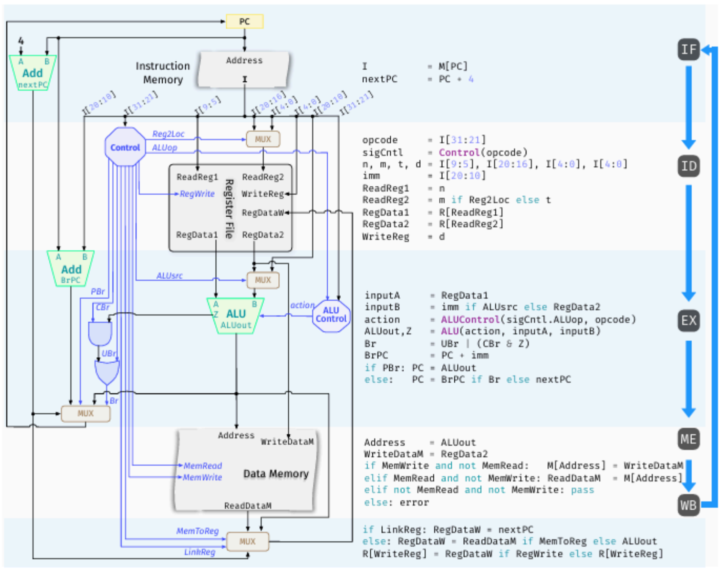
CS 382 Fall 2024 — Computer Architecture and Organization — Homework 3

Breona Pizzuta
"I pledge my honor that I have abided by the Stevens Honor System."

Requirement (READ FIRST!) You have to type the solutions. Handwritten homework will not be graded and will receive zero credit. You can annotate on this document directly (you might need to know how to insert a picture into a PDF file); or you can submit a separate PDF, but with solutions clearly marked with question numbers.

1 (15 points)

When silicon chips are fabricated, defects in materials (e.g., silicon) and manufacturing errors can result in defective circuits. A very common defect is for one signal wire to get "broken" and always register a logical 0. This is often called a "stuck-at-0" fault. Answer the following questions based on Figure 3.27 in the textbook.



```
I            = M[PC]
nextPC       = PC + 4

opcode       = I[31:21]
sigCntl      = Control(opcode)
n, m, t, d   = I[9:5], I[20:16], I[4:0], I[4:0]
imm          = I[20:10]
ReadReg1     = n
ReadReg2     = m if Reg2Loc else t
RegData1     = R[ReadReg1]
RegData2     = R[ReadReg2]
WriteReg     = d

inputA       = RegData1
inputB       = imm if ALUsrc else RegData2
action       = ALUControl(sigCntl.ALUop, opcode)
ALUout,Z     = ALU(action, inputA, inputB)
Br           = UBr | (CBr & Z)
BrPC         = PC + imm
if PBr: PC   = ALUout
else:   PC   = BrPC if Br else nextPC

Address      = ALUout
WriteDataM   = RegData2
if MemWrite and not MemRead:   M[Address] = WriteDataM
elif MemRead and not MemWrite: ReadDataM  = M[Address]
elif not MemRead and not MemWrite: pass
else: error

if LinkReg: RegDataW = nextPC
else: RegDataW = ReadDataM if MemToReg else ALUout
R[WriteReg] = RegDataW if RegWrite else R[WriteReg]
```

Figure 3.27: A summary of five stages each instruction goes through in the datapath, with description language on the side.

1.1 (5 points)

Which instructions fail to operate correctly if the Br wire is stuck at 0?

B, BL, CBZ, B.EQ, B.GT, CBNZ

Anything instructions with conditional or unconditional branching. This is because if the BR wire is stuck at 0, PC will never be able to update the correct destination address during branching.

1.2 (5 points)

Which instructions fail to operate correctly if the ALUsrc wire is stuck at 0?
AND, ADD, ADDS, SUB, SUBS, ORR, ANDS, LDR, STR
Any arithmetic and logical operations using Immediate value instructions will not operate correctly. Meaning that this will also affect LDR and STR that use an immediate for a byte offset.

1.3 (5 points)

Which instructions fail to operate correctly if the RegWrite wire is stuck at 0?
AND, ORR, SUB, ADD, LDR, ANDS, SUBS, ADDS, BL, and in some cases MOV
Any instructions that write to registers and all arithmetic and logic instructions. .

