Name **Midterm Quiz Solutions**________________________ GTid#____________________

CS 4290/6290: High-Performance Computer Architecture

Spring 2004

Midterm Quiz

This is a closed-book exam. There are 6 problems on this quiz and the maximum total number of points is 40. Write your answers in the space provided. Use the extra pages for scratch space is needed.

Problem 1. [3 points] Amdahl's Law

For each of the following statements, indicate whether it is true or false:

For an overall speedup of 2, the new execution time must be 50% or less of the old execution time.

FALSE **A)** A speedup of 40 on 40% of the program will result in an overall speedup of at least 2.

The new overall execution time is 60%+40%/40=61% of the old.

FALSE **B)** A speedup of 20 on 50% of the program will result in an overall speedup of at least 2.

The new overall execution time is 50%+50%/20=52.5% of the old.

TRUE **C)** A speedup of 10 on 60% of the program will result in an overall speedup of at least 2.

The new overall execution time is 40%+60%/10=46% of the old.

Problem 2. [3 points] Processor Performance Equation

For each of the following statements, indicate whether it is true or false:

Execution time = clock cycle time * CPI * IC

FALSE **A)** Program execution time increases when the clock rate increases

Clock rate is the inverse of clock cycle time.

TRUE **B)** Program execution time increases when the CPI increases

TRUE **C)** Program execution time increases when the instruction count (IC) increases

Problem 3. [4 points] Register Renaming

For each of the following statements, indicate whether it is true or false:

FALSE **A)** Register renaming eliminates stalls due to name dependences through memory

Register renaming renames registers. It does not affect dependences through memory.

TRUE **B)** Register renaming eliminates stalls due to output (WAW) dependences on registers

Output dependences are name dependences and are removed by register renaming.

TRUE **C)** Register renaming eliminates stalls due to anti (WAR) dependences on registers

Anti-dependences are name dependences and are removed by register renaming.

FALSE **D)** Register renaming eliminates stalls due to flow (RAW) dependences on registers

Flow dependences are true (not name) dependences and are not removed by register renaming.

Problem 4. [3 points] Pipelining

For each of the following statements, indicate whether it is true or false:

FALSE **A)** Splitting the shortest stage of a five-stage pipeline will result in a higher clock rate.

The longest stage in the pipeline determines the clock rate .

TRUE **B)** With single-issue, in-order execution, and the classical five-stage pipeline with no bypassing, WAW hazards never cause any "bubbles" (stalls) in the pipeline.

All writes occur in-order in the WB stage of the pipeline, so there are no WAW hazards .

FALSE **C)** With a single-issue, in-order execution, and the classical five-stage pipeline with bypassing, RAW hazards never cause any "bubbles" (stalls) in the pipeline.

Even with bypassing, there can still be a RAW hazard from a memory load to a dependent instruction immediately after the load. This is because the loaded value is available at the end of the MEM stage, but it needed at the beginning of the EXE stage in the same cycle.

Name Midterm Quiz Solutions GTid#

Problem 5. [20 points] Tomasulo's Algorithm

A processor uses Tomasulo's algorithm in its floating-point unit. The processor has one CDB, but can issue two instructions per cycle. The figure shows the current state of the processor, at the very end of a clock cycle. The adder is not doing anything, but the multiplier has just finished an operation. A new cycle begins now.

Instruction	Instruction Status		
	Issue	Execute	Wr. Result
MUL F0, F1, F2	Yes	Yes	Yes
ADD F1, F0, F3	Yes		
ADD F2, F3, F2	Yes	Yes	
ADD F3, F1, F4	Yes		
ADD F0, F1, F2			

Register Status	V_i	Q_i
F0	125*75	~~Mul1~~
F1		Add1
F2		Add2
F3	~~20~~	Add3
F4	43	
F5	12	
F6	3.75	
F7	0.01	

Reservation Stations

Name	Busy	Op	V_j	V_k	Q_j	Q_k
Add1	Yes	ADD	125*75	20	~~Mul1~~	
Add2	Yes	ADD	20	75		
Add3	~~No~~ Yes	ADD		43	Add1	
Mul1	~~Yes~~ No	~~MUL~~	~~125~~	~~75~~		
Mul2	No					

A) [8 points] What will happen in this new cycle when the processor tries to issue the next two instructions? Update the figure to reflect the resulting state of the processor. If one or both of the two instructions can not issue in this cycle, briefly explain why it/they can not issue.

The first add (ADD F3, F1, F4) can issue and the updates to the figure for this part are shown in red. The second add (ADD F0,F1,F2) can not issue because there is no more available reservation stations for adds.

B) [5 points] The multiplier and the adder are both free to begin a new computation in this cycle. Which instructions can begin to execute in the adder and which in the multiplier? Update the figure to reflect the resulting state after beginning execution for these instructions.

