Ryan Young

CSE 3666

11/11/2019

Homework 6

1. Consider a pipeline with the following stage latencies:
   1. What is the clock-cycle time in a non-pipelined and a pipelined processor?

The clock cycle of a non-pipelined processor would be all the latencies added together which comes out to 300 + 100 + 250 + 350 + 50 = 1050 ps. In a pipelined processor the clock-cycle takes the same time as the longest latency in the pipeline so in this case it would be 350 ps.

* 1. What is the total latency of a LW instruction in a non-pipelined processor and a pipelined processor?

The total latency of a LW instruction in a non-pipelined processor is 300 + 100 + 250 + 350 + 50 = 1050 ps because LW needs all 5 stages of the pipeline. While the pipelined processor would take 5 \* 350 ps = 1750 ps. The pipelined implementation is slower because each clock cycle has to match the highest latency value.

* 1. If we can split one stage of the pipelined datapath into two new stages, each half the latency of the original stage, which stage should be split and what is the new clock-cycle time of the processor?

If we can split one stage of the pipelined datapath into two new stages, each half the latency of the original stage, the stage I would split would be MEM because it has the highest latency. So, the new clock-cycle time would be 300 ps, again because this is the next highest latency.

1. For the following sequence of instruction:

lw $t0, 0 ($s0)

lw $t1, 0 ($t0)

lw $t2, 0 ($t1)

addi $t2, $t2, 5

sw $t2, 0 ($t1)

addi $s0, $s0, 4

add $t6, $t4, $s5

2.1. Indicate the dependencies present in the given code.

|  |  |  |
| --- | --- | --- |
| Data Input Dependent | Data Output Dependent | Writing after Writing Dependent |
| Line 2 on Line 1 ($t0) | Line 6 on Line 1 ($s0) | Line 4 on Line 5 ($t2) |
| Line 3 on Line 2 ($t1) | Line 5 on Line 4 ($t2) | Line 3 on Line 4 ($t2) |
| Line 5 on Line 2 ($t1) |  | Line 5 on Line 3 ($t2) |
| Line 4 on Line 3 ($t2) |  |  |
| Line 5 on Line 4 ($t2) |  |  |

2.2. Assume there is no forwarding in this pipelined processor; add NOP instructions to eliminate any hazards.

lw $t0, 0 ($s0)

NOP

NOP

lw $t1, 0 ($t0)

NOP

NOP

lw $t2, 0 ($t1)

NOP

NOP

addi $t2, $t2, 5

NOP

NOP

sw $t2, 0 ($t1)

addi $s0, $s0, 4

add $t6, $t4, $s5

2.3. Assume there is forwarding in this pipelined processor; add NOP instructions to eliminate any hazards.

lw $t0, 0 ($s0)

NOP

lw $t1, 0 ($t0)

NOP

lw $t2, 0 ($t1)

NOP

addi $t2, $t2, 5

NOP

sw $t2, 0 ($t1)

addi $s0, $s0, 4

add $t6, $t4, $s5

3- For the following two sequences of instructions: (15 points)

a:

LW $t0, 0 ($sp)

ADD $t0, $t0, $t0

SW $t0, 0 ($sp)

b:

ADD $t0, $a0, $a1

SUB $t1, $t0, $t1

ADD $t2, $t0, $t1

|  |  |  |
| --- | --- | --- |
| Without Forwarding | With ALU-ALU Forwarding | With full forwarding |
| 250 ps | 275 ps | 300 ps |

Use the given clock cycle times to answer the following questions:

3.1. What is the total execution time of each instruction sequence without forwarding and with full forwarding? What is the speed-up achieved by adding full forwarding to a pipeline that had no forwarding?

a. Without forwarding it would take 11 clocks so 11 \* 250ps = 2750 ps. With full forwarding you can forward from the exe stage of ADD to the mem stage of SW and this gives us a time of 8 \* 300 = 2400 ps. This results in a speed up of 2750/2400 = 1.146.

b. Without forwarding it would take 11 clocks again producing the same time of 2750 ps. With full forwarding this drops to 7 cycles producing 7 \* 300 = 2100 ps. And a speedup of 2750/2100 = 1.31 ps.

3.2. Add NOP instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to EX stage.)

a.

LW $t0, 0 ($sp)

NOP

NOP

ADD $t0, $t0, $t0

SW $t0, 0 ($sp)

b. ADD $t0, $a0, $a1

SUB $t1, $t0, $t1

ADD $t2, $t0, $t1

3.3. What is the total execution time of each instruction sequence with only ALU-ALU forwarding? What is the speed-up over a pipeline without forwarding?

a. With only ALU-ALU forwarding we get a decrease in performance resulting in 9 \* 275 = 2475 ps. This gives us a speedup of 2750/2475 = 1.11.

b. With ALU only forwarding this again has 7 cycles but this time 7\*275 = 1925 giving us a speedup of 2750/1925 = 1.43.

4- Page 30, Exercise 4.8 (15 points)

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

The clock-cycle in a pipelined processor is always the same as the slowest element or the element with the highest latency in this case it would be 350 ps. For a non-pipelined processor, we need to add all the elements’ latency together = 250 + 350 + 150 + 200 + 300 = 1250 ps.

4.8.2 What is the total latency of an LW instruction in a pipelined and non-pipelined processor?

The total latency of an LW instruction in a pipelined processor would be 350 ps \* 5 = 1750 ps. This is because there are five stages and out longest clock is 350 ps. In a non-pipelined processor since the LW instruction uses all 5 stages so it would be 1250 ps.

4.8.3 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 largest stage (the ID stage) into halves thus making the new longest clock cycle 300 ps.

