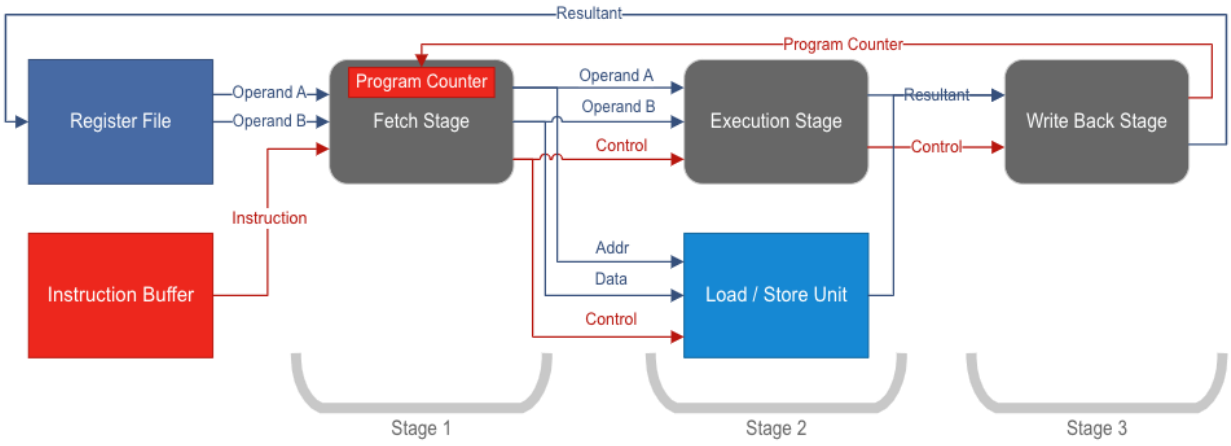
Performance Modeling Engineer Technical Screen

Welcome to Interview Inc. We're very excited to have you on board, and we need you to get to work right away. A new critical workload, ChipMark3024, has just been released and we're about to do a design freeze on our new processor, codename Kindred! We need you to develop a simulator that can evaluate the performance of Kindred running ChipMark3024 (and a handful of other workloads). *Please keep in mind that you do not need to implement a full functional model, you only need a simulation that can report the proper runtime of the given workloads.*

Processor Architecture

Kindred is a very simple 3 stage processor. It has a single hart in a single core. It has 3 pipelined stages: Fetch, Execution, and Write-Back. Workloads are kept resident in a read-only Instruction Buffer accessible by the Fetch stage. The Fetch and Write-Back stages have access to a Register File with 2 read ports and 1 write port. The chip has full support for the RISC-V RV32I instruction set.

Kindred Pipeline Diagram



The Fetch stage fetches an instruction from the Instruction Buffer, decodes it, and then fetches 2 associated operands from the Register File. The Execution stage completes a single instruction each cycle. The Write-Back stage can write a single operand to the Register File each cycle. There is no forwarding path for Resultants before Write-Back, meaning that operations using dependent operands may need to stall in the Fetch stage for up to 2 cycles before Write-Back has completed. All ops, including NOPs, propagate through all 3 stages.

Stores are sent to the Load/Store Unit (LSU) in the Execution stage, requiring 1 additional cycle to complete. Loads are sent to the LSU in the Execution stage, also requiring 1 additional cycle to complete and return, allowing data to be written to the register file in the Write-Back stage without stalling. In this system there is no hazard or stall required between load and store operations of the same address, however stalls between pipeline stages to allow for Write-Back are still inserted.

Control transfer instructions that modify the PC, such as jumps or branches that are taken, will cause stalls in the Fetch and Execute stages until the PC has been updated in the Write-Back stage, effectively inserting a 2 stage stall.

Workloads

Workloads are provided in RISC-V assembly. The expected runtime for Synthetic A is provided to help you test your model. *Please keep in mind that you only need to implement what is required to report the proper runtime performance of these workloads*.

Synthetic A

```
synth_a:   li   t0, 0
loop:      bge  t0, 1, end
           addi t0, t0, 1
           jal  x0, loop
end:       nop
```

Timing:

Cycle	Fetch	Execute	Write-Back
0	li		
1	X	li	
2	X	X	li
3	bge		
4	addi	bge	
5	jal	addi	bge
6	X	jal	addi
7	X	X	jal
8	bge		
9	X	bge	X
10	X	X	bge
11	nop		
12	X	nop	
13	X	X	nop

Expected runtime: 14 cycles

Synthetic B

```
      li    s0, 0xA000
      li    t0, 0
loop: bge   t0, 8, end
      slli  t2, t0, 2
      add   t3, s0, t2
      lw    t4, 0(t3)
      sub   t4, x0, t4
      sw    t4, 0(t3)
      addi  t0, t0, 1
      jal   x0, loop
end:  nop
```

ChipMark 3024

```
                li    s0, 0xA000
                li    t0, 0
outer_loop:     bge   t0, 3, end
                li    t1, 0
inner_loop:     slri  t2, t0, 2
                add   t3, s0, t2
                lw    t4, 0(t3)
                addi  t4, t4, 100
                sw    t4, 0(t3)
                addi  t1, t1, 1
                ble   t1, 9, inner_loop
                addi  t0, t0, 1
                jal   x0, outer_loop
end:            nop
```

Logistics:

You are more than welcome to program this model in whatever language you are most comfortable. Particularly, C++ (with or without SystemC) and/or Python would be good to show off. The assembly for the workloads is given above. Feel free to manually or programmatically manipulate these to best run on your model. If you want to write the whole model in Python, including assembly string parsing, go for it. If you want to write a C++ model that takes an instruction stream format that you define and hand translate, go for it. I just want to see that you have a simulator capable of calculating performance for the 3 instruction streams provided. Bonus points for any dumping, tracing, visualization, or other instrumentation that helps show extra details of what is happening inside the model.

This should only take a few hours. Keep in mind that you only need to simulate the ops in the provided workloads and that you don't need to model any more functionality than is required for proper control flow. Notice that only `t0` and `t1` are used for control flow in the 3 workloads.

If you have any questions please feel free to send me an email, text, whatsapp, or call. My email is creynia@d-matrix.ai, and my phone number is +1.386.479.7799. I'll be happy to chat with you or hop on a Teams meeting if anything isn't making sense.

When submitting your work, please send me an email to creynia@d-matrix.ai with the calculated runtime values for Synthetic B and ChipMark3024 as well as brief instructions on how to download, build, and run your code. In order to avoid file size issues, please don't send me the code directly. Upload it to github, dropbox, google drive, or some other service and then send me a link. If you're comfortable using github and posting your code publicly, that would be the preferred method.

Have fun and I look forward to seeing your work!