ECE331 Homework 5

PDF

ECE 331 – Fall 2024
Homework 5
Assigned Friday, October 18th
Due Friday, October 25th

1. (20 points) The following sequence of instruction is executed in a basic five-stage pipelined processor discussed in the class. There is NO forwarding support.

```
ldur      x2,[x1,#0]
add       x3,x3,x2
add       x4,x1,x3
sub       x1,x4,x1
cbz       x1, exit
stur      x3,[x1,#0]
ldur      x5,[x1,#8]
sub       x2,x5,x6
add       x6,x2,x3
exit:
          add x1,x2,x3
```

(a) List all hazards in the above code. For each hazard, indicate the instruction which creates the hazard and all affected instructions.

(b) Rewrite the code including minimum number of nop instructions to eliminate potential hazards. You can reorder the instructions. Assume that the pipeline has full forwarding support. Also, assume that register read and write at the same address can happen in one clock cycle.

(c) Could branch prediction help eliminate the hazards in the code for this question? Please explain why or why not in one or two sentences.

Answers

1.

Hazard	Affected Instruction	Reason
Ldur x2, [x1, #0]	Add x3, x3, x2	the add instruction relies on x2 which is loaded by ldur
Add x3, x3, x2	Add x4, x1, x3	depends on x3 being updated by previous add
Add x4, x1, x3	Sub x1, x4, x1	depends on x4 being updated by previous add
Sub x1, x4, x1	Cbz x1, exit	depends on x1 being updated by previous sub

Cbz x1, exit	Stur x3, [x1, #0]
	Ldur x5, [x1, #8]
	Sub x2, x5, x6
	Add x6, x2, x3
		cbz needs these instructions to check set x1 to zero, so it needs to wait for them to execute correctly

Ldur x5, [x1, #8]	Sub x2, x5, x6	Ldur needs x5 to be updated from prev instruction
Sub x2, x5, x6	Add x6, x2, x3	Sub needs the updated x2 value from the add instruction before

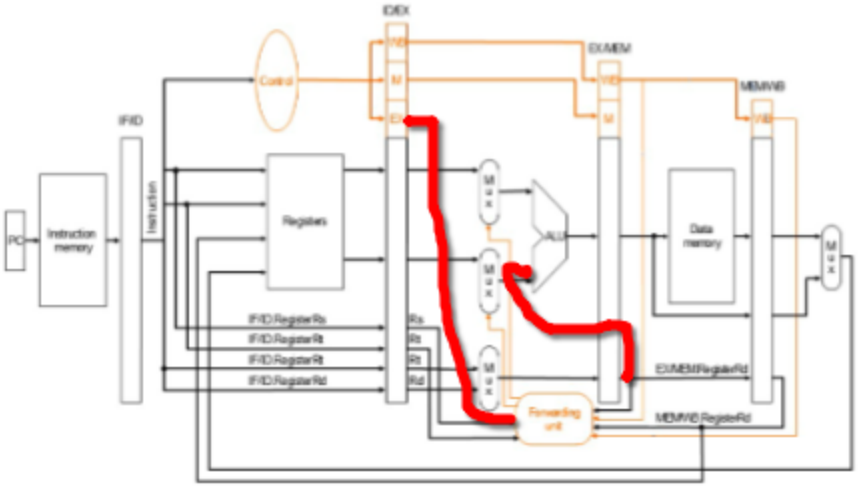
```
ldur x2. [x, #0] // load x2
nop // give time for ldur

add x3, x3, x2 //use x2
add x4, x1, x3 //use updated x3
sub x1, x4, x1 //use updated x4
cbz x1, exit // if x1 is zero
stur x3, [x1, #0] // x1 -> x3
ldur x5, [x, #8] // load x5
sub x2, x5, x6 // use x5
add x6, x2, x3 // use updated  x2

exit:
add x1, x2, x3 //execute after exit
```

c. branch would not help get rid of hazards because these hazards are dependent on previous instruction's data. a branch prediction works for branch instructions, not arithmetic instructions.

2. because arm processor is running with full forwarding support we don't need any nop instructions, the processor can pass the answer along before the instruction even finishes

```
ldur x1, [x19, #0] //load x1 into x19
ldur x2, [x19, #8] //load x2 into x19 with offset
sub x2, x2, x1 // x2 = x2 - x1
add x4, x2, x1 // x4 = x2 - x1
ldur x4, [x0, #8] // load x4 into x0 with offset
add x5, x4, x1 // x5 = x4 + x1
```

3. we can see that the arm code is loading x20 in the ldur instruction. sub relies on x20 so a hazard will result with x20 not being written to the register until after memory, we can write in the paths, in red to eliminate the hazards, making x20 always available to the ALU.


ID/EX
EX/MEM
MEM/WB
IF/ID
Control
WB
M
EX
Instruction memory
Instruction
Registers
ALU
Data memory
Forwarding unit
IF/ID.RegisterRs
IF/ID.RegisterRt
IF/ID.RegisterRt
IF/ID.RegisterRd
Rs
Rt
Rt
Rd
EX/MEM.RegisterRd
MEM/WB.RegisterRd