The only instruction that can begin execution (both of its source operand fields have values) is Add2, which can execute in the adder. The multiplier has already executed the only multiply available and becomes idle. The updates to the figure for this part are shown in blue.

C) [7 points] What is going to be broadcast on the CDB in this cycle? Update the figure to reflect the state after the broadcast and the resulting actions are complete.

The multiply has completed execution in the previous cycle and writes its result now. The value of the multiply's result and its "name" (Mul1) are broadcast on the CDB. The updates to the figure from this part of the problem are shown in green.

Name **Midterm Quiz Solutions** GTid#

Problem 6. [7 points] Branch Prediction

A single-issue, statically-scheduled, deeply pipelined processor is executing a program. During the entire program run, the processor retires (commits) a total of 1,000,000 instructions. With a perfect BTB (which correctly predicts both the target and the direction every time), there would be no stalls in the pipeline and the program would complete in exactly one millisecond. However, the BTB is not perfect. Of the 1,000,000 dynamic instructions, 200,000 are branches. Only 170,000 of those are BTB hits (the BTB has an entry for them). Of the 170,000 BTB hits, 150,000 are correctly predicted, while for the other 20,000 the BTB incorrectly predicts the direction, target, or both. Of the 30,000 BTB misses (those branches that the BTB did not have an entry for), 10,000 are not taken and 20,000 are taken. The BTB is the processor's only means of predicting branches, there are no delay slots, and the processor makes no attempt to correct a mispredicted branch until the branch goes through the last stage of the pipeline. At the end of that last pipeline stage the correct direction and target are known and, if needed, in the next cycle the fetch restarts from the first instruction that should execute after the branch. With the imperfect BTB, the overall execution time of the program is 1.8 milliseconds. How many stages are there in the processor's pipeline? Show your work.

Of the branches that hit (have an entry) in the BTB, only the 20,000 that are mispredicted will need recovery. When a branch misses (does not have an entry) in the BTB, the processor behaves like it is not a branch and keeps fetching on. This means that the 10,000 branches that miss in the BTB but are not taken are not a problem, but the 20,000 that miss in the BTB and are taken require recovery. This gives us a total of 40,000 branches that require recovery. Note that there are no non-branch instructions that hit in the BTB: entries in the BTB are allocated only for branches and the only way for an instruction to hit in the BTB is to match the address in a BTB entry; for an instruction to hit in the BTB, there had to have been a branch with the same address.

With no stalls and no branch recoveries (the ideal case) the processor executes 1,000,000 instructions in 1ms. Because ideally this processor executes one instruction per cycle, the cycle time is 1ns. Note: if we take into account the time it takes to "ramp up" the pipeline which is N stages long in the beginning, there are actually 1,000,000+N-1 cycles in 1ms. However, N is always way smaller than 1,000,000, so we can safely neglect the N-1 part without changing the end result.

Now there are at least two ways to solve the problem. The easier way is to note that 1.8 ms is 1,800,000 cycles, so our 40,000 recoveries have resulted in 800,000 extra cycles. Then each recovery took 800,000/40,000=20 cycles. The more difficult way is to use the processor performance equation. The "Real" execution time ET_{Real} is 1.8ms, the "Ideal" execution time ET_{Ideal} is 1ms. Instruction count and clock cycle time is the same in both cases, but the "Real" CPI and the "Ideal" CPI differ. Thus, $ET_{Real}/ET_{Ideal}=CPI_{Real}/CPI_{Ideal}$. We know that CPI_{Ideal} is 1 and that ET_{Real}/ET_{Ideal} is 1.8, so CPI_{Real} is 1.8. On all instructions except branches that need recovery CPI_{Real} is 1, but on branches that need recovery this CPI is 1 (for the branch itself) plus the recovery penalty P. We have:

CPI_{Real}=1*<Fraction of instrs that do not cause recovery> + (1+P)<Fraction of instructions that cause recovery>
$CPI_{Real}=1*(960{,}000/1{,}000{,}000)+(1+P)*(40{,}000/1{,}000{,}000)=1*0.96+(1+P)*0.04=1+P*0.04$
$1.8=1+P*0.04$
$P=(1.8-1)/0.04=0.8/0.04=20$

Each time a branch needs recovery, 20 cycles are lost. From the text of the problem, branch recovery is done only after the branch exits the Nth (last) stage of the pipeline. Look at the cycle in which the branch is in the last stage of the pipeline. All stages they have "bad" instructions, except the Nth (last) stage which has the branch itself. In the next cycle we start fetching correct instructions, so we lost N-1 cycles due to the recovery-causing branch. Now we know that N-1=20 cycles, so the pipeline has 21 stages. To convince yourself that the penalty is one cycle less than the full pipeline length, try an example with a short (e.g. 2-stage) pipeline. This last step is a bit tricky, so no points were taken off for not doing it. In other words, 20 is treated as a correct answer.