CS/ECE 552: Introduction to Computer Architecture

Prof. David A. Wood

Midterm Exam
March 13, 2007
7:15-9:15pm, 1221 CSS
Approximate Weight: 25%

CLOSED BOOK
ONE SHEET OF NOTES

NAME: _____Solution_________

DO NOT OPEN THE EXAM UNTIL TOLD TO DO SO!

Read over the entire exam before beginning. Verify that your exam includes all 7 pages. It is a long exam, so use your time carefully. Budget your time according to the weight of the questions, and your ability to answer them. Limit your answers to the space provided, if possible. If not, write on the BACK OF THE SAME SHEET. Use the back of the sheet for scratch work. WRITE YOUR NAME ON EACH SHEET.

Problem	Possible Points	Points
Problem 1	10	
Problem 2	15	
Problem 3	15	
Problem 4	25	
Problem 5	25	
Total	**90**	

Problem 1: (10 points)

Part A: (2 points)

Define the performance metric *MFLOPS* and explain why it can be problematic.

Millions of Floating Point Operations per Second

MFLOPS is a throughput measure that focuses on floating point computation. It is applicable only to floating point intensive applications. Furthermore, some machines will implement complex floating point operations (e.g., sqrt()) in hardware , while others will use a software routine that uses many simpler operations. Which will have (should have) the higher MFLOPS rate? This is usually solved using “normalized MFLOPS”, which eliminates this problem by assigning a total number of floating point operations to a specific benchmark.

Part B: (2 points)

Explain how the Intel MMX/SSE/SSE2 multimedia instructions are similar to or different from *vector instructions*?

The Intel multimedia instructions are like vector instructions in that they specify the execution of the same operation on multiple independent data items (e.g., bytes). The original MMX instructions operated on the 64-bit floating point registers, so the “vectors” could be at most eight one-byte data items. More recent extensions have introduced a separate register file and increased the size to 128-bits.

Part C: (2 points)

Explain how the Intel MMX/SSE/SSE2 multimedia instructions are similar to or different from *VLIW (Very Large Instruction Word) instructions*?

A VLIW instruction combines a group of potentially different operations into a single instruction. For example, the IA-64 architecture combines three independent instructions into a 128-bit instruction bundle. Each instruction can operate on different data (e.g., different registers) and specify different operations (e.g., add or sub).

Part D: (2 points)

What does it mean for exceptions to be *precise*? Explain why the *linear pipeline model* implements precise exceptions.

Precise exceptions enforce sequential semantics. All instructions before the excepting instruction must complete, all instructions after it must appear to have never begun (i.e., flushed from the pipeline without modifying architectural state), and the address of the excepting instruction must be passed to the exception handler.

Part E: (2 points)

Which mean should be used to average normalized MFLOPS? Briefly explain.

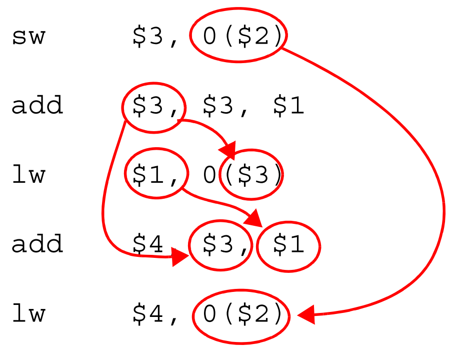
The harmonic mean is the correct mean to average normalized MFLOPS since it is a rate (i.e., inversely proportional to the total execution time). Normalized MFLOPS (defined in the answer to Part A) is NOT normalized to a base case (e.g., speedup), which calls for using the geometric mean.

NAME: ______________________________________

Problem 2: (15 points)

Part A: (5 points)

Indicate the *true data dependences* in the following MIPS code sequence:



```
sw      $3, 0($2)
add     $3, $3, $1
lw      $1, 0($3)
add     $4, $3, $1
lw      $4, 0($2)
```

Part B: (5 points)

What is meant by a *maybe dependence*? Explain and give an example. What problem can these cause in some pipelined processors? Explain.

A maybe dependence is a dependence that cannot be resolved at compile time. For example,

```
sw $3, 0($2)
lw $4, 0($5)
```

This cannot be resolved at compile time unless the compiler can prove whether or not $2 could equal $5. Note that the sw/lw sequence in Part A is a true dependence, because $2 does not change between the two instructions.

