## Problem 1:

In this exercise, we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word: 0x00c6ba23.

0x00c6ba23 translates to: 00000001100011010111010000100011
opcode: 0100011
funct3: 011
=> instruction: sd
rs1, rs2: 01101, 01100 => 13, 12
immediate: 000000010100 => sw x12, (20)x13

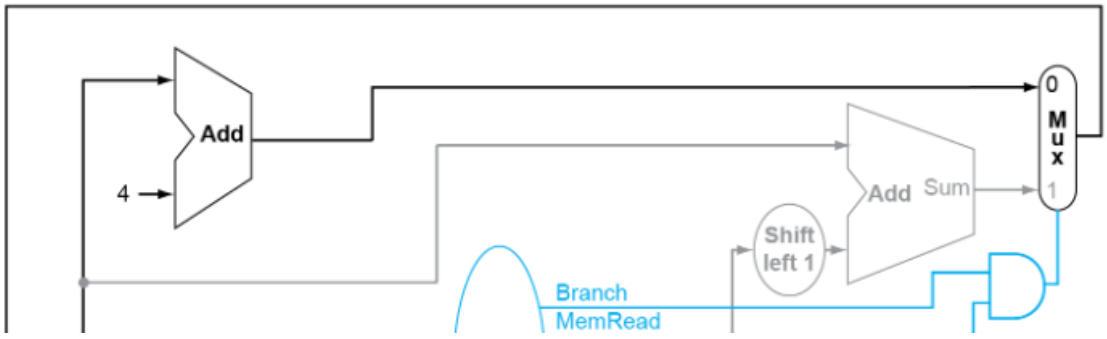
First 25 pages of the slides to figure out the answers

**1.1** What are the values of the ALU control unit's inputs for this instruction?

- Since it is a load/store type instruction, the value for the ALU control unit's input is 0010 and ALUop is 00

**1.2** What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

- The new PC address will be PC + 4. Since it is not a branch type instruction.




**1.3** For each mux, show the values of its inputs and outputs during the execution of this instruction. List values that are register outputs at Reg [xn].

- For the mux responsible for address manipulation for the next instruction, the two inputs are the current instruction address + 4 and 0x000000000000000A since the instruction is not a branch type instruction.
- For the mux after the instruction decoding, it will have the second register and the sign extended offset as inputs.

**1.4** What are the input values for the ALU and the two add units?

- ALU inputs: Reg[x13] and 0x0000000000000014 (offset, sign extended)
- PC + 4 adder inputs: PC and 4
- Branch adder inputs: PC and 0x0000000000000028

**1.5** What are the values of all inputs for the register's unit?

## Problem 2:

Problems in this exercise assume that the logic blocks used to implement a processor's datapath
have the following latencies:

| I-Mem / D-Mem | Register File | Mux | ALU | Adder | Single gate | Register Read | Register Setup | Sign extend | Control |
|---|---|---|---|---|---|---|---|---|---|
| 250 ps | 150 ps | 25 ps | 200 ps | 150 ps | 5 ps | 30 ps | 20 ps | 50 ps | 50 ps |

"Register read" is the time needed after the rising clock edge for the new register value to appear on
the output. This value applies to the PC only. "Register setup" is the amount of time a register's data
input must be stable before the rising edge of the clock. This value applies to both the PC and Register File.

**2.1** What is the latency of an R-type instruction (i.e., how long must the clock period be to ensure that this instruction works correctly)?

- R-type : 30 + 250 + 150 + 25 + 200 + 25 + 20 = 700ps

**2.2** What is the latency of ld? (Check your answer carefully. Many students place extra muxes on the critical path.)

- ld : 30 + 250 + 150 + 25 + 200 + 250 + 25 + 20 = 950ps

**2.3** What is the latency of sd? (Check your answer carefully. Many students place extra muxes on the critical path.)

- sd : 30 + 250 + 150 + 200 + 25 + 250 = 905ps

**2.4** What is the latency of beq?

- beq : 30 + 250 + 150 + 25 + 200 + 5 + 25 + 20 = 705ps

**2.5** What is the latency of an I-type instruction?

- I-type : 30 + 250 + 150 + 25 + 200 + 25 + 20 = 700ps

**2.6** What is the minimum clock period for this CPU?

- 950ps

**Problem 3:**

**(a)** Suppose you could build a CPU where the clock cycle time was different for each instruction.

| R-type/I-type (non-ld) | ld | sd | beq |
|---|---|---|---|
| 52% | 25% | 11% | 12% |

