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**)

The cycle time of a pipelined datapath is determined by the slowest stage. Slowest stages are IF and MEM stages.
So, the cycle time for the pipelined datapath = 235 ps + 15 ps = 250 ps

In single cycle datapath all stages are executed sequentially in one clock cycle.
So, the cycle time for the single cycle datapath = 235 ps + 115 ps + 165 ps + 235 ps + 100 ps = 850 ps

Speedup = single-cycle cycle time / pipelined cycle time
= 850 ps / 250 ps = 3.4

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**).

I assumed that there is forwarding after memory access.
If the result of a load is immediately used the pipeline stalling becomes like this:

IF - > ID -> EX -> MEM -> WB
bubble bubble bubble bubble
IF -> ID-> EX -> MEM-> WB

This costs 1 extra clock cycle. It adds 0.3 * 0.4 * 1 to CPI.

10% of the instructions are branch instructions and 20% of them are mispredicted with 3 clock cycle penalty. This adds 0.1 * 0.2 * 3 to CPI.

5% of the instructions are jump instructions and they took 2 clock cycles. This adds 0.05 * 2 to CPI.

Also, we need to calculate the CPI of the instructions which takes 1 clock to execute.

load = 0.3 * 1 = 0.3
store = 0.15 * 1 = 0.15
branch prediction rate is 0.8 it gives = 0.10 * 0.8 * 1 = 0.08
Jump is calculated.
R-type = 0.4 * 1 = 0.4

CPI = 0.3 * 0.4 * 1 + 0.1 * 0.2 * 3 + 0.05 * 2 + 0.3 + 0.15 + 0.08 + 0.4 = 1.21

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

45.0 = 101101 in binary = $1.01101 * 2^5$
Sign bit = 0
Significand = 01101
Exponent - bias = 5
Exponent = 5 + 7 = 12 = 1100 in binary

Floating point representation = 0 1100 01101

y = 0.75

0.75 = ¾ = 3 / (2^2) = 3 * 2^{-2} = 11 * 2^{-2} = 1.1 * 2^{-1}
Sign bit = 0
Significand = 10000
Exponent - bias = -1
Exponent = -1 + 7 = 6 = 0110 in binary

Floating point representation = 0 0110 10000

z = -44.0

-44.0 = 101100 in binary = 1.01100 * 2^5
Sign bit = 1
Significand = 01100
Exponent - bias = 5
Exponent = 5 + 7 = 12 = 1100 in binary

Floating point representation = 1 1100 01100

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	1	1	0
t		1	0	1	1	0	1	1	1	0
t		1	0	1	1	1	0			

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

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

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**)

The misprediction penalty is 3 clock cycles because we need to flush IF, ID, and EX stage results.

Always taken (misprediction rate is %55) CPI:
0.4 * 1 + 0.4 * 1 + 0.2 * 0.45 * 1 + 0.2 * 0.55 * 4 = 1.33

Always-not-taken (misprediction rate is %45) CPI:
0.8 * 1 + 0.2 * 0. 55 * 1 + 0.2 * 0.45 * 4 = 1.27

Dynamic predictor (misprediction rate is %40) CPI:
0.8 * 1 + 0.2* 0.6 * 1 + 0.2 * 0.4 * 4 = 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**)

We had 20k branch, 40k R-type, 40k other instructions. After the dynamic predictor, we have 10k branch, 60k R-type, 40k other instructions.
Branch = %9, R-Type = %55, Rest = % 36
R Type + Rest = %91
New CPI = 1 + 0.09 X 0.10 X 3 = 1.027
Speedup = Old CPI / New CPI = 1.24/1.027

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:

Predictions:	N-T-T
Actual branches:	T-T-T
Correctness (1 means correct, 0 means not correct):	0-1-1
Prediction accuracy:	2/3 = %66.6

Branch 2:

Actual branches:	N-N-N-N
Predictions:	N-N-N-N
Correctness (1 means correct, 0 means not correct):	1- 1- 1- 1
Prediction accuracy:	4/4 = %100

Branch 3:

Actual branches:	T-N-T-N-T-N
Predictions:	N-T-N-T-N-T
Correctness (1 means correct, 0 means not correct):	0-0-0-0-0-0
Prediction accuracy:	0/6 = %0