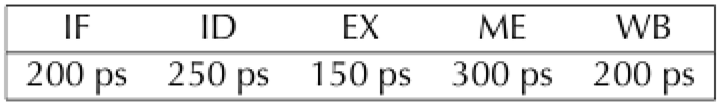
2 (10 points)

Consider adding a new instruction, SWAP Rd, Rn , to our architecture, which will swap the data stored in Rd and Rn. Discuss the necessary changes that must be made to the datapath above. There is no need to design a new datapath- just discuss what should be added to the existing one.

To add a new instruction SWAP Rd, Rn you would need to write from Rd to Rn and Rn to Rd at the same time which is not possible. To make this instruction work properly we would need to store either Rd or Rn in a temporary register so we can put the new value in one while still having the other saved. In this case we would need a WriteReg2 and a RegDataW2 to be able to have multiple reads/writes happening at one time. Doing two writes with the temporary register would also need the extra register and any other MUX that go with that. RegDataW2 would require a MUX and a control signal. The control signal needs information from opcode I[31:21], therefore we would also need different opcode.

3 (15 points)

Consider the addition of a multiplier to the CPU shown in Figure 3.27 in the textbook. This addition will add 200 ps to the latency of the ALU, but will reduce the number of instructions by 20% (because there will no longer be a need to emulate the multiplication instruction). Answer the following questions based on Chapter 3.3 of the textbook (no pipelining) and the following table for the latencies of the stages



IF	ID	EX	ME	WB
200 ps	250 ps	150 ps	300 ps	200 ps

3.1 (5 points)

What is the clock cycle time with and without this improvement?
Without improvement: 200+250+150+300+200=1100 ps
With improvement: 200+ 250+ (150+200)+ 300+ 200=1300 ps

3.2 (5 points)

What is the speedup achieved by adding this improvement?
Hint: *speedup* **= 1 − (***new time***/old time)**

For the new time we know it has 20% less instructions so that means 80% instructions due to a 20% reduction.
New time= 1300*0.8= 1040 ps
Oldtime= 1100 ps
Speedup= 1- (1040/1100)= 0.0545

Speedup= 0.0545%

3.3 (5 points)

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

It must be less then the old time (1100 ps) to be improved. Let's make x the variable newtime.
So: 0.8*x < 1100
X < 1100 / 0.8
x<1375
The slowest ALU time is the time left over after the total time of the 5 stages is subtracted from the greatest newtime: 1375 - 1100 (total time of stages) = 275ps
This means that with 20% less instructions, 275 ps is the slowest ALU can be to still have an improved performance.

4 (20points)

In this exercise, we examine how pipelining affects the clock cycle time of the processor. Questions in this problem assume that individual stages of the datapath have the latencies shown in Problem 3 above. Answer the following questions.

4.1 (5 points)

What is the clock cycle time in a pipelined and non-pipelined processor?
Non pipelined: 200+250+150+300+200=1100 ps per clock cycle (all stages)
Pipelined processor : 300 ps per clock cycle (longest stage)

4.2 (5 points)

What is the total latency of an LDR instruction in a pipelined and non-pipelined processor?
An LDR instruction must use all 5 stages.
Pipelined: 300* 5 stages= 1500 ps
Non-pipelined: 200+250+150+300+200=1100 ps

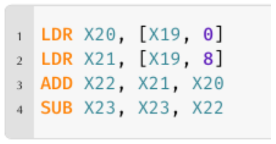
4.3 (10 points)

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?

We should split the longest stage. So, I would split ME because it takes the longest time (300ps), so now it would take 150 ps. This means the longest instruction would now be 250 ps at the ID stage. The clock cycle time would equal 250 ps while the pipeline's total latency would now equal 250(5)= 1250 ps.

5 (20 points)

Consider the following instructions executed in a pipeline with full forwarding support (including WB write/read in the same cycle). Identify the value of which register is forwarded from a stage of an instruction to a stage of a subsequent instruction. (Completely impossible example: "The value of X5 is forwarded from the IF stage of instruction 1 to the WB stage of instruction 2.") Submit a pipeline execution diagram as shown in the textbook.

