NYU Tandon School of Engineering

Fall 2022, ECE 6913

Homework Assignment 5

Instructor: Azeez Bhavnagarwala, email: ajb20@nyu.edu

Course Assistants

Varadraj Kakodkar (vns2008), Kartikay Kaushik (kk4332), Siddhanth Iyer (si2152), Swarnashri Chandrashekar (sc8781), Karan Sheth (kk4332), Haotian Zheng (hz2687), Haoren Zhang (kk4332), Varun Kumar (vs2411)

Homework Assignment 4 [released Wednesday October 23rd 2022] [due Wednesday November 2nd by 11:59PM]

You *are allowed* to discuss HW assignments with anyone. You are *not allowed* to share your solutions with other colleagues in the class. Please feel free to reach out to the Course Assistants or the Instructor during office hours or by appointment if you need any help with the HW.
Please enter your responses in this Word document after you download it from NYU Classes.
Please use the Brightspace portal to upload your completed HW.

.

Supportive tables:

(1) Table 1 -- ALU control unit

Name (Bit position)	Fields 31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

ALUOp		Funct7 field							Funct3 field			
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

(2) Table 2 -- Control lines

Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

(3)

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`**.

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

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

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]`.

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

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

Solution

1.1

The orginal instruction word 0x00c6ba23.

The "opcode" is 010 0011, which means it's a S-format instruction.

Further check the "funtc3" is 011. Thus we can know this instruction is "sd".

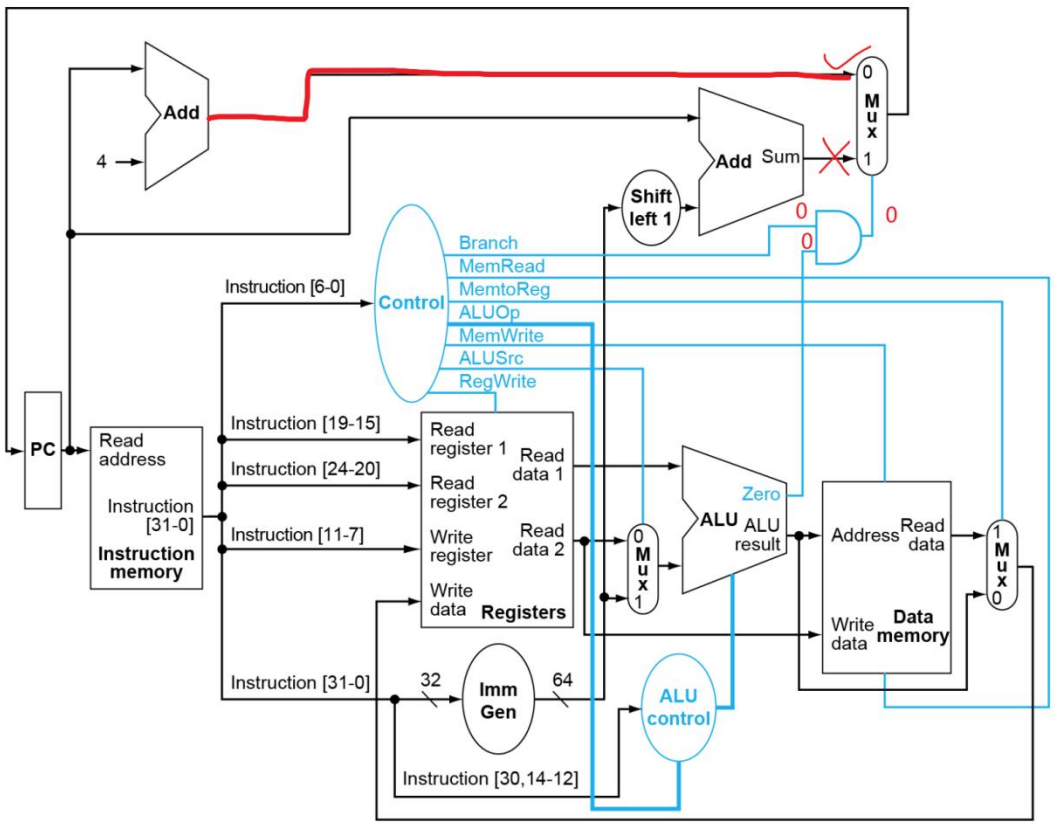
"sd" instruction is related to the ALU opcode "00".

