CS/ECE 552: Introduction to Computer Architecture

Prof. David A. Wood

Midterm Exam
March 9, 2006
7:15-9:15pm, 1221 CSS
Approximate Weight: 25%

CLOSED BOOK
ONE SHEET OF NOTES

NAME: ______Solution Key_______________

DO NOT OPEN THE EXAM UNTIL TOLD TO DO SO!

Read over the entire exam before beginning. Verify that your exam includes all 8 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**	

NAME: ______________________________

Problem 1: (10 points)

Part A: (2 points)

Define the performance metric *SCPI* and explain why it is useful.

Stall Cycles per Instruction

SCPI helps isolate the effect on CPI of different stall conditions. For example, $SCPI_{branch}$ identifies the stall cycles per instruction due to branches.

Part B: (2 points)

Explain how *vector instructions* differ from regular MIPS instructions.

A MIPS instruction specifies that a single operation be performed on a pair of scalar operands. A vector instruction specifies that the same operation be performed on pairs of vectors (i.e., one dimensional arrays) or a scalar and a vector. For example, the instruction addsv $v1, $1, $v2 adds the scalar operand in register $1 to each of the 64 operands in vector register $v2, and stores the results in vector register $v1.

Part C: (2 points)

Explain how *VLIW instructions* differ from regular MIPS 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 a processor to be *superscalar*?

A superscalar processor is capable of executing multiple scalar instructions (e.g., MIPS instructions) in a single cycle. Superscalar processors may be statically scheduled, meaning they execute the instructions in the order specified by the compiler, or dynamically scheduled, meaning that the hardware may execute instructions out-of-order (but must commit them in order, to ensure sequential semantics and precise exceptions).

Part E: (2 points)

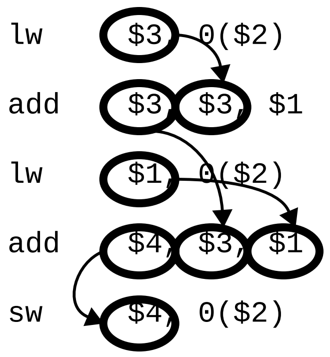
Which mean should be used to average execution time? Briefly explain.

The (weighted) arithmetic mean is the correct mean to average execution time since it is directly proportional to the total execution time of all the programs. The unweighted arithmentic mean assumes that each program is run equally often. The weighted arithmetic mean allows users to reflect which programs are more important (i.e., because they consume more time).

Problem 2: (15 points)

Part A: (5 points)

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



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

Part B: (5 points)

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

A control dependence arises when one instruction determines which instructions should be executed in the future. Conditional branches are the most common example, but other control transfer instructions (e.g., calls and jumps) cause control dependences. The problem that control dependences cause are called control hazards. These hazards arise because a pipelined processor normally assumes sequential execution flow and may have fetched and begun execution of instructions that should not be executed. The processor must stall to avoid this case and/or flush the incorrectly fetched instructions.

Part C: (5 points)

What is a *structural hazard*? Explain and give an example. What are the *three* techniques that can be used to avoid this problem?

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).

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 250ps and has a CPI of 1.0 in the absence of control and data hazards. However, taken branch instructions incur 3 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	15%	25%	10%	50%	1.395
W2	10%	35%	15%	40%	1.355

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	2.87
W2	2.95

$Speedup_B = Time_A / Time_B = (N \times 1 \times 1000ps) / (N \times CPI_B \times 250ps) = 4 / CPI_B$

Part C: (6 points)