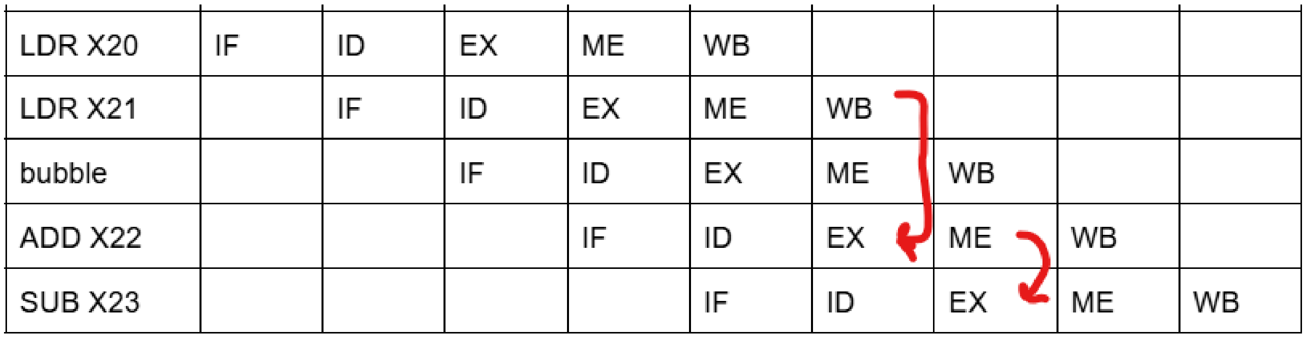


```
LDR X20, [X19, 0]
LDR X21, [X19, 8]
ADD X22, X21, X20
SUB X23, X23, X22
```

LDR X20, [X19, 0]
LDR X21, [X19, 8]
ADD X22, X21, X20
SUB X23, X23, X22
We will first have an issue with X21, and then an issue with X22. Therefore this would not work:

LDR X20	IF	ID	EX	ME	WB			
LDR X21		IF	ID	EX	ME	WB		
ADD X22			IF	ID	EX	ME	WB	
SUB X23				IF	ID	EX	ME	WB

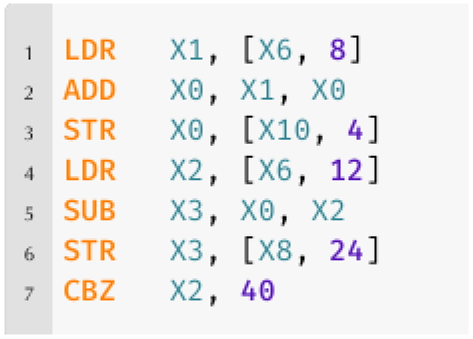
Fix it:
Insert a bubble between instructions 2 and 3 as you cannot forward from LDR to ADD
The value X21 is forwarded from WB stage of instruction 2 to the EX stage of instruction 3.
The value X22 is forwarded from ME stage of instruction 3 to the EX stage of instruction 4.



LDR X20	IF	ID	EX	ME	WB				
LDR X21		IF	ID	EX	ME	WB			
bubble			IF	ID	EX	ME	WB		
ADD X22				IF	ID	EX	ME	WB	
SUB X23					IF	ID	EX	ME	WB

6 (20points)

Consider the following instructions.



```
LDR   X1, [X6, 8]
ADD   X0, X1, X0
STR   X0, [X10, 4]
LDR   X2, [X6, 12]
SUB   X3, X0, X2
STR   X3, [X8, 24]
CBZ   X2, 40
```

IF	ID	EX	ME	WB

6.1 (10 points)

Add NOP instruction to the code above so that it will run correctly on a pipeline without forwarding, but WB write/read in the same cycle. (A pipeline execution diagram is not needed for this problem. Sketching it may help, but it will not be graded.)

LDR X1, [X6, 8]
NOP
NOP
ADD X0, X1, X0
NOP
NOP
STR X0, [X10, 4]
LDR X2, [X6, 12]
NOP
NOP
SUB X3, X0, X2
NOP
NOP
STR X3, [X8, 24]
CBZ X2, 40

6.2 (10 points)

Optimize the code execution by re-arranging the instructions to get the same correct result faster

```
LDR X1, [X6, 8]      //two needed before ADD
LDR X2, [X6, 12]    // two needed before SUB
NOP
ADD X0, X1, X0    // two needed before SUB
NOP
NOP
SUB X3, X0, X2  // two needed before STR X3
STR X0, [X10, 4]
NOP
STR X3, [X8, 24]
CBZ X2, 40     //don't move this
```

Less NOPs are used here, so we know it has been optimized.