**Efe Şencan 25083**

## Homework #3

**Assigned**: 04/05/2021
**Due**: 12/05/2021

**Note**: Only hardcopy submissions are acceptable to the instructor during the lecture hour.

1. 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. (**25pts**)

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) (**5 pts**)

**Always taken**:
(0.40) * 1 + (0.20) * ((0.45 * 1) + 0.55 * (1 + 3)) + (0.40) * 1 = **1.33**

**Always not taken**:
(0.40) * 1 + (0.20) * ((0.55 * 1) + 0.45 * (1 + 3)) + (0.40) * 1 = **1.27**

**Dynamic predictors**:
(0.40) * 1 + (0.20) * ((0.60 * 1) + 0.40 * (1 + 3)) + (0.40) * 1 = **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**)

Suppose we have 100 instructions in total. The previous CPI of the dynamic branch predictor was 1.24. According to the new setup, we will have 60 (40 + 10*2) R type

instructions, 10 (20/2) branch instructions, and 40 of them would be the rest. Then the CPI of the new system is: 0.60 * 1 + 0.10 * ( 0.90 * 1 + 0.10 (1 + 3)) + 0.40 * 1 = **1.13**
**Speedup** = 1.24 / 1.13 = **1.09734**

2. 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

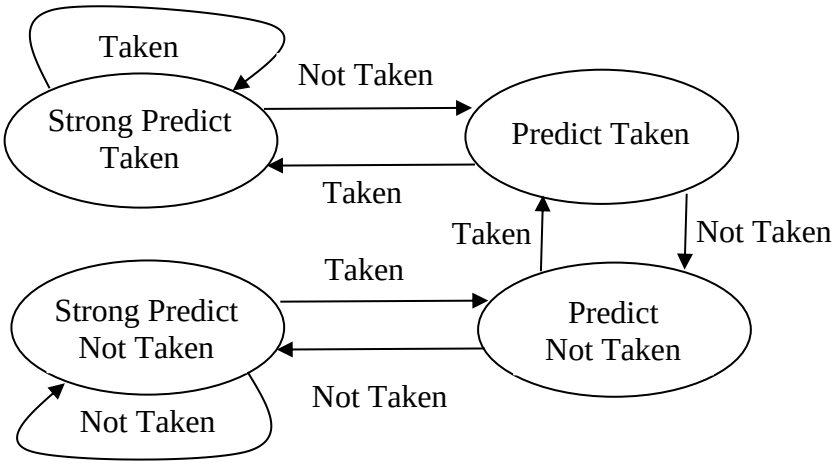
   For dynamic schemes, assume each branch has its own prediction buffer. List the predictions for the following branch prediction schemes for the execution of the code above. What are the total prediction accuracies? (**25 pts**)

   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 = **2/3**
Branch 2: N-N-N-N = 4/4 = **1**
Branch 3: N-T-N-T-N-T = 0/6 = **0**
Branch 4: N-T-T-T-N = **2/5**
Branch 5: N-T-T-N-T-T-N = **2/7**

   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. (**10 pts**)

Branch 1: N-T-T = **2/3**
Branch 2: N-N-N-N = 4/4 = **1**
Branch 3: N-T-N-T-N-T = 0/6 = **0**
Branch 4: N-T-T-T-T = **3/5**
Branch 5: N-T-T-T-T-T-T = **4/7**




3. Consider a processor whose pipeline depth is 20 (i.e. 20-stage pipeline) and issue width is 4 (i.e., four-issue processor). Consider also the following figures for branch instructions and branch prediction scheme adopted. (**25 pts**)

| Percentage of branch instructions of all executed instructions | Branch prediction accuracy | Branch outcome known in stage |
|---|---|---|
| 10% | 90% | 15 |

a. If there are no hazards (no structural hazard, no data dependency, and no branch mispredictions), what is the *ideal* performance improvement over a single-issue processor with the classical five-stage pipeline with the same ISA (i.e. ideal speedup)? Assume that the clock cycle time decreases in proportion to the number of pipeline stages. (**5 pts**)