Part C: (5 points)

What is a *structural hazard*? What are the three techniques that can be used to avoid these hazards. Give two examples of how these techniques are used to resolve structural hazards in the 5-stage pipeline discussed in class.

A structural hazard is a resource conflict between instructions. That is, when two (or more) instructions want to use the same resource at the same time. For example, if a pipelined processor uses a single memory for instructions and data, then a structural hazard arises when loads and stores conflict with instruction fetches.

There are three general solutions:

1) Stall. Allow one instruction to proceed and stall the other(s). If a load instruction conflicts with an instruction fetch, the processor can stall the fetch and allow the load to proceed.

2) Replicate. Replicate the resource so that both instructions can proceed in parallel. This could be by having separate instruction and data memories, as discussed in class, or by having separate ports to a single memory.

3) Schedule. Schedule access to the resource so that conflicts don't arise. The 5-stage pipeline discussed in class does this for the register file write port by delaying all register writes until the W stage (ALU instructions could write the register file in M, but this could cause a structural hazard with a preceeding load instruction trying to write the register file in W).

NAME: ______________________________

Problem 3: (15 points)

Consider two implementations A and B of the MIPS instruction set, both built using the same technology. Machine A uses a simple single cycle datapath design and has a CPI of 1.0 with a cycle time of 1000ps. Machine B uses a pipelined datapath to reduce the cycle time to 300ps and has a CPI of 1.0 in the absence of control and data hazards. However, taken branch instructions incur 2 stall cycles and loads followed by a dependent instruction incur 1 stall cycle.

Part A: (4 points)

For the two workloads below, assume that 60% of branches are taken and 50% of loads are followed by a dependent instruction. Compute the CPI for the pipelined datapath.

Workload	% Branches	% Loads	% Stores	% Other	CPI_B
W1	10%	30%	15%	45%	1.27
W2	15%	15%	15%	55%	1.255

Part B: (5 points)

Pipelining makes Machine B faster than Machine A. How many times is B faster than A (this is also called Speedup)? Show your work.

Workload	Speedup of B
W1	2.62
W2	2.66

$Speedup_B = Time_A / Time_B = (N \times 1 \times 1000ps) / (N \times CPI_B \times 300ps) = 3.33 / CPI_B$

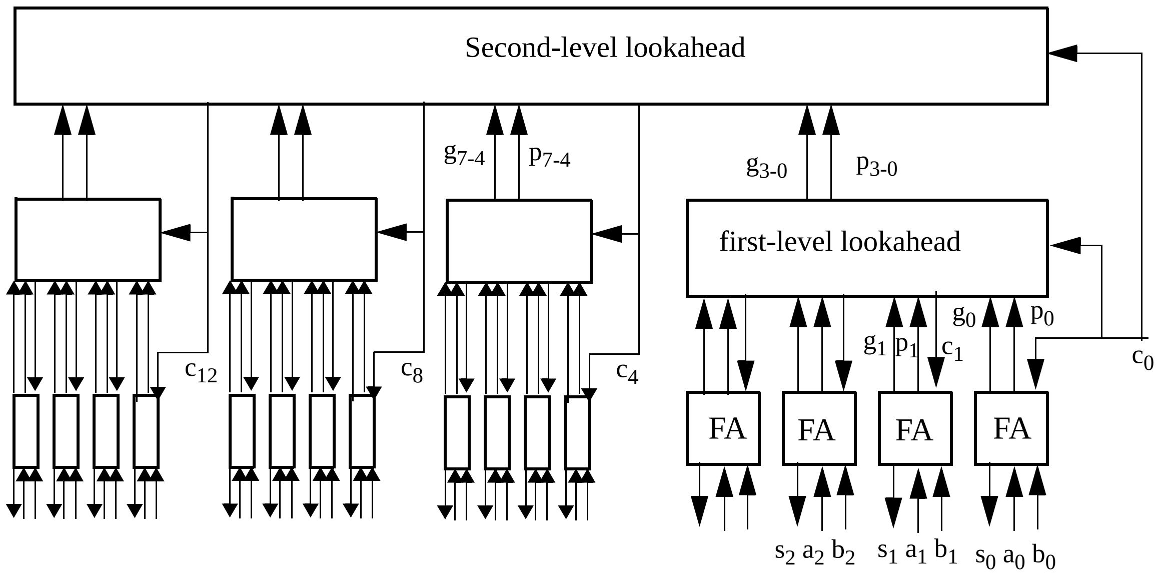
Part C: (6 points)

An alternative Machine C uses a pipeline design that reduces the cycle time to 200ps, but requires increasing the taken branch penalty to 4 stall cycles and the load-use delay to 2 stall cycles. Which machine has the best performance? Show your work.

