CSE 240A Homework Two

November 7, 2012

*Only Problem Set Two will be graded. Turn in only Problem Set Two before November 21, 2012, 11:00 AM.*

## 1 Problem Set One

- Hennessy & Patterson (5th Ed) 3.1
- Hennessy & Patterson (5th Ed) 3.3
- Hennessy & Patterson (5th Ed) 3.5
- Hennessy & Patterson (5th Ed) 3.16
- Hennessy & Patterson (5th Ed) 3.17
- Hennessy & Patterson (5th Ed) 3.19

## 2 Problem Set Two

1 (Exceptional Scoreboards)

From Homework 1, Question 3
A custom processor company is utilizing Tomasulo’s processor with register renaming as a backbone for its custom solutions. Given a particular application, as defined by its program, the company customizes the Tomasulo processor by analyzing the program and incorporating just the right number of functional units, reservation stations, and physical registers in order to enable it to operate in full speed at steady state while ensuring no resource is wasted by lying idle in steady state. The execution latencies for integer units, memory units, FP adders, and FP multipliers are 1, 2, 2 and 4, respectively. Assume that an unlimited number of simultaneous memory accesses are allowed.

Consider the following three loops. Each one of them reads values, manipulates values, and subsequently stores them into various location(s). For each of the loops below, first, identify the instruction schedules at steady state which fully utilize the given reservation stations, and also identify the minimum number of functional units, reservation stations and physical FP registers for these cases. Assume that no branch mispredictions, no exceptions and no cache misses arise while each loop is executing.

(A) LOOP:

```
L.D       F0, 0(R1)
ADD.D     F1, F0, F0
S.D       F1, 100(R1)
DADDIU    R1, R1, -8
BNZ       R1, LOOP; branches if R1 is not zero
```
(B)
LOOP:
L.D F0, 0(R1)
L.D F1, 100(R1)
ADD.D F2, F0, F1
S.D F2, 0(R2)
ADD.D F3, F2, 50
S.D F3, 0(R3)
DADDIU R1, R1, -8
DADDIU R2, R2, 8
DADDIU R3, R3, 16
BNZ R1, LOOP; branches if R1 is not zero

(C)
LOOP:
L.D F0, 0(R1)
L.D F1, 100(R1)
MULT.D F2, F0, F1
S.D F2, 0(R2)
S.D F1, 0(R3)
DADDIU R1, R1, -8
DADDIU R2, R2, 8
DADDIU R3, R3, 16
BNZ R1, LOOP; branches if R1 is not zero
(Part A) For the speculative Tomasulo with reorder buffer and register renaming, one of the concerns is the number of storage elements it requires. In addition to storage for physical registers (and possibly architectural registers as well), even reservation stations need to capture the real values and hold them until all the operands have become ready. Furthermore, the ROB keeps holding the final computed value up until the entry has been committed and the instruction is graduated. One company is exploring ways to cut down on this cost by having the reservation stations no longer internally capture the values as they become ready and instead let them go to the register space (in this case the physical renamed registers) and retrieve them only once both of the values have arrived through the CDB. Of course, the physical registers cannot retain the values forever, so some kind of physical register deallocation policy (and consequently, reuse of the deallocated registers) is necessary. This vantage point has given rise to a heated debate within the company wherein a number of prominent engineers are duking it out. Some are concerned that this physical register deallocation (and reuse) will now give rise to all kinds of name hazards through the physical registers, while others are claiming that name hazards are only applicable in the case of architectural registers and are either of no concern or impossible to occur in the case of physical registers. Others are claiming that WAR hazards need to be considered, while yet others claim that the only concern that may arise is WAW hazards. Which one do you agree with and why?
(Part B) In the meantime, with all this heated debate going on, the CEO of the company, in the infinite wisdom of any CEO worth their salt, has decided to leapfrog and examine the possible implications ahead of time, in case it turns out indeed that there is reason for concern for name dependencies. (S)he has tasked a number of leading engineers in the company to see how name hazards can be handled in case they turn out to indeed be a nuisance. The engineering group has opened up their books from graduate school days and noticed that an oft-mentioned deallocation policy is to deallocate physical registers when the mapped architectural register has been rewritten. Now they are struggling to figure out what kinds of hazards may end up being encountered as a result of this deallocation policy. Some are claiming that there can be no name hazards as a result of this policy, while others claim that the only hazards that will exist will be a subset of the hazards already embedded in the architectural names. Others are claiming that there can be hazards which did not exist in the architectural registers. Another subgroup is fighting it out trying to decide which hazards one needs to worry about, the architectural register, the physical register, both or neither. Please provide your opinion as to which of these positions you agree with and provide a reasoning, perhaps complemented by code fragment and a possible reasoning to illustrate it.

(Part C) While this debate is raging on, the CEO is really interested in getting ahead of the game and has tasked a group to prepare a technique to solve any WAR hazards. Should they occur, what modifications, should one do to the pipelining in order to implement this WAR hazard obviation? Which fields should be added possibly to reorder buffer entries, reservation stations or maybe even to physical registers? Based on some of these fields, what conditions should be checked in order to ensure an orderly progression of the pipeline? (Of course, you realize quickly that instead of generating new techniques and approaches from scratch in limited time, it may be appropriate to see how techniques that you learned about during your 240A course may actually be adopted for this problem your CEO has tasked you with.)
(Part D) Continuing on in the inexorable quest for getting ahead of the game, the CEO has tasked another group to go take a look at how to handle output dependencies under this deallocation policy. Some engineers are claiming that WAW will never happen under the outlined scenario, while others claim that it can happen even more frequently and that therefore resorting to the good old scoreboard technique of simply stopping the pipeline is inadvisable. Others are claiming that stopping the pipeline is not only inadvisable but also that it will lead to deadlock with the pipeline freezing indefinitely possibly. Yet others are claiming that the only hope is to adopt the scoreboard technique verbatim and that allowing WAW hazards to proceed under this deallocation policy would wreak havoc. Please describe which of these positions you agree with and illustrate your answers with perhaps a short code fragment and a possible evolution of its fortunes through the pipeline.

