COL216 Computer Architecture **Quiz 2** **17.03.2022**

Q 1. A processor is implemented as a 5 stage pipeline. On encountering a branch instruction, any inline instructions following it in the pipeline are flushed. The decision whether branch is to be taken and computation of the target address, both happen in the third stage. The next instruction is fetched after that. The processor shows an effective CPI of 1.4 for a specific benchmark. Find the percentage of instructions executed in the benchmark that are branch instructions. Assume that in 70% of branch instructions branch is taken. The implementation team is considering one of the following two enhancements. Find the performance speed up in each case.

a) Delayed branch with 1 delay slot is introduced for all branch instructions. Obviously, the instructions in the delay slots don't get flushed. Assume that for the benchmark under consideration, the compiler is able to fill up 80% of delay slots with useful instructions and remaining ones with no-ops.

b) Branch Target Buffer (BTB) is introduced which is looked up in the fetch cycle itself. If there is a BTB hit (assume hit rate of 80%), the target instruction is fetched in the next cycle from the address given by BTB. If there is a BTB miss, the processor continues with inline instructions (no flushing). When the branch instruction reaches the third stage, it is revealed whether the next instruction fetched is correct or not (assume that it is correct 90% of the time when there is a BTB hit and 80% when there is a BTB miss). Correct next instruction is fetched in the following cycle if required.

Solution:

Since the inline instructions after a branch instruction are flushed, the penalty is 2 cycles, both in case of branch taken and branch not taken.

$CPI_{effective} = CPI_{no\_hazards}$ + branch_percentage * branch_penalty
1.4 = 1 + branch_percentage * 2 => branch_percentage = **0.2 or 20%**

[2 marks]

a) With delayed branch, for 20% of branch instructions, an extra instruction (no-op) is executed because compiler is not able to fill all the delay slots with useful instructions. Effect of no-ops can be accounted for in computation of branch penalty. We consider penalty reduction from 2 cycles to 1 cycle when a delay slot is filled with a useful instruction and no reduction when it is filled with a no-op. The penalty is same both in case of branch taken and branch not taken.

[2 marks]

$CPI_{delayed\_branch} = CPI_{no\_hazards}$ + branch_percentage * $branch\_penalty_{delayed\_branch}$
= 1 + 0.2 * (0.8 * 1 + 0.2 * 2) = 1 + 0.2 * (1.2) = 1.24

[1 mark]

$Speedup_{delayed\_branch} = CPI_{effective} / CPI_{delayed\_branch}$
= 1.4 / 1.24 = **1.13**

[1 mark]

b) There are 4 possibilities. Penalties and probabilities for these four are as follows.

BTB hit	correct instruction fetched	Penalty = 0	Probability = 0.8 * 0.9 = 0.72
	incorrect instruction fetched	Penalty = 2	Probability = 0.8 * 0.1 = 0.08
BTB miss	correct instruction fetched	Penalty = 0	Probability = 0.2 * 0.8 = 0.16
	incorrect instruction fetched	Penalty = 2	Probability = 0.2 * 0.2 = 0.04

Overall average $branch\_penalty_{BTB}$ = 2 * 0.08 + 2 * 0.04 = 0.24

[2 marks]

$CPI_{BTB} = CPI_{no\_hazards}$ + branch_percentage * $branch\_penalty_{BTB}$ = 1 + 0.2 * 0.24 = 1.048

[1 mark]

Since the number of instructions executed does not change with BTB,
$Speedup_{BTB} = CPI_{effective} / CPI_{BTB} = 1.4 / 1.048 =$ **1.36**

[1 mark]

COL216 Computer Architecture **Quiz 2** **17.03.2022**

Q 2. Compare read miss penalty and write miss penalty for 3 different write policies for cache - WB, WTWA and WTNWA. Assume that the memory interface requires n_1 cycles to send address, n_2 cycles to read/write the first word after sending address and n_3 cycles to read/write each of the subsequent words. Wherever applicable, assume that the probability of a block being dirty at the time of its eviction is p. Do the comparison for the following two read/load policies.

a) Sequential read, no data forwarding, no wrap-around load
b) Concurrent read, data forwarding, wrap-around load

Solution:

Assumptions:
- Let cache read time = cache write time = t_{cache} (OK if this is taken as 1)
- Let b = number of words in a block.
- Let there be no write buffers.

