NYU Tandon School of Engineering

Fall 2023, ECE 6913

Homework Assignment 4

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

Course Assistant Office Hour Schedule

On Zoom: 9:30AM – 11AM Monday, Tuesday, Wednesday & Thursday

On Zoom: 4PM – 6PM Monday (Focus on Project A)

1. Arushi Arora, aa10350@nyu.edu
2. Prashanth Rebala, pr2359@nyu.edu
3. Moin Khan, mk8793@nyu.edu
4. Raj Ghodasara, g4357@nyu.edu

Homework Assignment 4 [released Saturday October 21st 2023] [due Sunday October 29th 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.*
.

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?

00c6ba2316 = 0000 0000 1100 0110 1011 1010 0010 0011
Since the 'add' operation is executed in the ALU for a sd instruction (add 2s complement offset in immediate 12 bit field (20_{10}) to base address (in rs1), the ALU Control lines take the value 0010.

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

Since the sd instruction does not take any branch, the next PC address after this instruction is executed is PC+4.

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]`.

ALUsrc is '1' and takes the input to this mux from the sign extend unit (0x0..014).
PCsrc is '0' since this mux is asserted only for branch instructions.
MemtoReg is value is irrelevant (don't care). It is asserted '1' this choice picks data from Data Memory to send to the register file for Write back. Data Memory is in Write mode so the data at output buffer of Data memory is undefined. This data is routed to write port of Register file array.

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

ALU inputs are rs1 (Reg[x13]) and 0x00..014 from the mux controlled by ALUsrc Branch adder inputs: PC and 0x0..014 shifted by 1 (0x0..28) PC adder inputs: PC and 4.

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

Read register 1 (rs1) input port has the 5-bit instruction field [19-15] identifying the register x13. Read register 2 (rs2) input port has the 5-bit instruction field [24 – 20] identifying the register x12. Write register input port has undefined/irrelevant bits since Reg Write control signal is disabled for the sd instruction.

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)?

1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps
[note - there is no number given for sign extension delay, so we can assume it is less than the register file delay of 150 ps]
5. ALU delay: 200 ps
6. Mux for WB to Register File of result from ALU: 25 ps
7. Write back setup time for Register File registers: 20 ps
total = 700ps

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

1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps
[note - number given for sign extension delay of 50 ps is less than the 150 ps for register file read access, so we can assume that mux control signal is designed to fire only when both inputs are ready]
5. ALU delay: 200 ps
6. Read data from Data Mem, whose address provided by ALU: 250 ps
7. Mux for WB to Register File of result from Data Mem: 25 ps
8. Write back setup time for Register File registers: 20 ps
total ld = 950ps

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

1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction =

25 ps
[note - there is no number given for sign extension delay, so we can assume it is less than the register file delay of 150 ps]
5. ALU delay: 200 ps
6. Write Data from R2 to Data Mem: 250 ps
Total for sd: 905 ps

2.4 What is the latency of `beq`?

1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps
[note - there is no number given for sign extension delay, so we can assume it is less than the register file delay of 150 ps]
5. ALU delay: 200 ps
6. Single gate delay AND of ‘zero’ output from ALU and ‘Branch’ control output from Control unit: 5ps
7. Mux for Branch address to PC: 25 ps
8. Write back setup time for PC register: 20 ps
Total for beq: 705 ps

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

1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
4. Mux to pick data from either R2 or sign extended bit data from immediate field of instruction = 25 ps
[note - there is no number given for sign extension delay, so we can assume it is less than the register file delay of 150 ps]
5. ALU delay: 200 ps
6. Mux for WB to Register File of result from ALU: 25 ps
7. Write back setup time for Register File registers: 20 ps
total = 700ps

2.6 What is the minimum clock period for this CPU?

950 ps

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%

Cycle time = $T_{R\text{-type}}$ x $f_{R\text{-type}}$ + T_{ld} x f_{ld} + T_{sd} x f_{sd} + T_{beq} x f_{beq}
= 700ps x 0.52 + 950ps x 0.25 + 905ps x 0.11 + 705ps x 0.12 = 785.6ps
Speed-up = 950ps / 785.6 ps = 1.21

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?

Previous problem: 950ps, with multiplier added: 1250ps

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

Speedup = 950ps / (0.95 x 1250ps) = 0.8

3. b3 What is the slowest the new ALU can be and still result in improved performance?

Even if the new CPU has 5% fewer instructions, it can improve the cycle time by at most 50ps. However, this is insufficient to make up for the 300ps penalty by adding a multiplier unit. So, the new ALU cannot be any slower than it is to result in improved performance of the new CPU

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?

speedup = 950/(950ps + 10ps) x (1-36% x 12%) = 1.03

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

4196/3996 = 1.05 or increase in cost by 5%
Performance increase of 3% is significant for a cost increase of only 5%

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.

In scenarios where the financial returns are highly dependent on performance, even a modest 3% boost in performance could readily warrant a 10% rise in expenditure. Such situations are typically observed in high-performance computing processors utilized within data centers. Conversely, in cases where cost efficiency is the paramount concern, as in Internet of Things and embedded systems, it would be unreasonable to incur a 10% cost increase for the sake of a mere 3% performance enhancement.

