 3. Accesses to `a` and `b` causes thrashing because cache index 0 will switch between bringing in a block from array `b` and then overwriting that block with a block from `a` and vice versa

♡ 1 ...



Anonymous Llama 7mth #984adf

so when we are loading a[1], do we essentially load a[0] - a[3] 16 bytes all together? similarly with b[0] - b[3].

♡ ...

 **Anonymous Crocodile** 7mth #984bcb

That makes sense, thank you! So if I'm understanding correctly, thrashing is essentially just consecutive conflict misses.

♡ ...

 **Anonymous Barracuda** 7mth #984fae

So why is it for results[0] its a compulsory miss, but results[1] and results[2] it's a hit? Isn't results[0] and results[1] two different places/accessing different parts of the array? Doesn't that mean that it should be a miss each increment?

♡ 1 ...

 **Anonymous Scorpion** 7mth #984bf ✓ Resolved

SP23-Final-Q2

If stage 1 includes IF&ID, isn't ImmGen already in stage 1? Than shouldn't ImmGen also be excluded in the CLK period calculation and the answer to Q2.2 be smt like move MUX?

♡ ...

 **E Eddy Byun** STAFF 7mth #984ed

[#984aa](#)

♡ ...

 **Anonymous Cattle** 7mth #984be ✓ Resolved

SU23-Final-Q4.5

Why all three cases are possible? No explanation in the answer.

Q4.5 (1 point) Supposing that SodaBot splits their memory addresses into the appropriate TIO bits for their cache, which of the following could possibly describe how the hit rate of our code changes after this conversion? You may assume that the program is correctly emulated by SodaBot (e.g. no arithmetic errors occur).

- The hitrate increases
- The hitrate remains the same
- The hitrate decrease

♡ 1 ...

 **Anonymous Wallaby** 7mth #984aea

Yeah unsure too

♡ ...

 **Justin Yokota** STAFF 7mth #984aee

Due to Chinese Remainder Theorem, SodaBot's cache and CoryBot's cache will switch to new block entirely independently of each other regardless of the power of 2/3 used; more formally, for any offset x , we can find a block boundary in SodaBot's cache that is x bytes away from a block boundary in CoryBot's cache. In particular, we can find a block boundary in SodaBot's cache that falls in the middle of a block in CoryBot's cache;

if all our memory accesses are within that block in CoryBot's cache, we'd have a single miss in CoryBot's cache, and two misses in SodaBot's cache, and therefore a lower hitrate for SodaBot. Similarly, we can find a block boundary in CoryBot's cache that falls in the middle of a block in SodaBot's cache, to yield a higher hitrate for SodaBot. We can also find a block of one cache that lies entirely in a block of the other cache, so accessing solely through that block would yield the same hit rate.

♡ 1 ...



Anonymous Crocodile 7mth #984bd

✓ Resolved

SP23-Final-Q3

Do we do $\text{num_rows}/\text{num_threads}$ because matrix is a 2D array? I initially did $(\text{num_rows} * \text{num_cols}) / \text{num_threads}$ because I thought this was a 1D array (similar to what we worked with in project 4).

Parallelize `matrix_average` using OpenMP without using `#pragma omp parallel` for or `reduction`. Each thread should work on one or more rows of the matrix. Assume `num_rows` is a multiple of `num_threads`.

```
1 double matrix_average(double** matrix, int num_rows, int num_cols) {
2     double global_sum = 0.0;

3     _____
4     {
5         int num_threads = omp_get_num_threads();
6         int thread_num = omp_get_thread_num();

7         int chunk_size = _____;
8
9         int start_row = _____;
10
11        int end_row = _____;
12
13        for (int i = start_row; i < end_row; i++) {
14            double row_sum = 0.0;
15            for (int j = 0; j < num_cols; j++) {
16                row_sum += matrix[i][j];
17            }
18
19            _____
20
21        }
22    }
23    return global_sum / (num_rows * num_cols);
24 }
```

Solution:

```
Q3.10: #pragma omp parallel
Q3.11: num_rows / num_threads
Q3.12: thread_num * chunk_size
Q3.13: (thread_num + 1) * chunk_size
Q3.14: #pragma omp critical
Q3.15: global_sum += row_sum
```

♡ ...