(Part E) While this debate is going on, a group of Young Turks in the company is interested in staging a coup d'état and is proposing a completely different scheme, wherein the register deallocation is only to be effected once all the preceding reading registers have actually captured their values in the reservation stations that are starting to chug along. They claim that this will resolve all concerns regarding name hazards (albeit at the expense of increasing the pressure on the registers and perhaps necessitating a few extra physical registers to keep it chugging). As word of the possibly impending coup has spread in the company, the establishment in fighting back, spreading rumors regarding the workability of this technique. Depending on who is listening, the establishment claims that this technique is simply unworkable or that while workable it would necessitate so much extra information (they are mumbling something about counters to keep track of outstanding reads), or that it simply will not solve any of the name hazards anyway and so on. The Young Turks, hearing of this counterattack, and in a paroxysm of messianic revolutionary fervor are surreptitiously posting various pithy fragments, such as “Freedom or Liberty”, “The answer, my friend, is blowin’ in the wind”, and perhaps the most damning “KISS (Keep It Simple, Stupid)”. Do you think the Young Turks have a point? Can such a scheme work without necessitating the keeping of counters per physical register? If so, how? How does one deallocate registers then? Are there any WAR hazards therein? Any WAW hazards to worry about?
4 (Branch Predictor Latency)

An important issue in current highly accurate branch predictors is that most of the predictors suffer from non-trivial amounts of latency, which decreases the improvement of performance enabled by high accuracy. This question asks you to explore the accuracy-latency trade-off for the most advanced branch prediction schemes.

(Part A) Consider a 6 stage pipeline, \textbf{IF1 IF2 ID EXE MEM WB}, in which the instruction fetch is split into two stages, IF1 and IF2. An aggressive branch predictor is accessed during the IF1 stage. However, because of the long latency, the prediction is not available until the \textbf{ID} stage. Therefore, the fetch continues from the \textbf{fall through path} until the \textbf{ID} stage. Then the processor will choose the path predicted by the branch predictor. During the \textbf{EXE} stage, the branch condition is definitively resolved. Assume that all the branch instructions encountered are guaranteed to be found in the Branch Target Buffer (BTB), which is accessed in the \textbf{IF1} stage. The prediction accuracy of the aggressive branch predictor is 90\%, and 60\% of all the branches in the benchmark are \textbf{taken}. Please compute the average number of stall cycles per branch.

(Part B) In your design team, one engineer suggests that before the prediction of the aggressive branch predictor is available, instead of just fetching from the fall through path, a simple predictor with reduced accuracy but less latency can be used to generate a prediction. Therefore, he proposes to use a hybrid prediction scheme: for each branch, the prediction is first given by the simple predictor which can generate a result in the \textbf{IF1} stage. Then the subsequent instructions following the path of this branch are fetched in the next cycle. Two cycles later, in the \textbf{ID} stage, a new prediction is also given by the aggressive predictor for the \textbf{same} branch. If this new prediction disagrees with the prediction given by the simple predictor, the path following the new prediction is chosen and the previous fetched path is canceled. Assume that the simple predictor has a prediction accuracy of 80\%, while the accuracy of the aggressive predictor is still 90\%. Please compute the average number of stall cycles per branch.

(Part C) In the case that only the simple predictor is used, what is the average number of stall cycles per branch? In comparing this result with the result you get from (Part B), which branch predictor is better, the simple predictor or the hybrid predictor?
5 (Branch Predictor Accuracy)

Consider the code segment below, which represents a loop containing 3 branch instructions in its body. Both the distance between the first two branches and the distance between the last two branches are 24 (word address). Assume this code segment is to be executed exclusively, and the loop branch is handled separately and does not use the branch predictor.

for j = 0 to 7
    ...;
    if (j mod 2 == 0) { // Branch1
        C = 1;
        ...
    }
    if (j mod 4 == 0) { // Branch2
        C = 0;
        ...
    }
    if (C == 1) { // Branch3
        ...
    }
end for

(Part A) An architecture with a 2-bit saturating counter branch predictor is being considered for this problem. The predictor has 16 entries and is accessed with the least significant bits of the branch address (word address). Please comment on the usefulness of using such a predictor with respect specifically to the prediction accuracy for the code above. Hint: your answer should consider the correlations between these three branches.

(Part B) Since there are some strong correlations between these three branches, a correlating predictor is proposed to be used for this code segment. This correlating predictor is accessed with the global history information. If we want to achieve the maximal prediction accuracy for Branch3 with the minimal amount of hardware resources, what is the minimal number of history bits that should be used? Please clearly show your analysis.

(Part C) One important issue in branch prediction is aliasing, i.e., two different branches map to the same entry in the branch predictor. In most cases aliasing will pollute the branch predictor and reduce the prediction accuracy significantly. For the correlating predictor with your own configuration in (Part B), will aliasing be a problem or not, in respect to the prediction accuracy for all 3 branches? If you think aliasing is not a problem, give a brief explanation. Otherwise, what is the minimal number of history bits that should be used to avoid aliasing? Explain your answer clearly.