

CS 433 Midterm Exam – March 4, 2008

Professor Sarita Adve

Time: 2 Hours

Please clearly print your name and NetID and circle the appropriate category in the space provided below.

Name	Solutions	
NetID		
Category	3 Credit Hours	4 Credit Hours

Instructions

1. You may only use class handouts from this semester's offering, the course text (*Computer Architecture: A Quantitative Approach* – 4th Edition – by Hennessy and Patterson), your own homework submissions for this course, slides presented in class for mini-projects, papers indicated as reference material in class, and notes written or typed by yourself. You may also use homework solutions and sample midterms provided on the course website. No other materials are allowed, including other books, notes prepared by others, or materials from previous offerings of this course (except as noted here) or from other universities.
2. Calculators are allowed. You may not use any other electronic devices for any purpose during the exam.
3. Please do not turn in loose scrap paper. Limit your answers to the space provided if possible. If this is not possible, please write on the back of the same sheet. You may use the back of each sheet for scratch work.
4. *In all cases, show your work. No credit will be given for numeric answers if there is no indication of how the answer was derived. Partial credit will be given even if your final solution is incorrect, provided you show the intermediate steps in reaching the final solution.*
5. If you believe a problem is incorrectly or incompletely specified, make a reasonable assumption and solve the problem. The assumption should not result in a trivial solution. In all cases, clearly state any assumptions that you make in your answers.
6. This exam has **5 problems** and **8 pages** (including this one). Part C of problem 5 is only for graduate students. All other problems are required for all students. Please budget your time appropriately. Good luck!

Problem	Maximum Points	Received Points
1	5	
2	20	
3	15	
4	6	
5	4 for undergrads, 10 for grads	
Total	50 for undergrads, 56 for grads	

Problem 1 [5 points]:

Suppose we apply an enhancement, E1, that speeds up 20% of a program by a factor of 4. And we apply another enhancement, E2, that speeds up another 10% of the program by a factor of 10. Assume the two enhancements are independent and affect different parts of the program (there is no overlap between the 20% and 10% of program). What is the overall speed up for the entire program using both E1 and E2 simultaneously?

Solution:

This problem tests the understanding of Amdahl's law.

According to Amdahl's law,

$$\text{Speedup} = (\text{old latency}) / (\text{new latency})$$

$$\text{new latency} = (1 - f1 - f2) * \text{old latency} + f1/S1 * \text{old_latency} + f2/S2 * \text{old_latency}.$$

$$f1 = 20\%, S1 = 4; f2 = 10\%, S2 = 10;$$

$$\text{new latency} = \text{old latency} * .76$$

$$\text{Final speedup} = 1.32$$

Grading:

2 point for listing Amdahl's law and the standard equation;

1 points for the equation of this problem;

2 points for substituting the correct values in the equation.

Problem 2 [20 Points]

This problem concerns Tomasulo's algorithm (with reservation stations) with the reorder buffer scheme discussed in detail in the lecture notes. We have the following changes/additions/clarifications relative to the discussion in class.

- Assume the following information about functional units.

Functional Unit Type	Cycles in EX
Integer Mul	2
Integer Div	10
Integer Add	1

- Assume you have unlimited reservation stations, functional units, CDBs, and reorder buffer entries.
- Functional units are not pipelined.
- An instruction waiting for data on the CDB can move to its EX stage in the cycle after the CDB broadcast.
- Assume that integer instructions also follow Tomasulo's algorithm (analogous to the floating point instructions) so they can be issued out of order and the result from the integer functional unit is also broadcast on the CDB and forwarded to dependent instructions through the CDB.
- Assume the processor can issue into the reservation stations (and reorder buffer) only one instruction per cycle. Assume it can commit only one instruction per cycle as well.

Complete the following table using Tomasulo's algorithm with the above specifications. Fill in the cycle numbers in each pipeline stage for each instruction (CMT stands for the commit stage). For each instruction, indicate where its source operands are read from (use RF for register file, ROB for reorder buffer, and CDB for common data bus). The entries for the first instruction and for the issue stage are filled in for you.

