Skill Assessment #7

Graded

Student

HARRY KIM

Total Points

109.925 / 125 pts

Question 1

A1 — **12.25** / 25 pts

1.1 **Part A** — **1.5** / 6.25 pts

✔ **+ 1.5 pts** Poor

A pipelined processor has 5 stages, and unpipelined processor has a CPI of 1.

1.2 **Part B** — **1.5** / 6.25 pts

✔ **+ 1.5 pts** Poor

CPI will increase as it was 1 previously, however not it can be more than 1.

1.3 **Part C** — **3** / 6.25 pts

✔ **+ 3 pts** Fair

It will be much faster as each cycle is effectively 1/5th of a cycle of an unpipelined processor, thus having a larger frequency. Frequency is cycles per second even if each cycle does less each time.

1.4 **Part D** — **6.25** / 6.25 pts

✔ **+ 6.25 pts** Excellent

Question 2

A2 — **60.175** / 62.5 pts

2.1 **Part A** — **13.175** / 15.5 pts

✔ **+ 13.175 pts** Good

average cpi 1.928, ex time 27 micro seconds

2.2 **Part B** — **25** / 25 pts

✔ **+ 25 pts** Excellent

2.3 **Part C** — **15.5** / 15.5 pts

✔ **+ 15.5 pts** Excellent

2.4 **Pard D** — **6.5** / 6.5 pts

✔ **+ 6.5 pts** Excellent

Question 3

A3 **37.5** / 37.5 pts

3.1 **Part A** **19** / 19 pts

✔ **+ 19 pts** Correct. Solution is correct and explanation is comprehensive

3.2 **Part B** **9.25** / 9.25 pts

✔ **+ 9.25 pts** Correct. Solution is correct and explanation is comprehensive

3.3 **Part C** **9.25** / 9.25 pts

✔ **+ 9.25 pts** Correct. Solution is correct and explanation is comprehensive.

Question 4

Late Penalty (if Appropriate) **0** / 0 pts

✔ **– 0 pts** Correct

Questions assigned to the following page: 1.2, 1.3, 1.4, 1.1, and 2.1

Harry Kim
u1226472

Skill Assessment #7

A1.

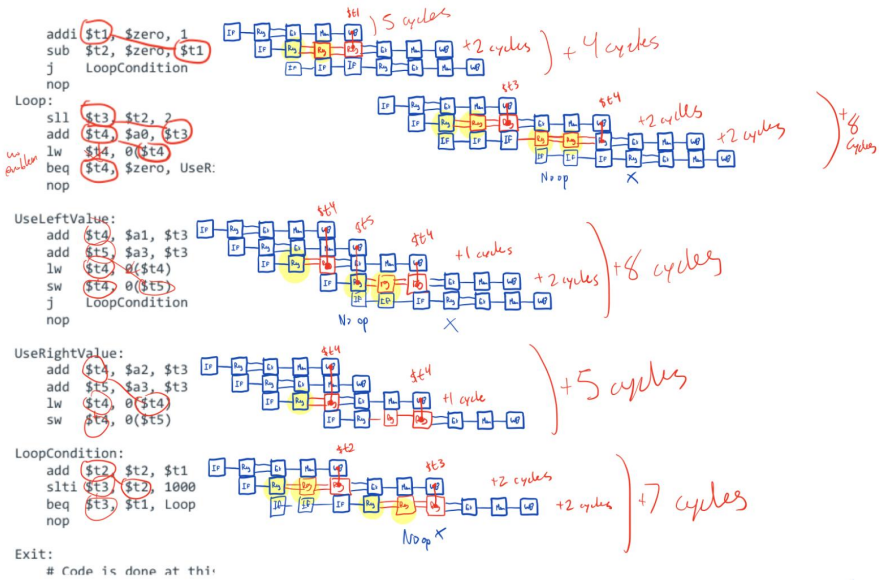
(a) In a non-pipelined instruction, the average CPI (cycles per instruction) would be how many stages the processor has. This is because only one instruction is being executed at a time and the next instruction does not begin until the previous instruction is done with all stages of the processor. If a processor had five stages and each stage took one clock cycle, then the CPI of this processor would be 5 CPI.

