Bonus2 ILP, IS2202

Linghan Zhao
020717-5563
linghanz@kth.se

February 14, 2024

Problem 1. Dependences and Hazards

a) My personal values: $Y = 0, y = 2, M = 0, m = 7, D = 1, d = 7$, and the resulting register $\$tD = \$t1$.
b) There is only one dependence involving register \$t1, the two instructions are
$i1: \quad lw \quad \$t1, 0(\$s0) \quad\quad \# \quad \$t1 <- \quad memory(\$s0)$
$i3: \quad add \quad \$t3, \$t1, \$t2 \quad \# \quad \$t3 <- \quad \$t1 + \$t2$
For this instruction pair, the kind of dependence is data dependence, which is RAW-Read after Write. That means the register \$t1 needs to be written first in instruction `i1`, then be read in instruction `i3`.
c) There are two dependences involving register \$t5.
Pair1: data dependence
$i6: \quad lw \quad \$t5, 12(\$s0) \quad\quad \# \quad \$t5 <- \quad memory(12 + \$s0)$
$i7: \quad addi \quad \$t6, \$t5, 212 \quad \# \quad \$t6 <- \quad \$t5 + 212$
Pair2: data dependence
$i6: \quad lw \quad \$t5, 12(\$s0) \quad\quad \# \quad \$t5 <- \quad memory(12 + \$s0)$
$i8: \quad beq \quad \$t5, \$t4, Li11 \quad\quad \# \quad if \quad \$t5 == \$t4 \quad goto \quad Li11$
Both of them are data dependences, because register \$t5 should be Read after Write.
~~**d)**~~ No, there will not be a hazard that requires stalling.
Because the instruction `i9:  nop`, which is an unused branch delay slot, being executed regardless of whether the branch condition in `i8` is true or not, so that the instructions in the pipelines can be executed normally.
If there doesn't exit `i9:  nop`, a solution for processor is stalling, waiting for the instruction's execution in pipelines, to make sure the following instructions not being executed incorrectly before the result of branch instruction.
e) As instructions are executed in-order in the simple in-order 5-stage pipeline, when it comes to Write stage, all Read operations have been completed in former stages. So WAR hazard cannot occur.
And all Write operations are executed in sequence, so there cannot be WAW hazard, in which two instructions want to write to a same register at the same time.
f) A structural hazard happens when multi-instructions request a resource simultaneously, other instructions need to wait because of limited hardware resources.

Problem 2. Static and Dynamic Scheduling

$P = 2$

a) It can execute 2 instructions simultaneously.

cycle1	cycle2	cycle3	cycle4	cycle5	cycle6	cycle7
i1, i2	i3	i4	i5, i6	i7, i8	i9, i10	Li11

Table 1: unmodified code snippet execution cycle

b) i1, i2, i3, i6, i4, i7, i5, i8, i9, i10, Li11

c)

cycle1	cycle2	cycle3	cycle4	cycle5	cycle6
i1, i2	i3, i6	i4, i7	i5, i8	i9, i10	Li11

Table 2: out-of-order execution cycle

Problem 3. Branch prediction

a) The main advantage is that 2-bit predictor can make more accurate prediction, because it can maintain in a state which needs two mispredictions before changing the prediction. While 1-bit predictor will make more mistakes when alternate between taken and not-taken frequently.

b)

$$H = M + 1 = 1(bit) \tag{1}$$

$$E = 16 \times 2^d = 2^{11} \tag{2}$$

$$H \times E = 2^{11} = 2048(bits) \tag{3}$$

So **2048** bits are required to implement a 1-bit branch-prediction table with 2^{11} entries.

c) There are 2^{11} entries, so there needs 11 bits to select the predictor in the prediction table. And because it is a processor with 32-bit addresses and a byte-addressed memory, so the least two significant bits of the program-counter are used as the byte-offset. Hence, the bit-2 to bit-12 are used as the input to the prediction table.

d) Considering a correctly predicted branch, the 32 bits PC as input, then the memory tells us if it is a branch(valid) and the prediction bits(taken/not-taken), finally output a 32-bit address of the branch target address if it is taken. When the branch is resolved, update the correct prediction with the 1-bit status.

If the branch is not predicted to be taken, update PC to next value.

Here, 32 input bits, 32 output bits, and 1 status bit.

~~**e)**~~ Branch folding is implemented in the BTB by merging multiple branch instructions which has the same target address, into a single BTB entry.

When a new branch instruction comes, first check whether an entry with the same target address already exists in the BTB. If it already exists, merge the new branch instruction with the existing entry; if it does not exist, create a new entry with the branch instruction as its only member.

It can represent multiple branch instructions in one entry term, which saves the storage.

3.e)Instead of storing predicted target address, the BTB with branch folding will store the target instruc tion(s). If the buffer signals a hit when it is unconditional, the processor will simply replace current loaded instruction with the instruction from the branch-target buffer. So the branch instruction works as if it takes 0-cycle.

And it obtains instructions at the branch target faster, and multiple instructions at the branch target can be provided at one time.

Problem 4. Architecture and Microarchitecture

a) Stage 5(Execute 1) would be the earliest possible stage where the misprediction could be detected if a branch is mispredicted.
Because branch misprediction usually cause instructions incorrectly executed in pipeline, and the execution stage is the earliest stage that can detect if the execution result matches the prediction.
When misprediction detected, instruction results must be canceled.
b) Every 10th instruction is a branch, so 10% possibility of a fetched instruction to be a branch;
90% of branches are correctly predicted, so 10% possibility of a branch to be mispredicted;
Every clock cycle fetching two instructions, and detecting at stage 5, so there are 8 instructions been fetched before stage 5.
So the persentage of fetched instructions will be discarded because of misprediction is $10\% \times 10\% \times 8 = 8\%$.
c) Stage 5(Execute 1) would be the earliest possible stage where the page-fault could be detected.
Because a Load instruction causes a page-fault on data fetch, when mismapping the corresponding physical memory as the computation of the target memory address in the execution stage.
d) Using Reorder Buffer to record the out-of-order execution's order of instructions before commitment. When an exception occurs, the processor cancels previously executed instructions by checking the buffer to ensure the exception handlers executing in correct order. And the exception and all instructions after that will not be executed.
e) Normally, the location of the 2nd input operand could be accessed directly from memory as in the accumulator-based machines.

References

Computer Architecture: A quantitative approach, 6th ed. by John Hennessy and David Patterson Morgan Kaufmann, 2017 ISBN: 978-0-12-811905-1.