Therefore, the corresponding ALU control unit is "0010" according to supportive table 1.

1.2

According to supportive table 2, "sd" instruction has "Branch" signal of 0.

Thus, the "PCSrc" signal is also 0. The new PC address is PC + 4.



As shown in this picture, the mux at the top right corner will select the value “PC + 4” as the new PC address.

1.3

Mux	Control input	input 1	input 2	output
ALUsrc	1	rs1	0x014	0x014
PCsrc	0	PC + 4	0x4	PC + 4
MemtoReg	δ	rs1 + 0x014	unknown	unknown

1.4
Inputs of ALU are: 0x014, rs1
Inputs of PC adder are: 4, PC
Inputs of Branch adder are: PC, 0x028

1.5
Read Register 1: x13, 01101
Read Register 2: x12, 01100
Write Register: None
Write Data: None

The last 2 inputs are none because the “RegWrite” signal now is set to “0”.

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)?
2.2 What is the latency of `ld`? (Check your answer carefully. Many students place extra muxes on the critical path.)
2.3 What is the latency of `sd`? (Check your answer carefully. Many students place extra muxes on the critical path.)
2.4 What is the latency of `beq`?
2.5 What is the latency of an I-type instruction?
2.6 What is the minimum clock period for this CPU?

Solution:
2.1
For R-Type instruction:

PC regsiter read delay following rising edge of clock: 30ps
Instruction Memory: 250ps
Read register file: 150ps
Mux to pick data from R2 or sign extended data: 25ps
ALU delay: 200ps
Mux for WB to register file of result from ALU: 25ps
Write back setup time for register file registers: 20ps

The total latency is: 700ps

2.2
For "ld" instruction:

PC regsiter read delay following rising edge of clock: 30ps
Instruction Memory: 250ps
Read register file: 150ps
Mux to pick data from R2 or sign extended data: 25ps
ALU delay: 200ps
Read data from data memory, whose address provided by ALU: 250ps
Mux for WB to register file of result from data memory: 25ps
Write back setup time for register file registers: 20ps

The total latency is: 950ps

2.3
For “sd” instruction:

PC regsiter read delay following rising edge of clock: 30ps
Instruction Memory: 250ps
Read register file: 150ps
Mux to pick data from R2 or sign extended data: 25ps
ALU delay: 200ps
Write data from R2 to data memory: 250ps

The total latency is: 905ps

2.4
For “beq” instruction:

PC regsiter read delay following rising edge of clock: 30ps
Instruction Memory: 250ps
Read register file: 150ps
Mux to pick data from R2 or sign extended data: 25ps
ALU delay: 200ps
Single gate delay AND of “zero” output from ALU and “Branch” control output from Control unit: 5ps
Mux for branch address: 20ps

The total latency is: 705ps

2.5
For I-Type instruction:

PC regsiter read delay following rising edge of clock: 30ps
Instruction Memory: 250ps
Read register file: 150ps
Mux to pick data from R2 or sign extended data: 25ps
ALU delay: 200ps
Mux for WB to register file of result from data memory: 25ps
Write back setup time for register file registers: 20ps

The total latency is: 700ps

2.6
The minimum clock period = The longest latency = 950ps

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

3.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)

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

New cycle time is:

700ps * 0.52 + 950ps * 0.25 + 905ps * 0.11 + 705ps * 0.12 = 785.6ps

Speed up is: 950ps / 785.6ps = 1.209

3 (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).

3.b1 What is the clock cycle time with and without this improvement?

The clock cycle time with this improvement: 1250ps

The clock cycle time without this improvement: 950ps

3.b2 What is the speedup achieved by adding this improvement?

The speedup is:

950ps /((1 - 5%) * 1250ps) = 0.8

Thus, after applying this improvement the CPU is slower than before.

3.b3 What is the slowest the new ALU can be and still result in improved performance?
Based on the new ALU architecture, it is the slowest state right now. If we cannot further reduce the number of instructions, there won't be any improvements.