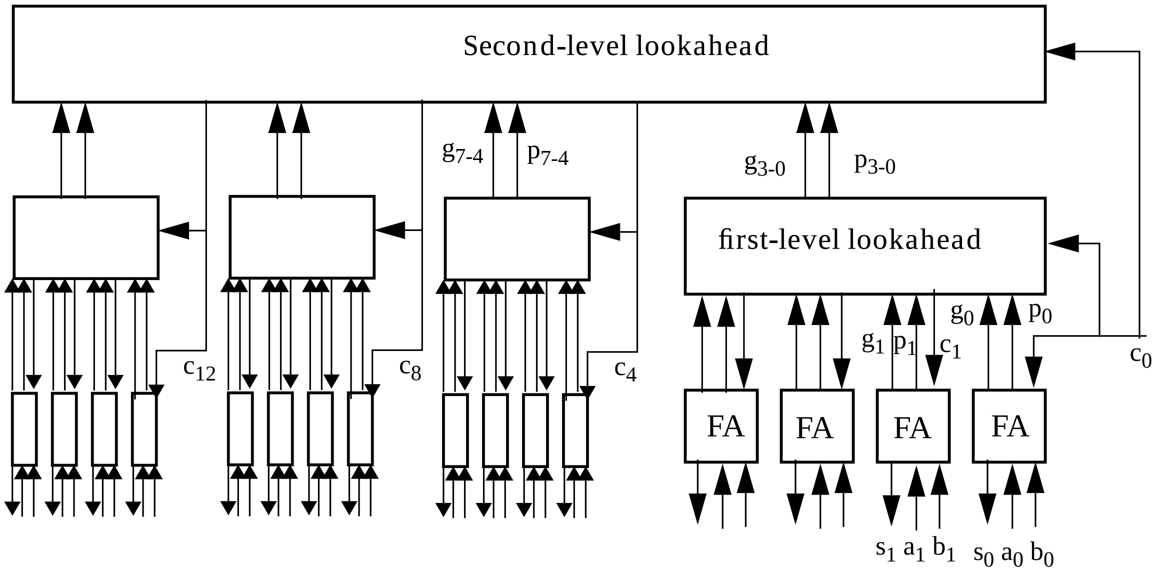
An alternative Machine C uses a pipeline design that reduces the cycle time to 200ps, but requires increasing the taken branch penalty to 5 stall cycles and the load-use delay to 2 stall cycles. Which machine has the best performance? Show your work.

Workload	CPI_C	Better machine
W1	1.7	C
W2	1.65	C

NAME: ______________________________

Problem 4: (25 points)