Since clock cycle time decreases in proportion to the number of pipeline stages, we have 4 times improvement with 20-stage pipeline compared to the 5-stage pipeline. Since, there are also four-issue processor, we have 4 times improvement compared to the one-issue processor. Hence, we have 4*4 = **16** times performance improvement in total.

b. How many branch instructions can be "in progress" at most (i.e. already fetched from the memory but not completed yet) at any given time? (**10 pts**)

Since we have 4 issue processor, we can execute 4 instructions simultaneously. Assuming that each consecutive instruction is branch instruction and we have 20-stage pipeline, we can get at most 20 x 4 = **80** branch instructions that are in progress. I also assume that a branch instruction is in progress until it completes its 20 stage.

c. How many instructions are fetched from the wrong path for each branch mispredictions and need to be flushed? Calculate the effect of mispredictions on CPI (clock cycle per second). Assume that there is no other hazard. (**10 pts**)

Since the outcome of the branch is determined during the $15^{th}$ step, **14** instructions should be flushed for each branch misprediction.

Updated CPI = (0.90 * 1/4) + 0.1 * (0.90 * 1/4 + 0.10 * (1/4 + 14)) = **0.39**

4. 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:

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

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

x = 45.0

45 = 32 * (1 + 0.40625)
Sign = 0, exponent – bias = 5, since bias is 7, exponent = 12
exponent = 1100, fraction = 01101
**Floating point = 0 1100 01101**

y = 0.75
0.75 = ½(1 + 0.5)
Sign = 0, exponent – bias = -1, since bias is 7, exponent = 6
exponent = 0110, fraction = 10000
**Floating point = 0 0110 10000**

z = -44.0
44 = 32 * 1.375
Sign = 1, exponent – bias = 5, since bias is 7, exponent = 6
exponent = 1100, fraction = 01100
**Floating point = 1 1100 01100**

5. Consider a processor with the following parameters:

| | |
|---|---|
| Base CPI, no stalls due to cache misses | 0.5 |
| Processor speed | 2.5 GHz |
| Main memory access time | 100 ns |
| First-level cache miss rate per instruction | 3% |
| **1st option for the second-level cache** | |
| Direct-mapped: the latency | 20 cycles |
| Local miss rate of the second level cache, direct mapped | 50% |
| **2nd option for the second-level cache** | |
| 8-way set associative: the latency | 50 cycles |
| Local miss rate of the second level cache, 8-way set associative | 40% |

(**15 pts**)

a. We are considering two alternatives for second-level cache memory as shown above. What are the global miss rates if we use second-level direct-mapped cache (1st option) and second-level 8-way set-associative cache (2nd option)? (**5 pts**)

   Global miss rate if we use 1st option:
   3% * 50% = **1.5%**

   Global miss rate if we use 2nd option:
   3% * 40% = **1.2%**

b. Calculate the CPI for the processor in the table using: i) only first-level cache, ii) a second-level direct mapped cache, and iii) a second-level 8-way set associative cache. (**10 pts**)

   **Only first level cache:**
   base CPI = 0.5, first level cache miss rate per instruction = 0.03
   Miss penalty cycles = (100 / 0.4)
   CPI = 0.5 + 0.03 * (100 / 0.4) = **8**

   **Second level direct mapped cache:**
   base CPI = 0.5, first level cache miss rate per instruction = 0.03, direct mapped latency = 20cycles, Global miss rate = 0.015, Miss penalty cycles = (100 / 0.4)
   CPI = 0.5 + (0.03 * 20) + (0.015 * (100 / 0.4)) = **4.85**

   **Second level 8-way associative cache:**
   base CPI = 0.5, first level cache miss rate per instruction = 0.03, 8-way set associative latency = 50 cycles, Global miss rate = 0.015, Miss penalty cycles = (100 / 0.4)
   CPI = 0.5 + (0.03 * 50) + (0.012 * (100 / 0.4)) = **5**