Branch 4:

Actual branches:	T-T-T-N-T
Predictions:	N-T-T-T-N
Correctness (1 means correct, 0 means not correct):	0- 1- 1-0-0
Prediction accuracy:	2/5 = %40

Branch 5:

Actual branches:	T-T-N-T-T-N-T
Predictions:	N-T-T-N-T-T-N

Correctness (1 means correct, 0 means not correct): 0-1-0-0-1-0-0
Prediction accuracy: 2/7 = %28.57
In total we made 25 predictions and 10 of them were correct.
Thus, the total accuracy is 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**)

ST means strong taken SN means strong not taken, T means taken, N means not taken
Branch 1:

Actual branches: T-T-T
Predictions: N-T-ST
Correctness (1 means correct, 0 means not correct): 0-1-1
Prediction accuracy: 2/3 = %66.6

Branch 2:

Actual branches: N-N-N-N
Predictions: N-SN-SN-SN
Correctness (1 means correct, 0 means not correct): 1- 1- 1- 1
Prediction accuracy: 4/4 = %100

Branch 3:

Actual branches: T-N-T-N-T-N
Predictions: N-T-N-T-N-T
Correctness (1 means correct, 0 means not correct): 0-0-0-0-0-0
Prediction accuracy: 0/6 = %0

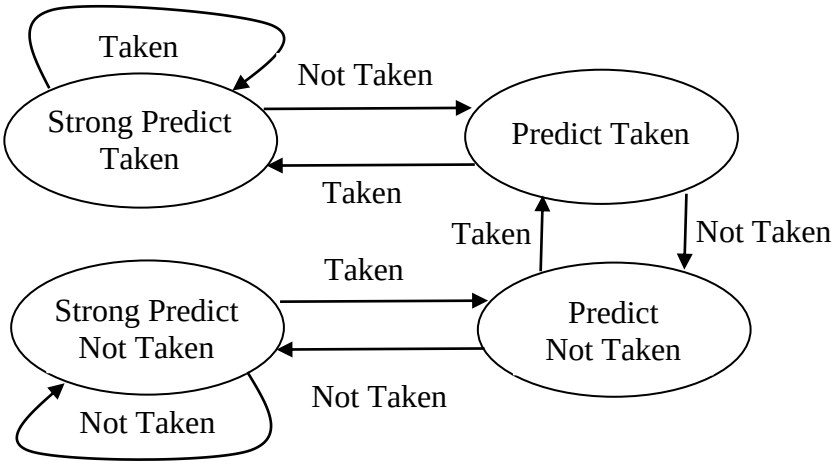
Branch 4:

Actual branches: T-T-T-N-T
Predictions: N-T-ST-ST-T
Correctness (1 means correct, 0 means not correct): 0- 1- 1-0-1
Prediction accuracy: 3/5 = %60

Branch 5:

Actual branches: T-T-N-T-T-N-T
Predictions: N-T-ST-T-ST-ST-T
Correctness (1 means correct, 0 means not correct): 0-1-0-1-1-0-1
Prediction accuracy: 4/7 = %57.14

In total we made 25 predictions and 10 of them were correct.
Thus, the total accuracy is 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**)

There are two data dependencies.

"sw" instruction depends on the "lw" instruction for the value in $t1.

"sw" instruction depends on the "add" instruction for the value in $t6.

Since there is no jump or branch instruction is given there is no control dependency.

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

lw instruction has this format: lw $rt, imm($rs). So, the data hazard has this format:

lw-sw: MEM/WB.rt = ID/EX.rs = $t1

sw instruction has this format: sw $rt, imm($rs). So, the data hazard has this format:

add-sw: EX/MEM.rd = ID/EX.rt = $t6

Since there is no forwarding mechanism, we need to wait 2 clock cycles.

```
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**)

If there is forwarding from ALU we remove the data hazard between lw-sw.
The only data hazard is this:
add-sw: EX/MEM.rd = ID/EX.rt = $t6

```
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**)

If there is full forwarding we remove all the data hazards, so we don't need any nop

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

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:
300 ps * 5 = 1500 ps = 1.5 ns
360 ps * 4 = 1440 ps = 1.44 ns
400 ps * 3 = 1200 ps = 1.2 ns