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

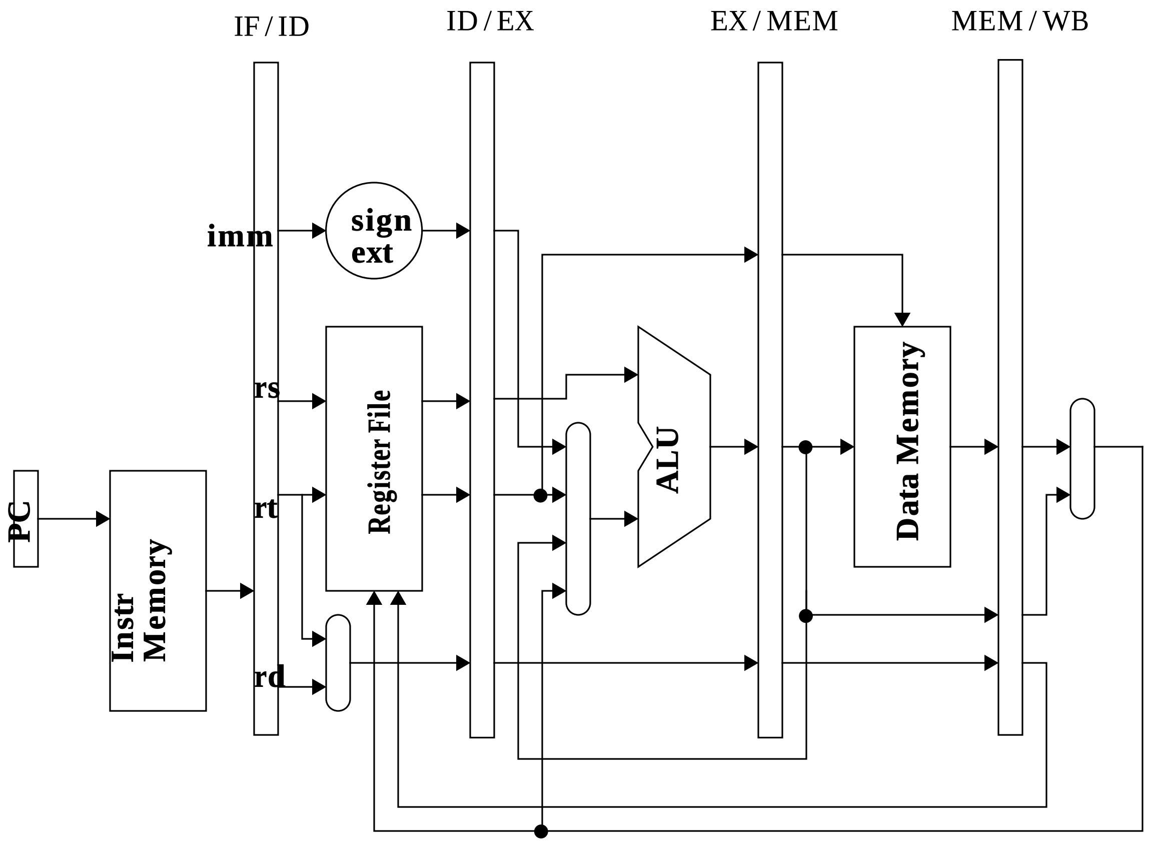


Write the boolean equation for each output signal listed in the table below. The equations should be optimized to minimize the delay from module inputs to outputs, where the modules are the full adder (FA), and the first- and second-level lookahead blocks. Compute the delays using the model below. The *worst case delay* is the critical path from any input of a module to the output. The *total 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 $\bar{f}$. You also have the complements of the basic inputs available as well. The delay is computed using the formula *delay = (3 + 4n)*τ, where n is the number of inputs to the gate. Thus a 2-input AND gate has delay 11τ and the logic function f = ab + cde has delay 26τ (2-input OR with delay 11τ plus a 3-input AND with delay 15τ).

Signal	Equation	Worst case delay	Total delay
$p_1 =$	$a_1 + b_1$	11 t	11 t
$c_6 =$	$g_5 + p_5g_4 + p_5p_4c_4$	30 t	101 t
$c_7 =$	$g_6 + p_6g_5 + p_6p_5g_4 + p_6p_5p_4c_4$	38 t	109 t
$g_{7-4} =$	$g_7 + p_7g_6 + p_7p_6g_5 + p_7p_6p_5g_4$	38 t	49 t
$c_{12} =$	$g_{11-8} + p_{11-8}g_{7-4} + p_{11-8}p_{7-4}g_{3-0} + p_{11-8}p_{7-4}p_{3-0}c_0$	38 t	87 t
$s_7 =$	$(a_7\bar{b}_7 + \bar{a}_7b_7)\bar{c}_7 + (a_7b_7 + \bar{a}_7\bar{b}_7)c_7$ OR $a_7\bar{b}_7\bar{c}_7 + \bar{a}_7b_7\bar{c}_7 + \bar{a}_7\bar{b}_7c_7 + a_7b_7c_7$	22 t 44 t	131 t 143 t
$s_{14} =$	$(a_{14}\bar{b}_{14} + \bar{a}_{14}b_{14})\bar{c}_{14} + (a_{14}b_{14} + \bar{a}_{14}\bar{b}_{14})c_{14}$ OR $a_{14}\bar{b}_{14}\bar{c}_{14} + \bar{a}_{14}b_{14}\bar{c}_{14} + \bar{a}_{14}\bar{b}_{14}c_{14} + a_{14}b_{14}c_{14}$	22 t 44t	139 t 151 t

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 only has bypass paths on one side of the ALU.* 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**	D	D	X	M	W												
or $6, $1, $4			**F**	F	F	D	X	M	W											
and $7, $4, $8						F	D	D	X	M	W									
lw $9, 4($7)							F	F	D	D	D	X	M	W						
add $1, $2, $4									F	F	F	D	X	M	W					
sw $1, 4($7)												F	D	D	D	X	M	W		