Workload	CPI_C	Better machine
W1	1.54	C
W2	1.51	C

Problem 4: (25 points)

A 16-bit carry-lookahead adder composes multiple 4-bit carry-lookahead blocks into a two level tree structure.

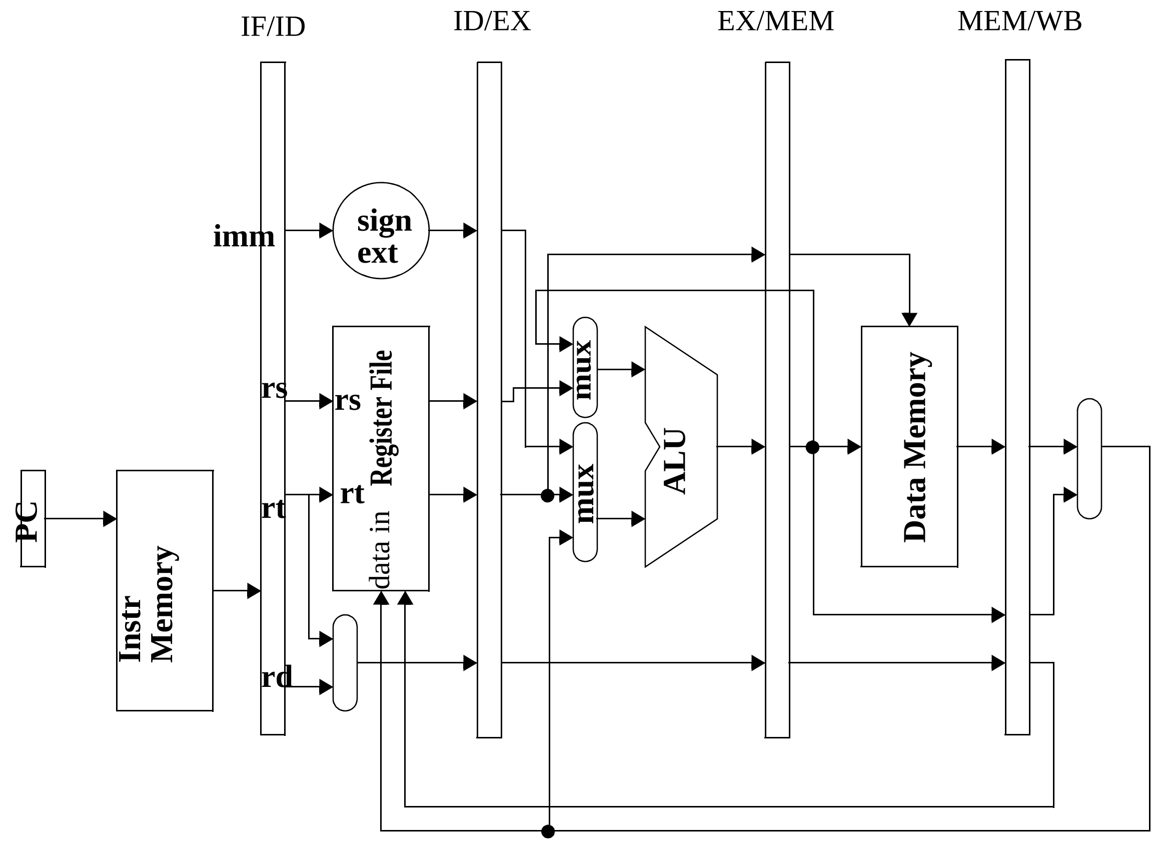


Write the boolean equation for each output signal listed in the table below. The equations should be optimized to minimize the delay from module inputs to outputs, where the modules are the full adder (FA), and the first- and second-level lookahead blocks. Compute the delays using the model below. The *worst case module delay* is the critical path from *any* input of a module to the output. The *critical path delay* is the critical path from the basic inputs a_i, b_i and c_0, which are assumed to change at time 0. Assume that you have only AND and OR gates available, but that each gate generates both the true output f and its complement $\overline{f}$. You also have the complements of the basic inputs available as well. The delay is computed using the formula *delay = (4 + 5n)*τ, where n is the number of inputs to the gate. Thus a 2-input AND gate has delay 14τ and the logic function *f = ab + cde* has delay 33τ (2-input OR with delay 14τ plus a 3-input AND with delay 19τ).

