Homework #3

Assigned: 18/04/2023
Due: 30/04/2023

1. Consider a five-stage pipelined datapath for MIPS architecture with the following latencies:

IF	ID	EX	MEM	WB	Pipeline registers
235 ps	115 ps	165 ps	235 ps	100 ps	15 ps

Note that pipeline registers also have some latency, namely 15 ps (last column in the table). (**20 pts**)

a. Assuming there are no stalls, what is the speedup achieved by pipelining a single-cycle datapath? (**10 pts**)

235 + 115 + 165 + 235 + 100 = 850 ps
Largest stage is 235 ps, which should be clock period. So 235 + 15 from pipeline registers = 250 ps
850/250 = 3.4 = 340%

b. Assume that instructions of a benchmark executed by the pipelined processor are broken down as follows:

Load	Store	Branch	Jump	R-type
30%	15%	10%	5%	40%

Also assume that 40% of loads are immediately followed by an instruction that uses the result of load; and branch penalty on misprediction is three clock cycles and 20% of branches are mispredicted. Jumps take two clock cycles to execute. Compute CPI of the pipelined processor (**10 pts**).

Assuming CPI = 1 in original:
Overall, 12% stalls (load):
30/100 * (40/100 * 2 + 60/100 * 1) + 15/100 * 1 + 10/100 * (20/100 * 3 + 80/100 * 1) + 5/100 * 2 + 40/100 * 1 = 1.21 CPI on avg

2. Consider the following 10-bit floating-point number system in which we use 1 bit for the sign, 4 bits for the exponent, and 5 bits for the significand (mantissa). The exponent bias is equal to 7, and the largest and smallest biased exponents are reserved (similar to IEEE 754 Standard). The significand uses a hidden (implicit) 1. The value of a floating-point number can be calculated using the following expression:

$$value = (-1)^{sign} \times (1.0 + significand) \times 2^{exponent-bias}$$ (**20 pts**)

a. Find floating-point representation of the following real numbers: (**10 pts**)

x) 45.0

Binary form: 101101.0 (normalized version: 1.01101 * 2^5)

Sign bit: 0

Exponent bits: 1110 (bias: 7, exponent: 5)

Fraction bits: 0 1101

Result: 0 1100 0110 1

y) 0.75

Binary form: 0.11 (normalized version: 1.1 * 2^-1)

Sign bit: 0

Exponent bits: 0110 (bias: 7, exponent: -1)

Fraction bits: 1 0000

Result: 0 0110 1000 0

z) -44.0

Binary form: 101100.0 (normalized version: 1.011 * 2^5)

For +44: Binary form: 0010 1100 (normalized version: 1.01100 * 2^5)

Sign bit: 1 (-)

Exponent bits: 1100 (bias: 7, exponent: 5)

Fraction bits: 0 1100

Result: 1 1100 0110 0

b. Compute `t = x+y` using precise floating-point arithmetic with guard (G), Round (R), and Sticky (S) bits. Use round-to-nearest-even scheme if you need to. (**5 pts**)

								G	R	S
x		1.	0	1	1	0	1	0	0	0
y		0.	0	0	0	0	0	0	1	1
t	+	1.	0	1	1	0	1	0	1	1
t		1.	0	1	1	0	1	0	1	1

No need to normalize further.
t = 1.01101 * 2^5

c. Compute `v = t+z` the same way. Is what you found correct? (**5 pts**)

t = 1.01101 * 2^5
z = 1.01100 * 2^4 -> shifted to right by 1 to match the the exponent of t.

