

## Midterm Exam 2

### ECE 741 – Advanced Computer Architecture, Spring 2009

Instructor: Onur Mutlu

TAs: Michael Papamichael, Theodoros Strigkos, Evangelos Vlachos

April 22, 2009

### EXAM 2 SOLUTIONS

<b>Problem</b>	<b>Points</b>
<b>1</b>	60
<b>2</b>	30
<b>3</b>	20
<b>4</b>	20
<b>5</b>	10
<b>6</b>	15
<b>7</b>	20
<b>8</b>	25
<b>Total</b>	200

- This is a closed book midterm. You are allowed to have only 3 letter-sized cheat sheets.
- No electronic devices may be used.
- This exam lasts 1 hour 50 minutes.
- If you make a mess, clearly indicate your final answer.
- For questions requiring brief answers, please provide brief answers. Do not write an essay. You can be penalized for verbosity.
- Please show your work. We cannot give you partial credit if you do not clearly show how you arrive at an answer.
- Please do not write on the back of the sheets (if printed in simplex).
- **Please write your name on *every* sheet.**

### Problem 1 (Short answers) [60 Points]

Please be concise, clear, and to-the-point in all your answers.

#### i) DRAM Controllers [7 points]

Some recent chip designs, such as Sun Niagara and AMD Athlon 64 have included the DRAM controller on-chip (i.e., on the processor die) as opposed to off-chip (i.e., in the chipset). Briefly describe two advantages of this design decision. Explain the reasoning clearly for each advantage.

1. Higher bandwidth between memory controller and core  $\Rightarrow$  More information about memory requests available to controller (e.g., criticality  $\Rightarrow$  better scheduling and prioritization decisions)
2. Reduces latency between processor and memory controller  $\Rightarrow$  faster memory access

Briefly describe one disadvantage of this design decision. Explain your reasoning clearly.

Ties processor to specific DRAMs (cannot use any other DRAM by simply changing the memory controller). Adds complexity on-chip. Likely increases on-chip power consumption.

#### ii) In-order vs. out-of-order [4 points]

In an in-order processor instructions are dispatched in **control-flow** order.

In an out-of-order processor instructions are dispatched in **dataflow** order.

(Note: the answer should be more sophisticated than in and out-of, which is not grammatically correct anyway.)

#### iii) Caching instructions [2 points]

In an instruction cache, an instruction can be stored in **one** location(s).

In a trace cache, an instruction can be stored in **multiple** location(s).

#### iv) Execution Models [4 points]

Finish the following sentence: The DAE (Decoupled Access/Execute) paradigm provides an opportunity for performance improvement over the VLIW paradigm because

It does not perform lockstep execution (In DAE, even if one engine stalls, the other one can perform some useful work. In VLIW, if one instruction stalls, every instruction in the bundle must stall)

#### v) SRAM vs. DRAM [2 points]

The main element of storage required to store a single bit of information depends on whether we are talking about DRAM cells or SRAM cells.

For DRAM cells it is:

A capacitor

For SRAM cells it is:

Cross-coupled inverters (4 transistors)

**vi) Prefetching [6 points]**

Which of the following prefetching methods cannot prefetch compulsory misses? Circle as many as that apply.

<b>1. Markov prefetcher</b>
-----------------------------

- |  |
|--|
| 2. Runahead execution<br>3. Stream buffers<br>4. Content directed prefetching<br>5. Software based pre-execution |
|--|

Why? Explain very briefly.

Markov prefetchers need to see a miss before they can prefetch it (they must see the correlation between miss A and miss B, so that they can prefetch B, when they see another miss to A). All other prefetchers can prefetch a never-before-seen address.
--

**vii) Vector processing [4 points]**

```
for (i=1; i < 500; i++)
    A[i] = (B[i] + A[i-1])/7919;
```

Is the above loop vectorizable? Circle one:      YES       NO

Explain why/why not (10 words should be enough).

