ELE 375
Fall, 2000
Sample Midterm Questions
Prof. Martonosi

(Use these questions to help practice important concepts and to get a sense of the style of questions I write, but do not necessarily use this as a gauge of exam length. The exam is geared to be done in 65 minutes.)

1) Assume a 5-stage DLX pipeline as discussed in class, with an ideal CPI of 1. Assume that instructions and data are fetched from the same memory, so that there is a structural hazard on every load or store instruction. If loads and stores together represent 23% of the instructions executed, what is the real CPI of this pipeline?
   a) 1.00 cycles per instruction
   b) 1.23 cycles per instruction
   c) 1.12 cycles per instruction
   d) 1.46 cycles per instruction

CPIreal = CPIbase + CPIstall
*CPIstall = probability of stall * number of stall cycles*
*CPIstall = 0.23 * 1 = 0.23*
So the answer is b

2) Name the 5 "classic components" of any computer

Datapath,
Control,
Memory
Input,
Output,

3) Distinguish between computer architectures and implementations

Computer architecture defines an instruction set and interface that compilers and software writers use to program CPUs. Implementations refer to individual chip implementations that implement a particular instruction set architecture. Typically a successful architecture lasts decades, while good implementations are out of date in a couple of years, due to improvements in fabrication technology etc.

4) Give an example in computers or in everyday life of latency vs. throughput.

Computers: Pipelining increases the instruction throughput of a machine, but does so without decreasing the latency of individual instructions.
Everyday Life: Adding a lane to the NJ turnpike improves the throughput of the highway. It only improves the latency of my drive to Newark airport if congestion has been a problem.

5) Computer A has an overall CPI of 1.3 and can be run at a clock rate of 600MHz. Computer B has a CPI of 2.5 and can be run at a clock rate of 750 Mhz. We have a particular program we wish to run. When compiled for computer A, this program has exactly 100,000 instructions. How many instructions would the program need to have when compiled for Computer B, in order for the two computers to have exactly the same execution time for this program?

*Time = InstrCount * CPI * Clock Cycle Time*
*Time for A = 100,000 * 1.3 * (1/600 MHz)*
*Time for B = InstrCountB * 2.5 * (1/750 Mhz)*

If the two execution times should be equal, then:
*InstrCountB = (750Mhz * 1.3 * 100,000) / (600Mhz * 2.5) = 65,000*

(Note that the instruction count is much lower for computer B than for computer A on the same program. To achieve this in real life, one would need a dramatically different architecture (eg. B is a CISC machine) or a much more aggressive compiler for B.)

6) Consider the MIPS datapath as shown eg in Figure 5.25. Consider the following timings given for different hardware elements. How many nanoseconds along the critical path through the hardware shown?

Hardware type	Delay
Memory access	4 ns
Register read/write	3 ns
ALU operation	2 ns

The critical path through this MIPS datapath is on a load instruction (although other datapaths or instruction sets might have a different critical path…) A load instruction does the following:

Instruction fetch	*Memory Access*	*4 ns*
Register fetch	*Register file read*	*3ns*
Address calc	*ALU op*	*2ns*
Data load	*memory access*	*4ns*
Register write	*register file write*	*3ns*

So the critical path is 16ns. The clock rate should be no faster than 1/16ns = 62.5MHz. (The GHz clocks on CPUs you can currently buy mean that these processors have single-cycle critical paths < 1ns!)

7) Imagine that you are able to perform benchmarking "races" to compare two computers you are thinking about buying. Come up with a list of 5 benchmark programs or usage scenarios you would use to create your own personalized benchmark suite. For each program you select, justify it. For the benchmark suite as a whole, discuss a method for calculated a weighted average of the different program run-times.

Answers are obviously very individual. The key thing would be to pick a mix of programs that you actually run in everyday life, eg Microsoft Word, Matlab, etc. And try to stress the machine in different ways: accessing memory, accessing disk, benchmarks with heavy graphics use (eg computer games). To calculate a weighted average, you'd probably want to guess at how you use the programs in real life. Eg, 30 percent email/netscape, 30 percent computer games, 30 percent Word, etc.