3 (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*)

3.c1 What is the speedup achieved by adding this improvement?
The original latency of "ld" instruction is: 950ps
The original latency of "sd" instruction is: 905ps

The new latency of "ld" instruction is: 960ps
The new latency of "sd" instruction is: 915ps

If we focus on the cycle time which equals to the longest latency time. The speed up is:
950ps / ((1 - 12%) * 960ps) = 1.125

3.c2 Compare the change in performance to the change in cost.
According to last problem, the performance is 12.5% faster than original.

For "ld" instruction, the original cost is:
1000 + 200 + 10 * 2 + 100 + 2000 + 5 * 2 = 3330
New cost is:
1000 + 400 + 10 * 2 + 100 + 2000 + 5 * 2 = 3530
The change in cost:
3530 / 3330 = 1.06

For "ld" instruction, the original cost is:
1000 + 2000 + 200 + 10 + 100 + 5 = 3315
New cost is:
1000 + 2000 + 400 + 10 + 100 + 5 = 3515
The change in cost:
3515 / 3315 = 1.06

Therefore, we reach a 12.5% performance improvement with a 6% increase in cost.

3.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.

It is a very comprehensive question. If the financial cost of adding more registers is afforadable or there is a very strict requirement of the performance, then we need to add more registers. Otherwise we can keep the current registers.

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 time for "ld" instruction is:
PC regsiter read delay following rising edge of clock: 30ps
Instruction Memory: 250ps
Read register file: 150ps
Read data from data memory, whose address provided by ALU: 250ps
Mux for WB to register file of result from data memory: 25ps
Write back setup time for register file registers: 20ps

The total latency is: 725ps

The new clock cycle time for "sd" instruction is:
PC regsiter read delay following rising edge of clock: 30ps
Instruction Memory: 250ps
Read register file: 150ps
Write data from R2 to data memory: 250ps

The total latency is: 680ps

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.)

The speed up of "ld" instruction is:
950 / (725 + 700) = 0.6667

The speed up of "sd" instruction is:
905 / (680 + 700) = 0.6558

So both "ld" and "sd" instruction are slower on this new CPU.

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

The primary factor influences the performance of a program is the number of "ld" and "sd" instructions since these 2 kind of instructions takes longer time to execute than other instructions.

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?

I think it depends on different situations. If a program contains large amount of "ld" and "sd" instructions, then we may choose this new design.

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?

To achieve this function, we do not need a new block.

This is because the execution process after getting the ALU result is the same. We only need to modify the data write back process, which is to fetch the data using the calculated address and the write back to register.

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

The control unit. More specifically, we need to enable the

"MemRead" and "RegWrite" signal.

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

No extra data paths is need for this instruction.

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?

For a non-pipelined processor:

Clock cycle time = 1250ps

For a pipelined processor:

Clock cycle time = 350ps

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

For a non-pipelined processor:

The total latency of an “ld” instruction = 1250ps

For a pipelined processor:

The total latency of an “ld” instruction = 350ps

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?

I would split the “ID” stage because it has the longest latency.
After split that, the longest latency is 300ps.

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

The utilization of data memory contains the “Load” and “Store” instructions. Thus the utilization is 20% + 15% = 35%

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

The "ALU" and "Load" instructions will write data to the write-register port. Thus, the utilization of this port is 45% + 20% = 65%.

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.

Solution:

k + n - 1

If all instructions can be pipelined perfectly, the last instruction will cost k cycles and the other n - 1 instructions cost 1 cycles. Thus the minimum number of cycles is k + n - 1.

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,x11,12
addi x14,x11,15
```

Solution:

x13 = 33

x14 = 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.

```
addi x11, x12, 5
add x13, x11,x12
addi x14, x11,15
add x15, x11,x11
```

Solution:

54

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

```
addi x11, x12, 5
```

```
add x13, x11,x12
addi x14, x11,15
add x15, x13,x12
```

Solution:

```
addi x11, x12, 5
NOP
NOP
add x13, x11,x12
addi x14, x11,15
NOP
add x15, x13,x12
```