Dependence between A[ i ] and A[ i-1 ], which is calculated in previous iteration.
--

**viii) Runahead execution [8 points]**

Suppose a branch is fetched in “runahead mode” of a runahead execution processor. Suppose we know “magically” whether or not the branch is mispredicted. The below questions deal with whether continuing runahead execution is always useless after the fetch of this branch.

Is runahead execution always useless after a mispredicted branch that does not depend on an L2 cache miss? Circle one:      YES       NO.      Why/Why not?

It may still encounter “useful” misses because execution on the wrong-path can provide useful prefetching. (Same example as below) Besides, the mispredicted branch can be resolved during runahead mode and then runahead execution will continue on the correct path.
---

Is runahead execution always useless after a mispredicted branch that depends on an L2 cache miss? Circle one:      YES       NO.      Why/Why not?

It may still encounter “useful” misses because execution on the wrong-path can provide useful prefetching. Example:
---

Miss A

```
If (condition) use1(B) else use2(B); // Miss A causes Runahead;
mispredicted branch condition dependent on Miss A; B misses during
Runahead;
```



## Problem 2 (Branch Prediction) [30 Points]

The first two-level branch predictor was published in Micro-24 in 1991. Since then, it has seen many variations. Each variation has been proposed to provide some additional benefit. Four such variations are described below.

For each variation, describe the intended additional benefit over the baseline two-level predictor that was developed in 1991. It is important to be explicit in your answers without being unnecessarily wordy.

- i) Instead of using the global history register to index into the pattern history table, use the XOR of the global history register combined with some bits of the branch's address.

1. Reduces negative interference of different branches in the pattern history table → better PHT utilization, higher accuracy
2. Provides more context information to predict a branch → higher accuracy

- ii) Combine the two-level predictor with a 2-bit counter or a last time taken predictor.

This is a hybrid predictor, and as a result provides all benefits of hybrid prediction:

1. Enables faster warmup of the branch predictor → a prediction can be made by the simpler 2-bit-counter predictor before the global two-level predictor is warmed up.
2. Provides better predictability → a biased branch can be better predictable with a predictor that does not take into account global history

- iii) Do not increment/decrement the 2-bit counter in the pattern history register on the basis of T/NT. Instead store a direction bit in the tag store entry for each branch, and increment/decrement on the basis of whether the prediction agrees or does not agree with the stored direction bit.

This is the concept of agree prediction. Reduces negative interference in the pattern history table → improved prediction accuracy.

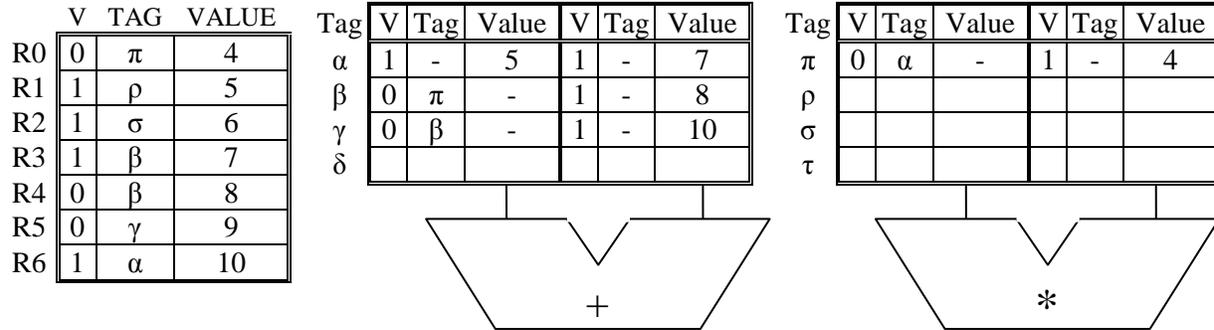
- iv) Instead of using the directions of the last several branches leading up to a branch as the index into the pattern history table, combine bits of the addresses of each of the intervening branches into an index.

This is the concept of using “path history.”