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

The utilization of the data memory is 35 % because SW is 15% of instructions run and LW is 20% of instructions run so adding them gives us 35%.

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

Every time we write to a register we have to use this write-registers port so all ALU operations (45%) + LW(20%) would use the port giving us 65%.

5- Page 361, Exercise 4.9 (15 points)

Note:

ALU-ALU forwarding is forwarding from the EX stage to the EX stage

Full forwarding is forwarding from the MEM stage to the EX stage and from the EX stage to the EX stage.

WAR: write after read

RAW: read after write

WAW: write after write

4.9.1 Indicate dependencies and their type.

R1 is read in line 2 after being written to in line 1. This is a RAW type dependence. R2 is read in line 3 after being written to in line 2. This is a RAW type dependence. R2 is being written to after being read from in line 1. This is a WAR dependence. R1 is written to in line 3 after being ready to in line 2 making a WAR dependence. Finally, R1 is written to in line 3 after being written to in line 1 creating a WAW dependency.

4.9.2 Assume there is no forwarding in this pipelined processor, indicate hazards and add NOP instructions to eliminate them.

There are no control hazards in this code because there is no branching occurring, there are however, data hazards and structure hazards, however. There is a structure hazard at line 1 and 3 because they both write to the same register this is fixed with the NOP statements. And there is a data hazard between line 1, line 2, and line 3. This is because line 3 uses data from line 1 and line 2 and line 2 uses data from line 1.

OR r1, r2, r3

NOP

NOP

OR r2, r1, r4

NOP

NOP

OR r1, r1, r2

4.9.3 Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them.

There are no hazards with full forwarding because you can get data directly from the EXE stage for the line of code from before and not have to wait. So, no NOP instructions are needed.

4.9.4 What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

Without forwarding: (7 cycles + 4 stalls ) \* 250 ps = 2750 ps.

With forwarding: ( 7 cycles + 0 stalls) \* 300 ps = 2100 ps.

Speedup = 2750/2100 = 1.31

4.9.5 Add NOP instructions to this code to eliminate hazards if there is ALU-ALU forwarding only.

There are no NOP instructions need because all lines of code can be done in the ALU. Thus, this means that all we need is ALU forwarding and ALU forwarding will eliminate the need to have stalls or NOP instructions here.

4.9.6 What is the total execution time of this instruction sequence with only ALU-ALU forwarding ? What is the speedup over a no-forwarding pipeline?

ALU-ALU forwarding allows us to have only 7 cycles and no stalls so 7 \* 290 ps = 2030 ps. This gives us a speedup of 2750/2030 = 1.35.

6- Page 362, Exercise 4.10 (20 points)

4.10.1

Since we can assume a hundred percent branch predict we know that we will not have any control hazard if we lineup the BEQ instruction’s mem stage and the ADD instruction’s IF. This makes the 5 staged pipelined sequence take 11 clock cycles. The NOP instructions cannot fix this issue because NOP instructions are needed to be fetched from instruction memory like all other instructions.

4.10.2

When we change from using memory to using registers as memory for save and load word instructions, we down can combine the stages of EX and MEM. This makes the number of cycles go from 9 to 8 and gives us a speedup of 1.13.

Addi r16, r16, 12

Sw r16, 0(r6)

Addi r16, r16, -4

Lw r16, 0(r6)

Beq r5, r4, Label

Add r5, r1, r4

Slt r5, r15, r4

4.10.3

If there are no hazards or stalls the number of cycles needed to complete the sequence of instructions is the number of instructions plus 4 aka 9 cycles. When the branch outcome is calculated in the ID stage the next instruction must be delayed one stage because we need to fetch the next instruction. However, when the branch is computed in the EX stage then we need to wait two stages. The speedup then would be 11 / 10 = 1.1 if the branch is calculated in the ID stage instead of in the EX stage.

4.10.4

Now the clock cycle will be 210 ps because we are adding 20 ps to the larger of two latencies which happens to be 190 ps. The speedup is now calculated by 9 \* 200 / 8 \* 210 = 1.07.

4.10.5

There is no change in speedup because the highest latency is still 200 ps after making that 50 percent change and the 10 ps change. So, the speedup is still 1.1.

4.10.6

EX stage now takes 130 ps. So, the new clock-cycle time is the same as the old one because it never changed the longest latency. Since there are 11 cycles and the longest latency is still 200 ps the execution time is 2200 ps. Now, since there are 12 cycles after moving the BEQ address computation to MEM. Speedup would be given by 2200 / 2400 = 0.92.

7- Page 364, Exercise 4.13 (20 points)

4.13.1

Add r5, r2, r1

NOP

NOP

LW r3, 4(r5)

LW r2, 0(r2)

NOP

Or r3, r5, r3

NOP

NOP

SW r3, 0(r5)

4.13.2

Add r5, r2, r1

LW r2, 0(r2)

NOP

LW r3, 4(r5)

NOP

NOP

Or r3, r5, r3

NOP

NOP

SW r3, 0(r5)

4.13.3

It does not require a hazard detection unit because the full forwarding is implemented perfectly. This is because we can go from the ALU in the add instruction to the ALU in the lw instruction, then go from the mem in lw to the ALU in the or instruction.

4.13.4

ADD instruction: PCWrite = 1, ALUIN1 = x and ALUIN2 = x

LW instruction: PCWrite = 1, ALUIN1 = x and ALUin2 = x

LW instruction: PCWrite = 1, ALUIN1 = 0 and ALUin2 = 0

OR instruction: PCWrite = 1, ALUIN1 = 1 and ALUin2 = 0

SW instruction: PCWrite = 1, ALUIN1 = 1 and ALUin2 = 0