E Eddy Byun STAFF 7mth #984fa

Yea, it's because we're working with a 2D array. Note that the argument is `double**`, which tells us that it's a 2D array instead of a 1D array!

♡ 1 ...



Anonymous Moose 7mth #984bc

✓ Resolved

Fall 23 Q3

Saw <https://edstem.org/us/courses/51705/discussion/4854445?comment=11228964>, but still

confused

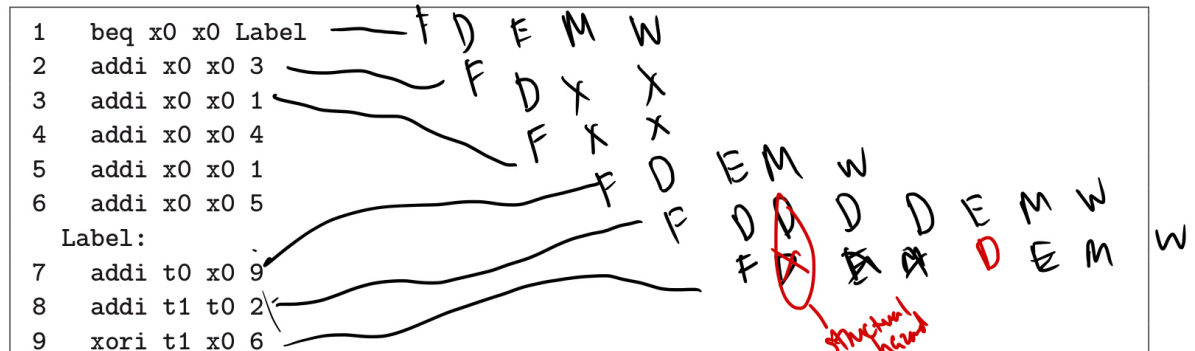
Why would it not look like this?

3 π -pelining

(6 points)

For this problem, assume that we're working with the five-stage pipelined datapath presented on the reference card, and that the CPU will always predict that branches are not taken.

Consider the following RISC-V code:



♡ ...

E Eddy Byun STAFF 7mth #984ef

Between lines 7 and 8, we have a data hazard. On line 7, we are updating `t0` to be 9 and on line 8, we are adding 2 to `t0` and storing that result back into `t1`. As a result, we can only decode the instruction on line 8 when the instruction on line 7 is finished writing back to the `t0` register.

There is no structural hazard between line 8 and 9. We're just overwriting the value in `t1`.

♡ ...

Anonymous Crocodile 7mth #984bb ✓ Resolved

SP23-Final-Q2

I don't understand the explanation for structural hazard: "This is not a structural hazard because it cannot be solved by adding another Regfile (since they won't share data)."

What does Regfile have to do with determining whether a program could experience a structural hazard?

Q2.4 (1.5 points) Assume that the pipeline has been correctly implemented. Which types of hazards could a program experience? Assume that you cannot read from and write to the Regfile in the same clock cycle.

- Control Data Structural None

Solution: Similar to the pipeline implemented in project 3B, there could be control hazards if a branch is taken (the instruction in stage 1 must be flushed). There can also be data hazards if the instruction in stage two writes to the Regfile, and the instruction from stage 1 attempts to read from the same register (since you cannot read and write to the Regfile in the same clock cycle). This is not a structural hazard because it cannot be solved by adding another Regfile (since they won't share data).

Grading: Each checkbox was graded as it's own true/false question, and selecting "None of the above" was treated as not selecting any of the other choices.

♡ ...

E Eddy Byun STAFF 7mth #984fe

The "this" is referring to the fact that the aforementioned data hazard can't be resolved by adding another Regfile.

Structural hazards are usually resolved by putting more hardware into our datapath, and in this case, adding another Regfile isn't going to solve the data hazard since our Regfiles won't share data.

♡ 1 ...