1. Reduces interference/aliasing of different branches in the global history register → better identification of correlated branches → better accuracy.
2. Provides better context information leading up to the current branch → higher accuracy

**Problem 3 (Out-of-order Execution) [20 Points]**

Initially, the register file of the Tomasulo-like machine shown below contains valid data in all registers. Four instructions are then fetched, decoded and issued in program order, none are executed. At that point the register file and reservation stations are as shown below:



Note: Some TAG information contained in the register file is left over from earlier instructions.

- a. Assuming no additional instructions are fetched, decoded, issued, fill in the state of the register file after the four instructions execute. [10 points]

	V	TAG	VALUE	Comments
R0	1	$\pi$	48	All V tags must be set at the end
R1	1	$\rho$	5	Was valid – NO change
R2	1	$\sigma$	6	Was valid – NO change
R3	1	$\beta$	7	Was valid – NO change
R4	1	$\beta$	56	
R5	1	$\gamma$	66	
R6	1	$\alpha$	10	Was valid – NO change

- b. Using only instructions of the form ADD  $R_d, R_s, R_t$  and MUL  $R_d, R_s, R_t$ , where  $R_d$  is the destination register and  $R_s$  and  $R_t$  are the source registers, show the four instructions in program order. [10 points]

1	ADD R5, R1, R3
2	MUL R0, R5, R0
3	ADD R4, R0, R4
4	ADD R5, R4, R6

Note for the destination of the first instruction (R5). It must be one of the registers that have  $V = 0$ , as all registers with  $V = 1$  are valid after the instructions are issued, but before they are executed. It cannot be R0, as the MUL instruction afterwards sources it and gets a valid value 4. Also, it cannot be R4, as it is sourced normally by the second ADD (the third instruction).

### Problem 4 (The Magic Processor) [20 Points]

We have a 16 wide issue microprocessor and (magically) a perfect branch predictor (100% accuracy) and perfect caches (100% hit rate). Everything is working correctly on the chip (i.e. no bugs).

- a. We measure the IPC on one application and it is less than 16. How is this possible? Explain. [4 points]

Data dependencies. The program simply does not have enough parallelism to exploit the wide issue of this processor. Also long-latency instructions (see b)

- b. Can the IPC be less than 1? Explain. If yes, give an example piece of code (in pseudo-assembly language) that will produce this. If no, why not? [8 points]

Long-latency instructions, such as FPMUL, forming a dependence chain.

```
FPMUL R1, R0, R0 // R1 is destination
FPMUL R2, R1, R1
FPMUL R3, R2, R2
...
```

In this example, we can easily see that the IPC will be  $1 / \text{FPMUL latency}$ .

Note: We did not define the latency of a cache hit. Therefore if one assumes that a hit takes more than one cycle, the same logic will apply.

- c. The pipeline consists of 2 stages. If you could redesign the chip, would you change the number of stages? Why or why not? Explain clearly. [8 points]

Yes, I would. I would increase the number of pipeline stages to increase frequency. Since branch mispredictions and cache misses are eliminated, filling the pipeline and keeping the pipeline busy is not a problem.

*More explanation (not required for the purposes of the exam): The main gain of pipelining is increased frequency; by splitting the work into multiple stages, we can clock the machine faster, as each stage has less work to do. The main problem of pipelining is cache misses (long latency stalls) and mispredictions (waste a lot of work). This machine does not suffer from these problems, so the number of stages can be increased.*

*While the dependence chains mentioned in b will become a problem for very deep pipelines, the current number of stages is so low, that we can safely increase the pipeline depth.*

*Note: A lot of students responded that the optimal depth is 16, as it is the width of the machine. This is irrelevant and incorrect. We make a processor wider to increase IPC (instructions executed in a single cycle); deeper to increase frequency (cycles per second) by reducing the amount of work per stage.*

### Problem 5 (Instruction caching) [10 Points]

The Intel Pentium processor had a split-line first-level instruction cache. Why was that a good idea? Be brief, please, but specific.

Split-line cache: if we request an instruction in the second half of line K, we'll get the first half of line K+1 as well.