8) Name 3 key differences between a CISC instruction set like x86 and a RISC instruction set like MIPS.

- *RISC instruction encodings are typically a fixed width*
- *RISC instructions have very regular encodings*
- *RISC instruction sets have fewer different opcodes*
- *RISC instruction sets have only a few addressing modes*
- *RISC instruction sets are geared for compilers to manipulate, rather than for humans to program*
- *RISC instruction sets have very regular register files, so that efficient register allocation can be done by the compiler*

9) Use the register and memory values in the table below for the next questions. Assume a 32-bit machine with addressing modes as discussed in H&P. Assume each of the following questions starts from the table values; that is, DO NOT use value changes from one question as propagating into future parts of the question.

Register	Value	Memory Location	Value
R1	12	12	16
R2	16	16	20
R3	20	20	24
R4	24	24	28

a) Give the values of R1, R2, and R3 after this instruction : add R3, R2, R1

12 + 16 = 28 => at end: R1=12, R2 = 16, R3 = 28

b) What values will be in R1 and R3 after this instruction is executed: load R3, 12(R1)

Address to load from = 12 + 12 = 24. Value in mem location 24 = 28. So at end: R1=12 still and R3=28

c) What values will be in the registers after this instruction is executed: addi R2, R3, #16

20 + 16 = 36. So at end of instruction: R3=20, R2=36

10 A) In the pipelined abstraction below, draw arrows to indicate instances where bypassing must be used to get the correct answer:

1. Sub R4, R5, R7	IF	ID	EX	M	WB				
2. Load R8, 12(R4)		IF	ID	EX	M	WB			
3. Add R3, R7, R4			IF	ID	EX	M	WB		
4. Add R7, R8, R4				IF	ID	EX	M	WB	
5. Store R7, 16(R0)					IF	ID	EX	M	WB

Bypassing is needed in 4 instances:

1. R4: from 1. Sub Ex phase to 2. Load Ex phase
2. R4: from 1. Sub Ex phase to 3. Add Ex phase
3. R8: from 2. Load M phase to 4. Add Ex phase
4. R7: from 4. Add Ex phase to 5. Store M phase

b) Now consider a machine that only provides bypassing to the left side of the ALU but not to the right hand side of the ALU. Instructions that would require bypassing for the RHS operand will need to stall instead. Redraw the pipeline diagram to indicate these stalls.

Assume that in an ALU instruction, the first source operand goes to the LHS of the ALU and the second source operand goes to the RHS of the ALU. For example, in the first instruction above, R5 is on the LHS and R7 is on the RHS. In memory operations, the address calculation is performed with the value from the register file on the LHS.

The key thing here is to remember that separate physical wires are needed when bypassing to each side of the ALU. The point of the question was to get rid of one set of wires (the ones to the RHS) but retain the other set of wires.

So this pair of instructions executes without delay:
Sub R1, R2, R3
Add R4, R1, R2
Because R1 is used on the LHS of the ALU in the add instruction, and that side of the ALU has bypassing.

On the other hand, this pair of instructions must stall for the second instruction to wait until R1 is written back:
Sub R1, R2, R3
Add R4, R2, R1
Because R1 is now used on the RHS of the ALU in the add instruction, and that side of the ALU DOES NOT HAVE bypassing into it, according to the problem statement.

Diagram for the code above looks like:

1. Sub R4, R5, R7	IF	ID	EX	M	**WB**					
2. Load R8, 12(R4)		IF	ID	EX	M	WB				
3. Add R3, R7, R4			**stall**	IF	**ID**	EX	M	WB		
4. Add R7, R8, R4					IF	ID	EX	M	WB	
5. Store R7, 16(R0)						IF	ID	EX	M	WB

Note that I've used boldface type to show the WB phase in instruction 1 (a subtract) that writes back the value of R4 that is then reg-fetched in the boldface ID phase of instruction 3 (an add).