Finding memory transfer times:
Word transfer time (memory to cache or cache to memory) = $t_{word} = n_1 + n_2$
Block transfer time (memory to cache or cache to memory) = $t_{block} = n_1 + n_2 + b * n_3$

[2 marks]

(a) Sequential read, No data forwarding, No wrap-around load

WB policy
Read miss: transfer block to be displaced from cache to memory (if dirty), transfer missing block from memory to cache and read from cache.
Read miss penalty = $p * t_{block} + t_{block} + t_{cache} = (p + 1) * (n_1 + n_2 + b * n_3) + t_{cache}$

Write miss: transfer block to be displaced from cache to memory (if dirty), transfer missing block from memory to cache and write into cache.
Write miss penalty = $p * t_{block} + t_{block} + t_{cache} = (p + 1) * (n_1 + n_2 + b * n_3) + t_{cache}$

[2 marks]

WTWA policy
Read miss: transfer missing block from memory to cache and read from cache.
Read miss penalty = $t_{block} + t_{cache} = (n_1 + n_2 + b * n_3) + t_{cache}$

Write miss: transfer missing block from memory to cache, write into cache and transfer a word to memory.
Write miss penalty = $t_{block} + t_{cache} + t_{word} = (2n_1 + 2n_2 + b * n_3) + t_{cache}$

[2 marks]

WTNWA policy
Read miss: transfer missing block from memory to cache and read from cache.
Read miss penalty = $t_{block} + t_{cache} = (n_1 + n_2 + b * n_3) + t_{cache}$

Write miss: transfer a word to memory.
Write miss penalty = $t_{word} = n_1 + n_2$

[2 marks]

Note: The above expressions for miss penalty are actually for time spent after the initial cache access that determines cache hit or miss. Since t_{word} time is spent even in case of write hit for WTWA and WTNWA policies, this may be considered as a part of write hit time and excluded from write miss penalty for these policies. If write buffers are considered to be present, writing operations can take

place in the background and the processor does not have to wait for completing these operations. this implies that all terms relating to writing into memory (i.e. t_{word} for WTWA/WTNWA and $p*t_{block}$ for WB) get dropped.

(b) Concurrent read, Data forwarding, Wrap-around load

Wrap-around load implies that the processor does not wait for the entire block to be loaded, it just waits for one word (the missing one) to be loaded, reducing t_{block} to t_{word}. Secondly, the term t_{cache} added after t_{block} goes away because of data forwarding. Thirdly, concurrent read implies that reading from memory begins without waiting for hit/miss determination, saving another t_{cache} time. Effect of all this is that $t_{block} + t_{cache}$ is replaced by $t_{word} - t_{cache}$ in all cases except write miss penalty for WTNWA.

WB policy
Read miss penalty = $p * t_{block} + t_{word} - t_{cache} = (p + 1) * (n_1 + n_2) + p * b * n_3 - t_{cache}$
Write miss penalty = $p * t_{block} + t_{word} - t_{cache} = (p + 1) * (n_1 + n_2) + p * b * n_3 - t_{cache}$

WTWA policy
Read miss penalty = $t_{word} - t_{cache} = (n_1 + n_2) - t_{cache}$
Write miss penalty = $t_{word} - t_{cache} + t_{word} = 2 * (n_1 + n_2) - t_{cache}$

WTNWA policy
Read miss penalty = $t_{word} - t_{cache} = (n_1 + n_2) - t_{cache}$
Write miss penalty = $t_{word} = n_1 + n_2$

[2 marks]

Note: For WB policy, if a dirty block is to be written into memory, this would need to be done before reading the missing block from memory unless the block to be written back can be placed somewhere temporarily. Such an arrangement is required in order to avail the benefit of concurrent read in case of WB policy.