**a1** What would the speedup of this new CPU be over the CPU presented in Figure 4.21 (in RISC-V text) given the instruction mix below? (assuming instruction latencies from the
problem 2

- Using the latencies calculated in problem 2, the average time for our new sped up instruction set would be (0.52 * 700) + (0.25 * 950) + (0.11 * 905) + (0.12 * 705) = 785.65 ps
- Speedup = 925/785.65 = 1.177 => ~ 18% speedup

**(b)** Consider the addition of a multiplier to the CPU shown in Figure 4.21. This addition will add 300 ps to the latency of the ALU, but will reduce the number of instructions by 5% (because there will no longer be a need to emulate the multiply instruction).
**b1** What is the clock cycle time with and without this improvement?
**b2** What is the speedup achieved by adding this improvement?
**b3** What is the slowest the new ALU can be and still result in improved performance?

**(c)** When processor designers consider a possible improvement to the processor datapath, the
decision usually depends on the cost/performance trade-off. In the following three problems,
assume that we are beginning with the datapath from Figure 4.21, the latencies from Problem 2 in this assignment, and the following costs:

| I-Mem | Register File | Mux | ALU | Adder | D-Mem | Single Register | Sign extend | Single gate | Control |
|---|---|---|---|---|---|---|---|---|---|
| 1000 | 200 | 10 | 100 | 30 | 2000 | 5 | 100 | 1 | 500 |

Suppose doubling the number of general-purpose registers from 32 to 64 would reduce the number of ld and sd instruction by 12%, but increase the latency of the register file from 150 ps to 160 ps and double the cost from 200 to 400. (Use the instruction mix [from 3(a) above] and ignore the other effects on the ISA)

**c1** What is the speedup achieved by adding this improvement

- Speedup achieved = 950 * (number of instructions) / 1250 * 0.95 * (number of instructions)
- = 0.8 => There is no real speed up, this will make the CPU slower

**c2** Compare the change in performance to the change in cost.

- Register File cost is now 400
- 12% reduction in load stores => (0.25 + 0.11) * 0.12 => 4.32% reduction in number of instructions.
- As compared to 950*n execution time, we now need 960 * 0.9568n execution time.
- => speedup = 950/ (960 * 0.9568) = ~ 1.0342 => 3.42 % speedup
- The relative increase in cost = sum of new costs / sum of new costs = 4707/4507 => 4.4% increase in cost.

**c3** Given the cost/performance ratios you just calculated, describe a situation where it makes sense to add more registers and describe a situation where it doesn't make sense to add more registers.

- From what we know, a 4.4% cost increase results in a 3.42% increase in performance.
- We don't even know whether the increase in cost would scale linearly. Even if we assume it does, the return on the increase in the cost by percentage is more than the% increase in performance.
- If let's say a deadline is being missed by a few hours then this scaling may be worth looking into.

## Problem 4:

ld is the instruction with the longest latency on the CPU from Section 4.4 (in RISC-V text).
If we modified ld and sd so that there was no offset (i.e., the address to be loaded from/stored to must be calculated and placed in rs1 before calling ld/sd), then no instruction would use both the ALU and Data memory. This would allow us to reduce the clock cycle time. However, it would also increase the number of instructions, because many ld and sd instructions would need to be replaced with ld/add or sd/add combinations.

**4.1** What would the new clock cycle time be?

- The new clock cycle would eliminate the need for the ALU i.e the cost would be 750ps now.

**4.2** Would a program with the instruction mix presented in Problem 2 run faster or slower on this new CPU? By how much? (For simplicity, assume every ld and sd instruction is replaced with a sequence of two instructions.)