The Pentium processor is superscalar, so it needs wide fetch to supply its execution engine. Given this (and exacerbated by the variable instruction length of the x86 ISA) it is uncertain whether enough instructions exist in the second half of a cache line, so the cache supplies the first half of the next line automatically. This way the processor experiences fewer fetch breaks.

Therefore, it is a good idea.

### Problem 6 (VLIW) [15 Points]

- a. An IA 64 instruction bundle consists of three 41-bit instructions, packaged in a 128-bit unit. What are the extra five bits used for? What value do they provide over previous VLIW designs? [10 points]

These template bits are used to describe which instructions within a "bundle" are independent vs dependent, i.e. they delimit independent instructions. This gets rid of the lock-step execution property of traditional VLIW designs.

- b. What are the disadvantages of this approach compared to traditional VLIW? [5 points]

Hardware is more complex, as it now has to check which instructions can be executed simultaneously by checking the template bits.

### Problem 7 (Block Structured ISA and Superblocks) [20 Points]

a. The Block-Structured ISA is fundamentally different from the Superblock. How so? [6 points]

Blocks in the BS-ISA are atomic, single-entry single-exit blocks; either they execute completely or not at all. In the latter case, the hardware will have to reverse the effects of every instruction that was executed when it shouldn't.

A superblock may execute partially and normally branch out prematurely because it is single-entry multiple exit. In the latter case, the compiler will have to insert fix-up code, if needed.

b. What are three advantages of Block-Structured ISA caused by this difference? [7 points]

1. Enables optimization of multiple paths (multiple enlarged blocks) in the program. Can dynamically adapt to changes in frequently executed paths at run-time due to the dynamic selection of "next enlarged block". Hence, it is less dependent on profile information than the superblock.
2. Atomic blocks enable more aggressive compiler optimizations (e.g. very speculative code reordering).
3. Compiler does not need to worry about generating fix-up code.
4. Can explicitly represent dependencies among operations within an enlarged block.

c. What are three disadvantages of Block-Structured ISA caused by this difference? [7 points]

1. Requires hardware support for atomicity.
2. Requires "next enlarged block" prediction.
3. Requires ISA support.
4. Can waste work due to atomicity requirement.

### Problem 8 (Register Alias Tables) [25 Points]

- a. What is the primary function of the register alias table in an out-of-order execution processor? Explain briefly. [2 points]

Register renaming.

- b. When does an instruction access the register alias table for that purpose? [2 points]

In decode/rename stage. (to read the source register value/tag, and to rename the destination register)

- c. Suppose you froze the execution of a processor and magically observed all entries in the register alias table. You found that all entries in the register alias table are invalid. Is this possible or is this due to an error in the design? If it is possible, explain why are all entries are invalid? If not, why is this not possible? [6 points]

It is possible. All registers could be waiting to be written by in-flight (i.e., issued but not yet finished execution) instructions.

- d. Pentium 4 contains a retirement register alias table in addition to a traditional register alias table. What function does the retirement register alias table serve? [2 points]

Maintain the (pointers to) architectural/committed register state. Used to recover the RAT state on exceptions and branch mispredictions.

- e. How many entries are there in Pentium 4's retirement register alias table? (We are not looking for an absolute number.) [2 points]

As many as the architectural registers in the x86 ISA.

- f. Suppose you froze the execution of the Pentium 4 and magically observed all entries in the retirement register alias table. You found that one entry in the retirement register alias table is invalid. Is this possible or is this due to an error in the design? If it is possible, explain why one entry could be invalid? If not, why is this not possible? [6 points]

It is not possible (must be a design error). The architectural register state of a processor should be valid at all times.

- g. Can the contents of the frontend register alias table and the retirement register alias table be the same? Explain why or why not? [5 points]

Yes. For example:

1. when a branch misprediction recovery is performed, the contents of the retirement RAT are simply copied into the frontend RAT, which will make the contents of each the same.
2. If none of the in-flight instructions have a destination register (e.g. all NOPs) retirement register state will be the same as frontend register state.