# 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.

1. 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)

Delay due to mispredicted branches (branch penalty) is 3 clock cycles.

Extra CPI = 3 × 0.2 × 0.55 = 0.33 🡺 CPI = 1.33

Extra CPI = 3 × 0.2 × 0.45 = 0.27 🡺 CPI = 1.27

Extra CPI = 3 × 0.2 × 0.4 = 0.24 🡺 CPI = 1.24

1. 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)

Total clock cycle in the original code: IC×1.24

Total clock cycles in the new code:

CC = IC×1.1×1 + IC×0.1×0.1×3 = IC×(1.10+0.12) = IC × (1.13)

Speedup = 1.24/1.13 = 1.097

1. 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?

1. There is one 1-bit dynamic predictor for each branch, and each is initialized to “Predict Not Taken”.

Branch 1: - + +

Branch 2: + + + +

Branch 3: - - - - - -

Branch 4: - + + - -

Branch 5: - + - - + - +

Total: 11 correct, 14 incorrect prediction

Accuracy = 100×11/25 = 44%

1. 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.

Branch 1: - + +

Branch 2: + + + +

Branch 3: - - - - - -

Branch 4: - + + - +

Branch 5: - + - + + - +

Total: 13 correct, 12 incorrect prediction, Accuracy = 100×13/25 = 52%

Strong Predict Taken

Predict Taken

Strong Predict Not Taken

Predict   
Not Taken

Taken

Not Taken

Not Taken

Taken

Taken

Not Taken

Taken

Not Taken

1. 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.

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

1. 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.

We compute the time-per-instruction as CPI times the clock cycle time. For the 1-issue 5-stage processor, we have a CPI of 1 and a clock cycle time of . For an -issue -stage processor we have a CPI of and a clock cycle of . Overall, we get a speed-up of:

1. What is the expected number of branch instructions “in progress” (i.e. already fetched from the memory but not completed yet) at any given time?

The number of in-progress instructions is equal to the pipeline depth times the issue width. The number of in-progress branches can then be easily computed because we know what percentage of all instructions are branches. We have:

1. 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.

We keep fetching from the wrong path until the branch outcome is known, fetching 4 instructions per cycle. If the branch outcome is known in stage of the pipeline and it is a misprediction, all instructions in the previous − 1 stages are from the wrong path. Assuming that the branch is just as likely to be the 1st, 2nd, 3rd, and 4rd instruction fetched in its cycle, we have on average 1,5 instructions from the wrong path in the th stage (the branch instruction can be one of the issue slot with equal probability). We have then:

(15 − 1) × 4 + 1.5 = 57.5

CPI will increase by 57.5 ×0.1×0.1 = 0.575

1. 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 × (1.0 + significand) × 2exponent-bias

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

x = 45.0 = (101101.0)×20 = 25×(1.01101) = 212-7×(1.01101) 🡺 0 1100 01101

y = 0.75 = (0.11) ×20 = 2-1×(1.10000) = 26-7×(1.10000)🡺 0 0110 10000

z = -44.0 = -(101100.0)×20 = 25×(1.01100) = 212-7×(1.01100) 🡺 1 1100 01100

1. 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% |

1. 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)?

Global miss rate if we use 1st option: 3% × 50% = 0.03 × 0.5 = 0.015 🡺 1.5%

Global miss rate if we use 2nd option: 3% × 40% = 0.03 × 0.40 = 0.012 🡺 1.2%

1. 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.

Base CPI: 0.5

Memory miss cycles: 100 × 2.5 = 250 clock cycles

i) Total CPI: 0.5 + 250 × 3% = 8

ii) Total CPI: 0.5 + 20 × 3% + 250 × 1.5% = 4.85

iii) Total CPI: 0.5 + 50 × 3% + 250 × 1.2% = 2.0 + 1.25 + 5.4 = 5.0