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?

For the new ld instruction, items 5 and 6 below proceed in parallel: with the faster of these (ALU delay) removed from the critical path
1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
6. Read data from Data Mem, whose address provided by ALU: 250 ps
7. Mux for WB to Register File of result from Data Mem: 25 ps
8. Write back setup time for Register File registers: 20 ps
total ld = 725ps

For the new sd instruction, items 5 and 6 below proceed in parallel: with the faster of these (ALU delay) removed from the critical path
1. PC Register Read delay following rising edge of clock: 30 ps
2. Instruction Memory: 250ps
3. Read Register File: 150 ps
6. Write Data from R2 to Data Mem: 250 ps
Total for sd: 680 ps

So, new cycle time would be 725ps.

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.)

Since ld/sd instructions are 36% of all instructions in problem 2, 3 the number of instructions would increase by 1.36x. So, the new cycle time of 725ps would multiply with 1.36n to give 1020n ps. Speedup = 950 / 986 = 0.963

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

As load and store instructions are known to be the least efficient ones, the extent to which they are used will determine the magnitude of enhancements observed in the new CPU's performance. If the frequency of their usage is minimal, the drawback of having a higher instruction count is negligible, while the gain in cycle time improvement is significantly more substantial.

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?

Referring to the previous response, the outcome is contingent on the frequency of load and store (ld/sd) instructions. By eliminating the ALU operation for ld/sd, there's a substantial 22%

improvement in cycle time. Nevertheless, if ld/sd instructions make up more than 22% of the total instructions, then the overall performance of the CPU may not be improved.

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?

The ALU could be used to add the 2 registers rs1 and rs2, the shifter could be used to shift the offset in register rs2, so no new hardware is necessary.

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

Only the control unit needs modification.

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

No new data paths are needed.

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?
pipelined 350ps (slowest stage) non-pipelined 1250ps

6.2 What is the total latency of an `ld` instruction in a pipelined and non-pipelined processor?
pipelined and non-pipelined = 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?
The slowest stage: 350ps. Now pipelined version is 300ps cycle time.

6.4 Assuming there are no stalls or hazards, what is the utilization of the data memory?
load and store = 35% (the only instructions that access data memory)

6.5 Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?
65% - only ALU and load utilize write back to register file write port.

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.

In a pipeline consisting of k stages, the initial instruction completes its journey through all k stages in k cycles. Subsequently, the subsequent n-1 instructions proceed to complete one stage per cycle during the following n-1 cycles. Consequently, when you have a total of n instructions, it takes a total of k + n - 1 cycles for all of them to be executed within a k-stage pipeline.

8. Assume address in memory of '`A[0]`', '`B[0]`' and '`C[0]`') are stored in Registers `x27, x30, x31`. Assume values of variables `f, g, h, i,` and `j` are assigned to registers `x5, x6, x7, x28, x29` respectively Implement in RISC V these lines of C code :

`(i) f = g - A[B[C[64]]]`

```
lw x8, 256(x31)      # C[64]
slli x8, x8, 2
add x8, x8, x30      # x8 has &B[C[64]]
lw x8, 0(x8)         # x8 has B[C[64]]
slli x8, x8, 2
add x8, x8, x27      # x8 has &A[B[C[64]]]
lw x8, 0(x8)         # x8 has A[B[C[64]]]
sub x5, x6, x8
```

`(ii) f = g - A[C[16] + B[32]]`

```
lw x30, 128(x30)     # B[32]
lw x31, 64(x31)      # C[16]
add x24, x30, x31
slli x24, x24, 2
add x4, x27, x24     # &A[C[16] + B[32]]
lw x4, 0(x4)         # A[C[16] + B[32]]
sub x5, x6, x4       # f
```

`(iii) A[i] = 4B[8i-81] + 4C[32i+32]`

```
slli x28, x28, 3     # 8i
addi x26, x28, -81   # 8i-81
slli x26, x26, 2
add x26, x26, x30    # &B[8i-81]
slli x28, x28, 2     # 32i
addi x25, x28, 32    # 32i+32
slli x25, x25, 2
add x25, x25, x31    # &C[32i+32]
slli x24, x28, 2
add x24, x24, x27    # &A[i]
lw x23, 0(x26)       # B[8i-81]
slli x23, x23, 2     # 4B[8i-81]
```

```
lw x22, 0(x25)        # C[32i+32]
slli x22, x22, 2      # 4C[32i+32]
add x23, x23, x22     # 4B[8i-81]+4C[32i+32]
sw x23, 0(x24)        # A[i]
```

***9.* (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?

```
addix11, x12, 5

addx13, x11, x12

addix14, x11, 15
```

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.

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

```
addix11, x12, 5       #x11 does not update to 27 (=22+5) until the add instruction below
addx13, x11, x12      #x13 = 33
addix14, x11, 15      #x14 = 26
addx15, x11, x11      #x15 = 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
NOP
addx13, x11, x12
NOP
addix14, x11, 15
NOP
addx15, x13, x12
```