	Instruction	Issue	Operand1	Operand1 Source	Operand2	Operand2 Source	EX	WB	CMT
1	MUL R2, R6, R12	1	R6	RF	R12	RF	2	4	5
2	DIV R1, R1, R2	2	R1		R2				
3	ADD R5, R1, R3	3	R1		R3				
4	ADDI R7, R5, 4	4	R5						
5	ADD R5, R6, R8	5	R6		R8				
6	ADDI R8, R8, 2	6	R8						
7	ADD R9, R6, R9	7	R6		R9				
8	ADD R5, R5, R10	8	R5		R10				
9	ADD R6, R8, R5	9	R8		R5				

Solution:

	Instruction	Issue	Operand 1	Operand 1 Source	Operand 2	Operand 2 Source	EX	WB	CMT
1	MUL R2, R6, R12	1	R6	RF	R12	RF	2	4	5
2	DIV R1, R1, R2	2	R1	RF	R2	CDB	5	15	16
3	ADD R5, R1, R3	3	R1	CDB	R3	RF	16	17	18
4	ADDI R7, R5, 4	4	R5	CDB			18	19	20
5	ADD R5, R6, R8	5	R6	RF	R8	RF	6	7	21
6	ADDI R8, R8, 2	6	R8	RF			7	8	22
7	ADD R9, R6, R9	7	R6	RF	R9	RF	8	9	23
8	ADD R5, R5, R10	8	R5	ROB	R10	RF	9	10	24
9	ADD R6, R8, R5	9	R8	ROB	R5	CDB	11	12	25

Grading:

0.5 point per entries

1 point if the majority is correct

Do not cascade errors. Do not take off additional points if an earlier error causes later inaccuracies.

Extra space for problem 4 if required.

Solution:

Loop:

```
LD.D F2, 0(R1)
MUL.D F4, F6, F2
ADD.D F4, F4, F8
SD.D F4, 0(R1)
```

```
LD.D T0, 8(R1)
MUL.D T2, F6, T0
ADD.D T4, T2, F8
SD.D T4, 8(R1)
```

```
LD.D T6, 16(R1)
MUL.D T8, F6, T6
ADD.D T10, T8, F8
SD.D T10, 16(R1)
```

```
LD.D T12, 24(R1)
MUL.D T14, F6, T12
ADD.D T16, T14, F8
SD.D T16, 24(R1)
```

```
DADDUI R1, R1, #32
BNE R1, R3, Loop
```

E:

For loop unrolling, not as long as the unrolled loop works the same, should give full credits. No need to schedule at this point.

Mem	FP ALU	Int ALU
LD.D F2, 0(R1)		
LD.D T0, 8(R1)		
LD.D T6, 16(R1)	MUL.D F4, F6, F2	
LD.D T12, 24(R1)	MUL.D T2, F6, T0	
	MUL.D T8, F6, T6	DADDUI R1, R1, #32
	MUL.D T14, F6, T12	
	ADD.D F4, F4, F8	
	ADD.D T4, T2, F8	
	ADD.D T10, T8, F8	
	ADD.D T16, T14, F8	
SD.D F4, -32(R1)		
SD.D T4, -24(R1)		
SD.D T10, -16(R1)		
SD.D T16, -8(R1)		BNE R1, R3, L

Grading:

6 points for loop unrolling, 1/3 pt each instruction, round up.

9 pts for filling up the table, .5 pt for each instruction element. Round up.

Problem 4 [6 points]

Consider a loop that is entered several times in a program. Each time it is entered, the loop performs 10 iterations. Each iteration executes four branches with the following outcomes (branch 1 occurs before branch 2 which occurs before branch 3 which occurs before branch 4 in each iteration):

	Iteration									
	1	2	3	4	5	6	7	8	9	10
Branch 1	N	N	N	N	N	N	N	N	N	T
Branch 2	T	T	T	T	T	T	T	T	T	
Branch 3	T	T	N	T	T	T	N	T	N	
Branch 4	N	N	T	N	N	N	T	N	T	

When Branch 1 is taken at iteration 10, the program counter leaves the loop, and branches 2, 3, and 4 are not reached.