								G	R	S
t		1.	0	1	1	0	1	0	0	0
z	+	0.	1	0	1	1	0	0	0	0
v		1 1	0	0	0	1	1	0	0	0

Don't ignore overflow bit.
Normalized result: v = 1.10001 * 2^6 (we have 1 more 1 bit on leftside.)
Round to nearest even: 1.1001 * 2^6

3. Assume that your benchmark has the following instruction frequencies:

	R-type	branch	the rest
Benchmark	40%	20%	40%

Also assume the following branch predictor accuracies:

	Always-taken	Always-not-taken	Dynamic predictor
Benchmark	45%	55%	60%

Assume also that CPI = 1 without branch mispredictions. (**20 pts**)

a. Stall cycles due to mispredicted branches increase CPI. Assume that the branch outcomes are determined in **`MEM`** stage, that there are no data hazards, and that no delay slots are used. Calculate the CPI after the stalls due to mispredicted branches with "Always-taken", "Always-not-taken", and "Dynamic" predictors? (Hint: When a branch is mispredicted, the instructions in **`IF`**, **`ID`**, and **`EX`** stages must be flushed out of the pipeline) (**10 pts**)

Branch misprediction stall is 3 cycles and CPI without branch misprediction is 1 cycles
When always taken:
misprediction rate is 100-45 = 55%
Average CPI = 1 + 0.2*0.55*3 = 1.33

When always not taken:
Misprediction rate is 100-55 = 45%
Average CPI = 1 + 0.2*0.45*3 = 1.27

With dynamic prediction:
misprediction rate is 100-60 = 40%
Average CPI = 1 + 0.2*0.4*3 = 1.24

b. With the "dynamic" predictor, what speedup would be achieved if we eliminated half of the branches by turning each branch into two R-type instructions? Assume that the performance of the predictor improves to 90% after this modification. (**Hint**: `clock count = IC × CPI`) (**10 pts**)

Initial average CPI with dynamic predictor was 1.24
New R-type count = 0.4k + 0.1k * 2 = 0.6k
New branch count = 0.1k
others = still 0.4k
1.1k count -> 0.1k is branch,
New frequencies: R type 55%, branch 9%, %36
Average CPI = 1 + 0.09*0.1*3 = 1.027
Speedup = 1.24/1.027 = 20.74% faster

4. Consider a program consisting of five conditional branches. Below are the outcomes of each branch for one execution of the program (T for taken, N for not taken).

Branch 1: T-T-T
Branch 2: N-N-N-N
Branch 3: T-N-T-N-T-N
Branch 4: T-T-T-N-T
Branch 5: T-T-N-T-T-N-T

For dynamic schemes, assume each branch has its own prediction buffer. List the predictions for the following branch prediction schemes for the execution of the code above. What are the total prediction accuracies? (**20 pts**)

a. There is one 1-bit dynamic predictor for each branch, and each is initialized to "Predict Not Taken". (**5 pts**)

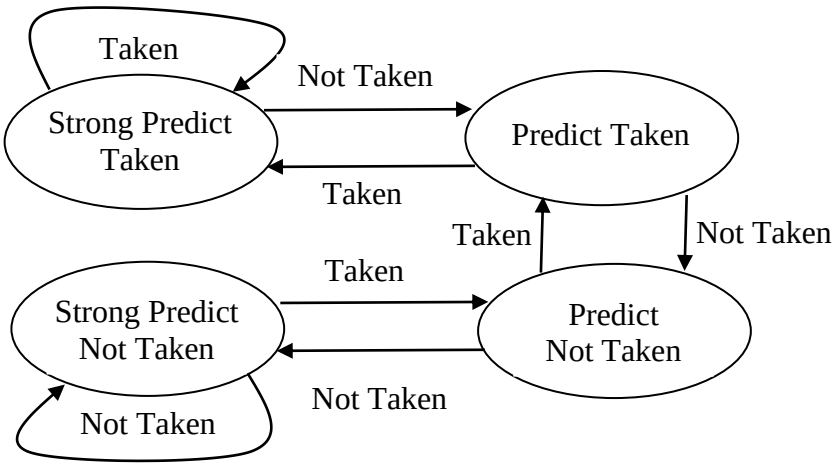
Branch 1: N T T -> 2 CORRECT 1 WRONG
Branch 2: N N N N -> 4 CORRECT 0 WRONG
Branch 3: N T N T N T -> 0 CORRECT 6 WRONG
Branch 4: N T T T N -> 2 CORRECT 3 WRONG
Branch 5: N T T N T T N -> 2 CORRECT 5 WRONG