Anonymous Crocodile 7mth #984bca

Thank you for your response - I just want to clarify: would we have a structural hazard if we could read and write to the Regfile in the same clock cycle? I'm trying to think of a case when we would have a valid structural hazard

♡ ...

J Jero Wang STAFF 7mth #984dad

↩ Replying to Anonymous Crocodile

An example of a structural hazard would be needing 2 ALU operations per instruction, because this can be fixed by adding another ALU.

♡ 1 ...

Anonymous Crocodile 7mth #984ba ✓ Resolved

SP23-Final-Q3

Would this be equivalent to: `vec_load(&matrix[i][j])`?

Also why don't we do `matrix[i] + (j*8)`? I thought we would have to multiply j by 8 since a double is 8 bytes.

- **vector** `vec_load(double* A)`: Loads 4 doubles at memory address A into a vector

Q3.4: `vec_load(matrix[i] + j)`

♡ ...

E Eddy Byun STAFF 7mth #984fc

Yep, `vec_load(matrix[i] + j)` is equivalent to `vec_load(&matrix[i][j])`

We don't do `matrix[i] + (j*8)` because of pointer arithmetic!

♡ ...

Anonymous Quail 7mth #984beeb

For Q3.3, would `vec_load(&(matrix[i] + j))` work?

♡ ...

Anonymous Crocodile 7mth #984af ✓ Resolved

SP23-Final-Q1

I'm confused on what the solution means by "switch between passing rs2 and rs3 into the data input port of the DMEM". When would we pass in rs2 vs rs3?

Also, how can we assume that our ALU does not contain multiplication?

Q1.2 (3.5 points) Assume that the changes, if any, for `readArr` have **not** been implemented for this subpart. What changes would we need to make to our datapath in order for us to implement the `writeArr` instruction with as few changes as possible? Select all that apply.

- Add a new immediate type for ImmGen
- Add a new output to Regfile for a third register value
- Add a new input to the AMux and update any relevant selector/control logic
- Add a new input to the BMux and update any relevant selector/control logic
- Add a new ALU operation and update any relevant selector/control logic
- Add a new DMEM mux which feeds into the data input of the DMEM, and any relevant selector/control logic
- Add a new input to WBMux and update any relevant selector/control logic
- None of the above

Solution: The `writeArr` instruction is not similar to any existing RISC-V instructions, so there are a couple of modifications needed. First, similar to `readArr`, it requires the ALU to compute the address at which to store `rs3`, which requires a new ALU operation ($rs1 + rs2 * 4$). Unlike existing S-type instructions, where the data stored in DMEM comes from `rs2`, here, the data comes from `rs3`. As a result, we also need to add a new DMEM mux which can switch between passing `rs2` and `rs3` into the data input port of the DMEM. Finally, `writeArr` is the only instruction that reads three register values, so we need to also add a new output to Regfile.

Grading: Each checkbox was graded as its own true/false question, and selecting "None of the above" was treated as not selecting any of the other choices.

♡ ...

E Eddy Byun STAFF 7mth #984ec

For store instructions in the current datapath, we take the value in `rs2` (whatever value register `rs2` contains) and put it into memory. The new `writeArr` instruction takes the value in `rs3` and puts it into memory. We need a new DMEM mux to decide whether we want to store value in `rs2` or if we want to store the value in `rs3`. We would pass in `rs2` for store instructions and `rs3` for `writeArr` instructions.

Our ALU does have multiplication implemented! However, the new operation we do now is ($rs1 + rs2 * 4$), and this is currently not supported by our ALU

♡ 1 ...

Anonymous Crocodile 7mth #984fb

Thank you! And just to clarify for the ALU: by the "new operation" are we referring to the additional operation? So we're performing two operations instead of one, which is why we need to implement the change.


♡ ...

E Eddy Byun STAFF 7mth #984fd

↩ Replying to Anonymous Crocodile


The "new operation" refers to the fact that we need to do an addition and multiplication in the same cycle ($rs1 + rs2 * 4$). Our datapath can support additions in and multiplications, but we currently can't do a multiplication and addition in the same cycle, which is what is required for the writeArr instruction.