Signal	Equation	Worst case module delay	Critical path delay
$p_2 =$	$a_2 + b_2$	14	14
$g_2 =$	$a_2 b_2$	14	14
$c_5 =$	$g_4 + p_4c_4$	28	104
$c_7 =$	$g_6 + p_6g_5 + p_6p_5g_4 + p_6p_5p_4c_4$	48	124
$g_{11-8} =$	$g_{11} + p_{11}g_{10} + p_{11}p_{10}g_9 + p_{11}p_{10}p_9g_8$	48	62
$p_{11-8} =$	$p_{11}p_{10}p_9p_8$	24	38
$c_{12} =$	$g_{11-8} + p_{11-8}g_{7-4} + p_{11-8}p_{7-4}g_{3-0} + p_{11-8}p_{7-4}p_{3-0}c_0$	48	105
$s_{13} =$	$(a_{13}\overline{b}_{13} + \overline{a}_{13}b_{13})\overline{c}_{13} + (a_{13}b_{13} + \overline{a}_{13}\overline{b}_{13})c_{13}$	56	161
$s_{15} =$	$(a_{15}\overline{b}_{15} + \overline{a}_{15}b_{15})\overline{c}_{15} + (a_{15}b_{15} + \overline{a}_{15}\overline{b}_{15})c_{15}$	56	181

NAME: ____________________________________

Problem 5: (25 points)



High performance datapaths use bypass paths (also known as data forwarding logic) to reduce pipeline stalls. However, bypass paths are relatively expensive, especially in some wire constrained technologies. To reduce the cost (and potential cycle time impact), some architects have explored omitting some of the possible bypass paths. Consider the datapath illustrated above (note that the PC update logic and all control logic is intentionally omitted). This pipelined datapath is similar to the one in the book, *but has several differences including limited bypass paths*. BE SURE TO STUDY THE DATAPATH CAREFULLY! Assume that the register file internally bypasses the value, so that if register $i is read and written in the same cycle, then the read returns the new value. Assume that the control logic bypasses the data as soon as possible using the given forwarding data paths, and stalls in decode otherwise. You may NOT add additional data paths.

In this problem, you will look at how a program snippet performs on this pipeline. Recall that R-format instructions have the form:

```
      opcode rd, rs, rt
```

and I-format instructions have the form

```
      opcode rt, imm(rs)
```

or

```
      opcode rt, rs, imm
```

Use the table on the next page to show how the given instruction sequence flows through the pipeline and where stalls are necessary to resolve hazards.

NAME: ____

Consider the code and pipeline schedule below. Show the execution timing of this code on the pipeline above.

	Cycle																			
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
add $1, $2, $3	**F**	**D**	**X**	**M**	**W**															
sub $4, $1, $5		**F**	**D**	X	M	W														
or $6, $1, $4			**F**	~~D~~	D	X	M	W												
and $7, $4, $8				~~F~~	F	D	X	M	W											
lw $9, 4($7)						F	D	X	M	W										
add $1, $9, $2							F	~~D~~	~~D~~	D	X	M	W							
sw $1, 4($7)								~~F~~	~~F~~	F	~~D~~	~~D~~	D	X	M	W				

For each cycle where a stall occurs, explain why below.

Cycle 4: Register $4 in the ‘or’ instruction is dependent on the preceding ‘sub’ instruction. Because $4 is the ‘rt’ register, it cannot forward from the XM latch (labelled EX/MEM in this figure). Instead, it must stall in decode, which also stalls the fetch of the ‘and’ instruction. Because ‘rt’ can be forwarded from the MW latche (MEM/WB), the stall is only a single cycle.

Cycle 8: Register $9 in the second ‘add’ instruction uses the value produced by the ‘lw’ instruction. Loads don’t produce their value until the end of the M (MEM) stage, requiring a load-use stall for the ‘add’ in this cycle. This also stalls the ‘sw’ instruction in fetch.

Cycle 9: This pipeline does not have bypassing from the MW (MEM/WB) latch on the ‘rs’ side, so the ‘add’ must stall again until it can read the result from the register file.

Cycle 11 & 12: Register $1 in the ‘sw’ instruction is the ‘rt’ register, but there is no bypass path on the path to the data memory. Thus ‘sw’ must stall until it can read $1 from the (bypassing) register file.