(b) I would expect the CPI to decrease since with pipelining, multiple instructions are being executed each at different stages of the processor . This means that we can start a new instruction at nearly every clock cycle and multiple instructions are being executed during a single machine cycle and thus the cycles per instruction decreases.

(c) I wouldn't expect the clock frequency to change because we aren't making the processor read and execute instructions faster but rather that we are increasing how many we can run in a single cycle. Pipelining only increases the throughput for the same clock speed.

(d) I would expect the latency to increase a bit but significantly increase the throughput. The increase in latency is because the slowest stage of executing the instruction determines the clock cycle. The significant increase in throughput comes from being able to execute a new instruction every clock cycle and multiple instructions each machine cycle. This multiplies the amount of instructions we can finish executing per shorter amount of time.

A2.



Questions assigned to the following page: 2.2 and 2.1

Harry Kim
u1226472

(a) The first instruction will take 5 cycles to get through the five stages of the pipeline. The instructions before LoopCondition add 4 cycles. Instructions under LoopCondition add 7 cycles, Loop adds 8 cycles, and then UseLeftValue adds 8 cycles. Loop condition, Loop, and UseLeftValue are repeated 500 times. The same thing happens again except that we add 5 cycles UseRightValue instead of UseLeftValue. Then the last loop condition accounts for an additional 7 cycles and then the code ends. All the cycles total up to 21,516 cycles.

$$5 + 4 + (7 + 8 + 8) * 500 + (7 + 8 + 5) * 500 + 7 = 21561\, cycles$$

For instructions, the instructions before LoopCondition add up to 4 instructions. Instructions underLoopCondition add up to 4 instructions, Loop has 5 instructions, and UseLeftValue has 6 instructions. Instructions under Loop condition, Loop, and UseLeftValue are repeated 500 times. The same thing happens again except that we add 4 instructions UseRightValue instead of UseLeftValue. The last LoopCondition instructions have 4 instructions. All instructions total up to 14,008 instructions.

$$4 + (4 + 5 + 6) * 500 + (4 + 5 + 4) * 500 + 4 = 14008\, instructions$$

To find the average CPI for this program, we simply divide the cycles by instructions which gives us:

$$21561\, cycles\ /\ 14008\, instructions\ =\ 1.54\, CPI$$

Because the processor has a 1 Ghz clock, it means that it can perform $1.0 * 10^9$ cycles per second. By dividing our total number of cycles by the processor clock speed, we get the following:

$$21516\, cycles\ /\ (1.0 * 10^9\ cycles\ /\ sec) = 2.1516 * 10^{-5}\, seconds$$

So this processor will take $2.1516 * 10^{-5}\, seconds$ to execute the given program.

(b) I have made the following adjustments to the code to avoid stalls and wasted cycles:

(i) Moved the first sub instruction after the jump instruction to fill the nop instruction and to reduce a cycle which comes from sub depending on addi to give $t1.

(ii) I moved “add $t5, $a3, $t3” from both UseLeftValue and UseRightValue under the beq instruction in Loop to fill the nop instruction.

(iii) I moved the remaining add instruction under UseLeftValue under the jump instruction to fill the nop instruction.

(iv) I noticed that both UseLeftValue and UseRightValue compute “lw $t4, 0($t4)” and “sw $t4, 0($t5)” after computing their respective addresses, so I moved those two instructions from both UseLeftValue and UseRightValue to LoopCondition. I