♡ 1 ...

 **Anonymous Penguin** 7mth #984abfd
 ↩ Replying to Eddy Byun


couldn't we do a multiplication and then feed the result back into the ALU to do an addition afterwards? Or would this require more than the minimum number of changes?

♡ ...

 **Jero Wang** STAFF 7mth #984addb
 ↩ Replying to Anonymous Penguin

That would require two cycles, which is not possible with our datapath.

♡ ...

 **Anonymous Llama** 7mth #984e ✓ Resolved

Q3.1 (3 points) If all hazards are resolved through stalling (no double pumping or forwarding paths), during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is a control hazard from line 1 to line 7, as the branch will always be taken. Then, from line 7 to line 8, we have a data hazard from writing to `t0` to accessing it again in the next line. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1. beq x0 x0 Label	IF	ID	EX	MEM	WB									
2. addi x0 x0 3 -> nop		IF	X	X	X	X								
3. addi x0 x0 1 -> nop			IF	X	X	X	X							
4. addi x0 x0 4 -> nop				IF	X	X	X	X						
7. addi t0 x0 9					IF	ID	EX	MEM	WB					
8. addi t1 t0 2 -> nop						IF	X	X	X	X				
8. addi t1 t0 2 -> nop							IF	X	X	X	X			
8. addi t1 t0 2 -> nop								IF	X	X	X	WB		
8. addi t1 t0 2									IF	ID	EX	MEM	WB	
9. xori t1 x0 6										IF	ID	EX	MEM	WB

Fall 23 final #3.1

Assuming no forwarding, I thought `addi t1 t0 2` needs to wait for the `wb` stage of `addi t0 x0 9`. In this diagram, it looks like they both start at cycle. I'm also not sure why the `addi t1 t0 2` instruction can't start one cycle earlier. Since the result of `WB` is at end of cycle 9, can't we have the `EX` stage of `addi t1 t0 2` in cycle 10?

Q3.2 (3 points) If we implement double pumping and all forwarding paths, during which cycle does the `xori` on line 9 execute its WB stage?

For each hazard, write down the hazard, the instructions involved, and the number of stalls required. We will only look at this box if you request a regrade.

Solution: There is still the control hazard from line 1 to line 7, as the branch will always be taken. The data hazard from line 7 to line 8, for register `t0` is resolved via forwarding. This leads to the below timing diagram:

Instruction	1	2	3	4	5	6	7	8	9	10	11
1. <code>beq x0 x0 Label</code>	IF	ID	EX	MEM	WB						
2. <code>addi x0 x0 3 -> nop</code>		IF	X	X	X	X					
3. <code>addi x0 x0 1 -> nop</code>			IF	X	X	X	X				
4. <code>addi x0 x0 4 -> nop</code>				IF	X	X	X	X			
7. <code>addi t0 x0 9</code>					IF	ID	EX	MEM	WB		
8. <code>addi t1 t0 2</code>						IF	ID	EX	MEM	WB	
9. <code>xori t1 x0 6</code>							IF	ID	EX	MEM	WB

Why do we have to wait 3 cycles if we do take the branch? Don't we know at the end of cycle 3 in the EX stage that we do indeed have to take the branch and thus we can begin inst7 at cycle 4?

♡ 1 ...

E Eddy Byun STAFF 7mth #984ab

Regarding 3.2: Note that in our pipelined datapath:

<https://cs61c.org/sp24/pdfs/resources/reference-card.pdf>, we send the ALUOut to the PC after the pipelined register. While we're able to calculate the place we branch to during the EX stage, we actually send ALUOut during the Mem stage

♡ ...

E Eddy Byun STAFF 7mth #984ac

For 3.1: We can still fetch the `addi t1 t0 2` instruction while `addi t0 x0 9` is writing back to the regfile. Remember all the IF stage does is fetch the instruction from IMEM. In the decode stage is where we get the value of `t0`, and by the time the `addi t0 t0 2` instruction reads the value in `t0`, we would've already finished writing back to the reg file!

♡ ...