Overall accuracy (correctness): 10/25 = 40%

b. You use 2-bit dynamic predictor and that each branch has its own prediction buffer which is initialized to "Predict Not Taken". Compute the prediction accuracy for each branch and overall branch prediction accuracy. (**15 pts**)

Branch 1: N T T -> 2 CORRECT 1 WRONG
Branch 2: N N N N -> 4 CORRECT 0 WRONG
Branch 3: N T N T N T -> 0 CORRECT 6 WRONG
Branch 4: N T T T T -> 3 CORRECT 2 WRONG
Branch 5: N T T T T T T -> 4 CORRECT 3 WRONG

Overall accuracy: 13/25 = 52%



5. (**25 pts**) Consider the following piece of code that will be executed in a five-stage pipelined datapath of MIPS processor:

```
lw  $t1, -16($t5)
add $t6, $t2, $t2
sw  $t6, 50($t1)
```

Assume that a value written to a register can be read in the following clock cycle.

a. Indicate dependencies. (**5 pts**)

In sw instruction, $t1 depends on $t1 in lw instruction -> IF/ID.rt = MEM/WB.rd
In sw instruction, $t6 depends on $t6 at add instruction -> IF/ID.rt = EX/MEM.rd

b. Assume that there is no forwarding in this pipelined processor. Indicate hazards and add **nop** instructions to eliminate them. (**5 pts**)

Hazards:

MEM/WB.rd = IF/ID.rt, IF/ID.rt = EX/MEM.rd
Before the first instruction loads word to $t1, sw instruction may try to access it.
Similarly, before $t6 register is overwritten with new value (2nd instruction), sw (3rd) may save the old value to memory.

Nops:

Since both second and first instructions has a conflict with 3rd instruction, we need to wait 2 cycles for data to be written from memory and ALU computes add operation and writes to respected register.

```
lw  $t1, -16($t5)
add $t6, $t2, $t2
nop
nop
sw  $t6, 50($t1)
```

c. Assume there is forwarding only from the ALU. Indicate hazards and add nop instructions to eliminate them. (**5 pts**)

Hazards:

Before the first instruction loads word to $t1 register, sw instruction may try to access it. In the forwarding case, there has to be a forwarding from MEM to ALU to calculate address for sw instruction.
Similarly, before $t6 register is overwritten with new value (2nd instruction), sw (3rd) may save the old value to memory. In forwarding case, there has to be a forwarding from the ALU result of addition to the mem.
Since there is no forwarding from memory, we still need to wait 2 clock cycles for the lw operation. In terms of addition, we have to wait one, so 1 nope is enough.
(MEM/WB.rd = IF/ID.rt)

Nops:

Same.

```
lw  $t1, -16($t5)
add $t6, $t2, $t2
```

```
nop
sw  $t6, 50($t1)
```

d. Assume there is full forwarding. Indicate hazards and add `nop` instructions to eliminate them. (**5 pts**)

There is no hazard since loaded word from memory and result of addition from ALU is forwarded to respected components by the same time they are needing it. No nops needed.

Use the following clock cycle times for the remainder of this exercise. Notice that the forwarding technique complicates the data path, which can result in slower clock frequency.

Without forwarding	With ALU forwarding	With full forwarding
300 ps	360 ps	400 ps

e. What is the total execution time of this instruction sequence without forwarding, with ALU-forwarding and with full forwarding? (**5 pts**)

Without forwarding:
9 cycles are needed. (5 + 1+1 + 2 nops)
9*300=2700 ps

With ALU-forwarding:
8 cycles are needed. (5 + 1+ 1 + 1 nop)
8*360=2880 ps

Full forwarding:
Only 7 cycles are needed. (5 + 1 + 1 + 0 nop)
7*400 = 2800 ps