Questions assigned to the following page: 2.2 and 2.3

placed “lw $t4, 0($t4)” right before the beq instruction and “sw $t4, 0($t5)” after the beq instruction in order to fill up the no operation and to reduce a cycle between those two instructions as sw depends on lw to give $t4.

```
   # This code copies data values into a destination array.
   # The data values are copied from either the 'left'
   # array or the 'right' array.
   # For each element position, a third 'Selector' array tells us which
   # value to copy into the destination array.
   #
   # Assume $a0 is selector addr
   # Assume $a1 is left addr
   # Assume $a2 is right addr
   # Assume $a3 is dest addr
   #
   # $t2 is an index for keeping an array index
   # There are 1,000 elements in each array

   addi $t1, $zero, 1    # Put a 1 in $t1 as a constant
   j    LoopCondition
   sub  $t2, $zero, $t1  # Use $t2 for an index, set the index to -1
Loop:
   sll  $t3, $t2, 2      # Multiply the index by 4, we'll use it several places
   add  $t4, $a0, $t3    # Compute the selector array address
   lw   $t4, 0($t4)      # Load the value from the selector array
   beq  $t4, $zero, UseRightValue  # If the selector is zero, copy from right array
   add  $t5, $a3, $t3    # Compute the dest address

UseLeftValue:
   j    LoopCondition    # Jump to check loop condition
   add  $t4, $a1, $t3    # Compute the left address

UseRightValue:
   add  $t4, $a2, $t3    # Compute the right address

LoopCondition:
   add  $t2, $t2, $t1    # Advance the index by 1
   slti $t3, $t2, 1000   # See if the index is less than one thousand
   lw   $t4, 0($t4)      # Load the value from the left array
   beq  $t3, $t1, Loop   # If we're not done yet, loop again
   sw   $t4, 0($t5)      # Store the value to the dest array

Exit:
   # Code is done at this point.
```

(c) The first instruction will take 5 cycles to get through the five stages of the pipeline. The instructions before LoopCondition add 3 cycles. Instructions under LoopCondition add 6 cycles, Loop adds 9 cycles, and then UseLeftValue adds 2 cycles. Loop condition, Loop, and UseLeftValue are repeated 500 times. The same thing happens again except that we add 1 cycle UseRightValue instead of UseLeftValue. Then the last loop condition

Question assigned to the following page: 2.3

accounts for an additional 6 cycles and then the code ends. All the cycles total up to 16,514 cycles.

$$5 + 3 + (6 + 9 + 2) * 500 + (6 + 9 + 1) * 500 + 6 = 16514\, cycles$$

For instructions, the instructions before LoopCondition add up to 3 instructions. Instructions underLoopCondition add up to 5 instructions, Loop has 5 instructions, and UseLeftValue has 2 instructions. Instructions under Loop condition, Loop, and UseLeftValue are repeated 500 times. The same thing happens again except that we add 1 instruction UseRightValue instead of UseLeftValue. The last LoopCondition instructions have 5 instructions. All instructions total up to 11,508 instructions.

$$3 + (5 + 5 + 2) * 500 + (5 + 5 + 1) * 500 + 5 = 11508\, instructions$$

To find the average CPI for this program, we simply divide the cycles by instructions which gives us:

$$16514\, cycles \,/\, 11508\, instructions \;=\; 1.43\, CPI$$

Because the processor has a 1 Ghz clock, it means that it can perform $1.0 * 10^9$ cycles per second. By dividing our total number of cycles by the processor clock speed, we get the following:

$$16514\, cycles \,/\, (1.0 * 10^9\, cycles \,/\, sec) = 1.6514 * 10^{-5}\, seconds$$

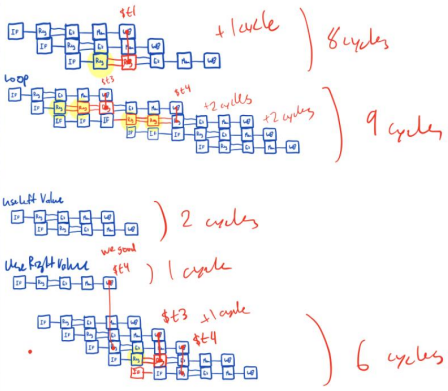
So this processor will take $1.6514 * 10^{-5}\, seconds$ to execute the given program.

As seen in the work provided below, the dependencies are preserved and the code should still output the same result.