Anonymous Sardine 7mth #984cad

I'm confused about for 3.1, how do we know to no-op/X out instructions before the first instruction completes the Mem stage? From my understanding, we only know whether a branch is taken after the Memory stage, so for lines following the first line how do we know that they will no-op before we reach column 4? Also, why do we include IF instead of also having IF be an X in the diagram?

♡ 1 ...

Anonymous Spoonbill 7mth #984d

✓ Resolved

[Spring 2023] Q 7

For the solution when we set `data[i] = codecopy` are we not setting `data[i]` to the entire array starting at what `codecopy` points to or just the one element that `codecopy` is pointing to?

```

8   for(int i = 0; _____; i++) {
                                Q7.2
9       num += _____;
                                Q7.3
10  }
11  int** data = malloc(_____);
                                Q7.4
12  int* codecopy = calloc(n+1, _____);
                                Q7.5
13  // Hint: You should not need any more memory allocations
14  memcpy(codecopy, code, _____);
                                Q7.6
15  for(int i = 0; _____; i++) {
                                Q7.7
16      data[i] = _____;
                                Q7.8
17      while(_____ && _____ != 0) {
                                Q7.9                                Q7.10
18          _____;
                                Q7.11
19      }
20      _____;
                                Q7.12
21      codecopy++;
22  }
23  _____;
                                Q7.13
24  return _____;
                                Q7.14
25 }

```

Q7 Cumulative: RV32tok

(14 points)

Write a program `splitCode`, which will split a RISC-V program into blocks of code with no branches or jumps (`jal` or `jalr`). Specifically, `splitCode` will have the following function signature:

- **Input:** `int* code`, an array of RISC-V instructions. Each RISC-V instruction is stored as a 32-bit integer, equal to its translation. You may assume that all instructions are valid RISC-V base instructions, and that there are no pseudoinstructions, `ecalls`, or `ebreaks`.
- **Input:** `n`, the number of instructions in `code`.
- **Input:** `int*** result`, a pointer to store your result. Your result should be an array of `int*s`, where each `int*` points to the beginning of a sequence of consecutive instructions with no branches or jumps. Each of these arrays should be "null-terminated"; that is, the last element of each array should be the number 0, to signify the end of the array. Every non-branch/non-jump instruction must be represented in exactly one subarray of your result. No branch/jump instruction should be in any subarray of your result.
- **Output:** `int`, the length of your result.

For example, for the following RISC-V code:

♡ ...

M Myrah Shah STAFF 7mth #984f

Remember that in C, a pointer to an array is actually a pointer to the first element of the array! So by using `codecopy`, we are getting the item currently pointed to by `codecopy` (which is why we're able to get the next one by just incrementing `codecopy`).

♡ ...



Anonymous Spoonbill 7mth #984b

✓ Resolved

[Spring 2023]

I was a bit confused on this question as it's including the immediate generator in stage 2. However, I thought the immediate generator is part of the IF/ID stage (not execute)? Could someone explain what the main steps are in IF/ID stage and why immediate generator is during the execute stage instead?

Q2 IF Only ID Pipelined Better

(10 points)

In Project 3, we implemented a RISC-V CPU with two stages; stage 1 included IF and stage 2 included ID/EX/MEM/WB. For this question, imagine instead that we implement a two-stage pipeline with a different split; stage 1 will include IF/ID and stage 2 will include EX/MEM/WB (IF/ID/EX/MEM/WB are defined equivalently to the pipelined CPU on the reference card).

For Q2.1 and Q2.2, assume the following delays for each component. Any component not listed is assumed to have a negligible delay.

Component	Delay
$\tau_{\text{clk-to-q}}$	35ps
τ_{setup}	20ps
Mux	75ps
Regfile Setup	20ps
Regfile Read	175ps
Immediate Generator	150ps
Branch Comparator	200ps
ALU	200ps
Memory Read	300ps

Q2.1 (3 points) What is the minimum clock period of this circuit, in picoseconds, to achieve correct behavior?