- Doublin of the ld/sd instructions means that the total number of instructions is now 1.36 * n.
- The new clock time is 750ps => the speedup would be 950 * n / (750 * 1.36 * n)
- = 0.931 => there is no speedup, the machine has slowed down owing to the change in clock time and increase in instructions.

**4.3** What is the primary factor that influences whether a program will run faster or slower on the new CPU?

- In the new CPU, the number of loads and stores dominates how the CPU will perform.

**4.4** Do you consider the original CPU (as shown in Figure 4.21 of RISC-V text) a better overall design; or do you consider the new CPU a better overall design? Why?

- That is a highly subjective question. The newer CPU will do well when the number of load/stores are low and the older CPU will do better with high load stores is my guess.

## Problem 5:

**(a)** Examine the difficulty of adding a proposed lwi.d rd, rs1, rs2 ("Load With Increment") instruction to RISC-V. Interpretation: Reg[rd]=Mem[Reg[rs1]+Reg[rs2]]

**5.a1** Which new functional blocks (if any) do we need for this instruction?

- We would need to add a new muxes to the datapaths for this implementation.

**5.a2** Which existing functional blocks (if any) require modification?

- No functional blocks would require modification, just addition of data paths and muxes would be enough

**5.a3** Which new data paths (if any) do we need for this instruction?

- We would need a path from the ALU output directly to the data memory and a path from read data to the data memory's address input.

## Problem 6:

In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

| IF | ID | EX | MEM | WB |
|---|---|---|---|---|
| 250 ps | 350 ps | 150 ps | 300 ps | 200 ps |

Also, assume that instructions executed by the processor are broken down as follows:

| ALU/Logic | Jump/Branch | Load | Store |
|---|---|---|---|
| 45% | 20% | 20% | 15% |

**6.1** What is the clock cycle time in a pipelined and non-pipelined processor?

- Non pipelined would be the sum of all the stages = 1250ps
- Pipelined would be the longest stage i.e ID stage = 350ps

**6.2** What is the total latency of an ld instruction in a pipelined and non-pipelined processor?

- ld instruction would make use of all the stages so the latency would be the same for pipelined and non pipelined processor = 1250ps

**6.3** If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

- ID being the lengthiest in terms of time, splitting that would reduce the clock cycle time the most.
-

**6.4** Assuming there are no stalls or hazards, what is the utilization of the data memory?

- Load % + Store % = 35%

**6.5** Assuming there are no stalls or hazards, what is the utilization of the write-register port of the “Registers” unit?

- ALU/Logic % + Jump/Branch % = 65%

## Problem 7:

What is the minimum number of cycles needed to completely execute n instructions on a CPU with a k stage pipeline? Justify your formula.

- For a k stage pipeline, the first instruction doesn’t enter the last stage i.e the WB stage will k cycles are done.
- After that the rest of the n-1 instructions will enter the WB stage one by one.
- Which gives us a minimum of k + n - 1 cycles

## Problem 8:

(a) Assume that x11 is initialized to 11 and x12 is initialized to 22. Suppose you executed the code below on a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of registers x13 and x14 be?

```
addi x11, x12, 5
add x13, x11, x12
addi x14, x11, 15
```

- Since data hazards are not handled, x13 will contain 33 and x14 will contain 26

(b) Assume that x11 is initialized to 11 and x12 is initialized to 22. Suppose you executed the

code below on a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of register x15 be? Assume the register file is written at the beginning of the cycle and read at the end of a cycle. Therefore, an ID stage will return the results of a WB state occurring during the same cycle. See Section 4.7 and Figure 4.51 for details.
addix11, x12, 5
addx13, x11, x12
addix14, x11, 15
addx15, x11, x11

- x15 will contain 54

(c) Add NOP instructions to the code below so that it will run correctly on a pipeline that does not
handle data hazards.

addix11, x12, 5
addx13, x11, x12
addix14, x11, 15
addx15, x13, x12

- addix11, x12, 5
- NOP
- NOP
- addx13, x11, x12
- addix14, x11, 15
- NOP
- addx15, x13, x12