Questions assigned to the following page: 2.3, 2.4, and 3.1

Harry Kim
u1226472

```
# This code copies data values into a destination array.
# The data values are copied from either the 'left'
# array or the 'right' array.
# For each element position, a third 'Selector' array tells us which
# value to copy into the destination array.
#
# Assume $a0 is selector addr
# Assume $a1 is left addr
# Assume $a2 is right addr
# Assume $a3 is dest addr
#
# $t2 is an index for keeping an array index
# There are 1,000 elements in each array

  add  $t1, $zero, 1   # Put a 1 in $t1 as a constant
  j   LoopCondition
  sub  $t2, $zero, $t1 # Use $t2 for an index, set the index to -1
Loop:
  sll  $t3, $t2, 2     # Multiply the index by 4, we'll use it several places
  add  $t4, $a0, $t3   # Compute the selector array address
  lw   $t4, 0($t4)     # Load the value from the selector array
  beq  $t4, $zero, UseRightValue  # If the selector is zero, copy from right array
  add  $t5, $a3, $t3   # Compute the dest address

UseLeftValue:
  j   LoopCondition    # Jump to check loop condition
  add  $t4, $a1, $t3   # Compute the left address

UseRightValue:
  add  $t4, $a2, $t3   # Compute the right address

LoopCondition:
  add  $t2, $t2, $t1   # Advance the index by 1
  slti $t3, $t2, 1000  # See if the index is less than one thousand
  lw   $t4, 0($t4)     # Load the value from the left array
  beq  $t3, $t1, Loop  # If we're not done yet, loop again
  sw   $t4, 0($t5)     # Store the value to the dest array

Exit:
  # Code is done at this point.
```



(d) If we added a branch architecture to add a branch predictor that was 90% accurate and eliminated the branch delay slots, it would speed up the code, but because my solution depends on the branch delay slots to do useful work, it would mess up the program and render it useless. I would have to rework the code to account for the missing branch delay slots.

A3.

(a) In order to solve for the accuracy of the branch predictor, we can set of the following expression:

$$1.28 = (0.86)(1.21) + (0.14)((x)(1.21) + (1 - x)(2.21))$$

With 1.28 being the average CPI of the entire program. $(0.86)(1.21)$ is all of the data instructions (non-branch), which accounts for 86% of the code, and $(0.14)((x)(1.21) + (1 - x)(2.21))$ is all of the branch instructions, which accounts for 14% of the code, with $(x)(1.21)$being the correct branch prediction and $(1 - x)(2.21)$ being the incorrect branch prediction. 2.21 comes from the stall penalty which is the CPI

Questions assigned to the following page: 3.3, 3.1, and 3.2

of the branch instructions plus 1 clock cycle (1.21 + 1). We can then solve for x which gives us:

$$x = 0.5$$

So the branch prediction accuracy is 50%.

(b) Prediction: Branch always taken:

Prev	Now		Prob
T	T	Correct	$0.5 * 0.5 = 0.25$
T	N	Incorrect	$0.5 * 0.5 = 0.25$
N	T	Correct	$0.5 * 0.5 = 0.25$
N	N	Incorrect	$0.5 * 0.5 = 0.25$

For always taken, the branch predictor would be incorrect 50% of the time. This would be the same for never taken so it wouldn't make much of a difference.

(c) Prediction: Two bit predictor:

2 cycles ago	Prev	Now		Prob
T	T	T	Correct	$0.5 * 0.5 * 0.5 = 0.125$
T	T	N	Incorrect	$0.5 * 0.5 * 0.5 = 0.125$
T	N	T	Incorrect	$0.5 * 0.5 * 0.5 = 0.125$
T	N	N	Correct	$0.5 * 0.5 * 0.5 = 0.125$
N	T	T	Correct	$0.5 * 0.5 * 0.5 = 0.125$
N	T	N	Incorrect	$0.5 * 0.5 * 0.5 = 0.125$
N	N	T	Incorrect	$0.5 * 0.5 * 0.5 = 0.125$
N	N	N	Correct	$0.5 * 0.5 * 0.5 = 0.125$

The performance would not improve because it would still be 50% correct and 50% incorrect.