Solution: $855\text{ps} (\tau_{\text{clk-to-q}} (35) + \text{Immediate Generator} (150) + \text{Mux} (75) + \text{ALU} (200) + \text{Memory Read} (300) + \text{Mux} (75) + \tau_{\text{setup}} (20))$.

The critical path occurs in stage 2 for a load instruction.

Grading: Partial credit was given for errors that showed conceptual understanding of what the critical path is, but excluded or included an extra component's timing. We did not give partial credit for excluding some components since we cannot clearly distinguish between a conceptual misunderstanding or a mechanical error.

♡ 2 ...



E Eddy Byun STAFF 7mth #984aa

The Immediate Generator was considered to be part of the EX stage in Spring 2023: <https://inst.eecs.berkeley.edu/~cs61c/sp23/pdfs/resources/reference-card.pdf> (Spring 2023 ref card). This semester, the Imm Gen is considered to be part of the ID stage.

♡ 1 ...



Anonymous Quetzal 7mth #984aad

Any technical reason for the switch? Or is it arbitrary?

♡ ...



E Eddy Byun STAFF 7mth #984aba

↩ Replying to Anonymous Quetzal

I think the idea was that if ImmGen is in the ID stage, you can run the RegFile read and Immediate generation in parallel AND the branch comparator and ALU in parallel. If the ImmGen is in the EX stage, the ALU has to wait for the Immediate Generator to finish (for any instructions that require an immediate) before it can get going.

♡ 1 ...



Anonymous Spoonbill 7mth #984abc

↩ Replying to Eddy Byun

Got it so to clarify, in IF we have the PC+4 calculation, PC goes through IMEM to fetch instruction. In ID regfile reads instructions, and control signals set, in execute we go through everything until DMEM, and then DMEM is memory and finally the writeback is done. Is this the right order of operations and stages?

Also for clock period calculations, do we only consider the regfile, DMEM and PC to be clocked elements (where you can start / stop for critical path calcs)? Also outside of loading and storing instructions, do you ever go through the DMEM (or instead directly go from ALU output to WBSel mux) or does every instruction technically go through the DMEM?

♡ 2 ...



Anonymous Spoonbill 7mth #984a

✓ Resolved

Hi! For spring 2023, are we suppose to know how to do q 6 (is it in scope)? And if so, which lecture was this content covered in?

Q6 Cumulative: Potpourri

(5 points)

Q6.1 (1 point) The OS running on a cluster of computers in a datacenter allows a single machine to read and write the memory and local disk of a remote machine in the same rack or array. According to lecture, which of the following are true? Select all that apply. ("farther away" means that the distance the data travels increases in steps, first to our local machine, then to a machine in our same rack, then to a machine in our same array.)

- As our CPU sends data to DRAM farther away, bandwidth increases
- As our CPU sends data to Disk farther away, bandwidth increases
- As our CPU sends data to DRAM farther away, latency increases
- As our CPU sends data to Disk farther away, latency increases
- We have higher latency to DRAM on an Array computer than to our own disk
- We have higher bandwidth to DRAM on an Array computer than to our own disk
- None of the above

Q6.2 (1 point) After a single machine M finishes its assigned portion of a map task in a MapReduce cluster, which of the following can happen immediately, regardless of the overall program state? Select all that apply.

- M can shut down without risking the success of the overall computation
- M can be assigned a new map task
- Data shuffling of the workload M just finished can begin
- The reduce task for the workload M just finished can begin
- None of the above

Q6.3 (1 point) We want to send **one** bit using a Hamming error correcting code. What are the valid bit patterns you could send that correspond to 0b0 and 0b1?

0b0: 0b1:

♡ 1 ...

 **Anonymous Spoonbill** 7mth #984c

I think this is from the optional data centers lecture but just want to ensure that it is out of scope this year.

♡ ...

 **E Eddy Byun** STAFF 7mth #984ad

Q6 is out of scope!

♡ 1 ...

 *This comment was deleted*

 **T Timothy Bergal** 7mth #984adae

This looks like a cs170 question

♡ ...

W

Wil Rothman 7mth #984adbb

And I realized I posted this in the CS61C thread not the 170 thread. Sorry!

♡1 ...