Of all the dynamic branch predictors studied in class, state the cheapest predictor that will give the best misprediction rate for each of the following branches. Explain why.

(A) Branch 1:

Solution:

Branch 1 is almost always not taken. We need to use a **2-bit predictor**. It will have one misprediction each time the branch leaves the loop. In contrast, a 1 bit predictor will have two mispredictions – in the first and last iterations of each loop invocation.

(B) Branch 2:

Solution:

Branch 2 is always taken. It will have the same result on all predictors. Therefore, use a **1-bit predictor**.

(C) Branch 4:

Solution:

Branch 4 is always the opposite of the most recent Branch 3. Therefore, use a **(1,1) correlating predictor**.

Grading:

2 points each: 1 per predictor, 1 per reason.

Problem 5 [4 points for undergraduates, 10 points for graduates.]

Consider a program that accesses all elements of a large 40 MB array, and each element is accessed exactly once. Consider a system with a 1 MB single level cache running this program. What can you say about the order in which the program accesses the different array locations and/or any other cache parameters for the following scenarios to be true?

There may be many possible correct solutions. As long as the students' solutions make sense, full credit.

Part A (2 points): The cache miss ratio for the program is 100%.

Solution:

There are many possibilities. Here are 2:

- i) An array element and a cache line are the same size. Every miss is compulsory.
- ii) The cache is direct-mapped, and element accesses are such that all elements with the same index bits are accessed in sequence. For example, if the address is 64, the offset is 2 bits, the index is 8 bits, and each element is one byte, then the elements could be accessed in the order 0, 1024, 2048, etc., followed by 1, 1025, etc. Every cache access evicts the last line and loads the new one in the same location. The first miss in each cache address is compulsory; the rest are conflict.

Part B (2 points): The cache miss ratio for the program is 1/16 (or 6.25%).

Solution:

There are many possibilities. Here are 3:

- i) Each cache line contains 16 elements, and they are accessed in sequence. Every miss is compulsory.
- ii) Every cache line contains 32 elements. In accessing array elements, all even entries are accessed in order, followed by all the odd entries. The first half of cache misses are compulsory. By the second half, the lines to be accessed have already been evicted. Therefore the second half of cache misses are capacity.
- iii) C.ii below, with one access per element.

The next part is only for graduate students.

Part C (only graduate students must do this part) (6 points): Now assume that the program accesses each array element exactly twice over the course of its execution. With a 1MB level 1 cache, the miss ratio is 100%. Adding a 10 MB level 2 cache, however, significantly reduces the miss ratio. Describe two possible (different) scenarios that can make this possible. What types of L1 cache misses are affected by each of these scenarios?

Solution:

There are many possibilities. Here are 2:

- i) Like A.i above, each cache line contains one array element. The program accesses each 10MB block in order, and then accesses it again in order. For each element, its first access is a compulsory miss, and its second is capacity. By the time the second access comes around, the entire block has been evicted to make room for the next block. Adding the L2 cache removes all capacity misses, and moves the miss ratio down to 50%

- ii) Each cache line (for both L1 and L2 caches) is 1KB, and each element is 64 bytes. One line = 16 elements, and the array has 640K entries. The elements are accessed in this order: 0, 64, 128, ... 640K, 1, 65, ... After the entire array is accessed, it is accessed again in the same order. The first 1024 entries are compulsory misses. The rest are capacity misses. By the time the process accesses entry 1, the cache line with entries 0-15 will already have been evicted from the L1 cache. But it will still be in the L2 cache. When the process gets to entry 16, none of the desired lines will be in either cache; ultimately every line in L2 will be replaced. The same will happen starting at 32 and 48. Therefore, only multiples of 16 will be misses (compulsory ones), so the miss ratio drops to $1/16=6.25\%$