CS/ECE 552: Introduction to Computer Architecture

Prof. David A. Wood

Midterm Exam
March 6, 2012
7:15-9:15pm, B371 Chemistry
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 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	15	
Problem 2	15	
Problem 3	20	
Problem 4	25	
Problem 5	25	
Total	**100**	

NAME: _____

Problem 1: (15 points)

Part A: (3 points)

What is the *iron law of performance*?

Time/Program = Instructions/Program * Cycles/Instruction * Time/Cycle

Part B: (3 points)

In an ideal world, a processor with an N-stage pipeline would execute with a clock frequency N times faster than a single-cycle (i.e., non-pipelined) processor. Give two reasons why the clock frequency of a real pipelined processor would be *less* than N times faster.

Pipeline latch latency

Load imbalance between pipeline stages

Additional logic, e.g., muxes for data forwarding

Part C: (3 points)

In an ideal world, a pipelined processor would have a CPI of 1.0. Give two examples of why a real pipelined processor would have a CPI greater than one.

Control hazards, e.g., branches, may stall execution or squash instructions

Data hazards, e.g., load-use hazards, may stall execution

Part D: (3 points)

Explain the difference between a dependence and a hazard.

A dependence is a property of the instructions in a program, for example a true dependence arises when one instruction uses the value produced by an earlier instruction.

A hazard is a potential problem in a pipeline that may arise from a dependence. For example, a true data dependence causes a hazard when the value produced by the earlier instruction is not yet available in the register file when the dependent instruction attempts to fetch it.

Part E: (3 points)

The MIPS instruction set has fixed size instructions with only three instruction formats with key fields always occuring in the same place. Explain why these two properties make it easier to implement a pipelined processor. Give brief examples.

Fixed size instructions means that it is, in the absence of a control hazard, possible to determine the next instruction before decoding the current instruction. This allows fetch of instruction i+4 to proceed in parallel with the decode of instruction i.

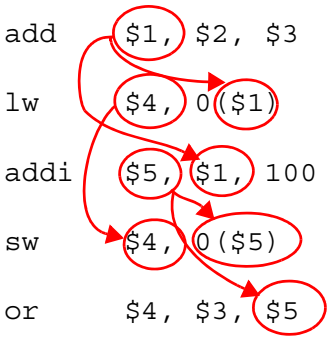
Fixed field placement makes it easier to decode instructions in parallel and perhaps partially overlap execution. For example, in MIPS it is possible to read the register file in parallel with instruction decode because the source register specifiers are always in the same place.

NAME: _____

Problem 2: (15 points)

Part A: (5 points)

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



Part B: (5 points)

What is meant by an *anti-dependence*? Are there any examples of anti-dependences in the code above?

An anti-dependence occurs between an instruction that reads a register (or memory location) and a subsequent instruction writes a new value to the same location. Equivalently, two instructions have an anti-dependence if swapping their order would result in a true dependence.

An anti-dependence exists between the sw instruction, which reads $4, and the or instruction, which overwrites $4 with a new value.

Part C: (5 points)

What problems, if any, do true and anti-dependences cause in the MIPS 5-stage pipeline that we have analyzed in class? Explain.

True dependences require that the data value produced by one instruction is passed to the consuming instruction. Since registers are read in decode (D) and written in writeback (W), there is a potential read-after-write (RAW) hazard. This can be resolved by stalling the pipeline or, in many cases, forwarding the value (except in the load-use case).

Anti-dependences are not a problem for register accesses because all instructions execute in order and always read the register file before they write it (i.e., read in Decode and write in Writeback), thus preventing a later instructions write from occuring before an earlier instructions read. They are not a problem for memory access because instructions are executed in order and all memory operations occur in the memory (M) stage.

A common misunderstanding is that “sequential semantics” alone prevents anti-dependences from causing hazards. Out-of-order processors still must provide sequential semantics, even though they rearrange the order of instruction execution (subject to maintaining dependence order).

NAME: _____

Problem 3: (20 points)

Consider two implementations A and B of the MIPS instruction set, both built using the same technology, but using different pipelines. Both machines have a base CPI of 1.0, but have different cycle times and different stalls for control and data hazards. In particular, the pipelines stall differently for taken and not-taken branches, when loads are followed by dependent instructions, and Machine B stalls a cycle on all stores.

	Machine A	Machine B
Cycle time	400ps	240ps
Taken branch stalls	1	4
Not-taken branch stalls	0	1
Load-use stalls	1	3
Store stalls	0	1

Part A: (12 points)

For the two workloads below, assume that 60% of branches are taken and 45% of loads are followed by a dependent instruction.

Workload	% Branches	% Loads	% Stores	% Other
W1	15%	30%	15%	40%
W2	20%	35%	10%	35%

Write the (symbolic) equations for the stall cycles per instruction (SCPI) for each type of stall:

$SCPI_{branch\text{-}taken}$	%branches * %taken branches * stall cycles per taken branch
$SCPI_{branch\text{-}nottaken}$	%branches * (1 - %taken branches) * stall cycles per not-taken branch
$SCPI_{load\text{-}use}$	%loads * %instructions dependent on loads * stall cycles per dependent load
$SCPI_{stores}$	% stores * stall cycles per store

Compute the SCPIs for both datapaths.

	Machine A		Machine B	
	W1	W2	W1	W2
$SCPI_{branch\text{-}taken}$	.15 * .6 * 1 .09	.2 * .6 * 1 .12	.15 * .6 * 4 .36	.2 * .6 * 4 .48
$SCPI_{branch\text{-}nottaken}$	.15 * .4 * 0 0	.20 * .4 * 0 0	.15 * .4 * 1 .06	.2 * .4 * 1 .08
$SCPI_{load\text{-}use}$	.3 * .45 * 1 .135	.35 * .45 * 1 .1575	.3 * .45 * 3 .405	.35 * .45 * 3 .4725
$SCPI_{stores}$	0	0	.15	.1

Compute the overall CPI for both datapaths.

	Machine A		Machine B	
	W1	W2	W1	W2
CPI	1.225	1.2775	1.975	2.1325

Part B: (4 points)

Which machine is faster? Compute the Speedup of Machine B over Machine A (i.e., Machine A is the "old" machine). Show your work.

Workload	Speedup of B	Faster machine?
W1	1.034	B
W2	0.98	A

$Speedup_B = Time_A / Time_B = (N \times CPI_A \times 400ps) / (N \times CPI_B \times 240ps)$

W1: 1.225 * 400 / 1.975 * 240 = 1.034

W2: 1.2775 * 400 / 2.1325 * 240 = 0.998

Part C: (4 points)

The slower machine would perform better with a faster clock. How fast would the slower machine's clock need to be to have the same performance as the faster machine? Show your work.

Workload	Slower Machine	Clock cycle time to achieve equal performance
W1	A	387ps
W2	B	240ps

$CPI_{slow} * Cycle_{slow} = CPI_{fast} * Cycle_{fast}$

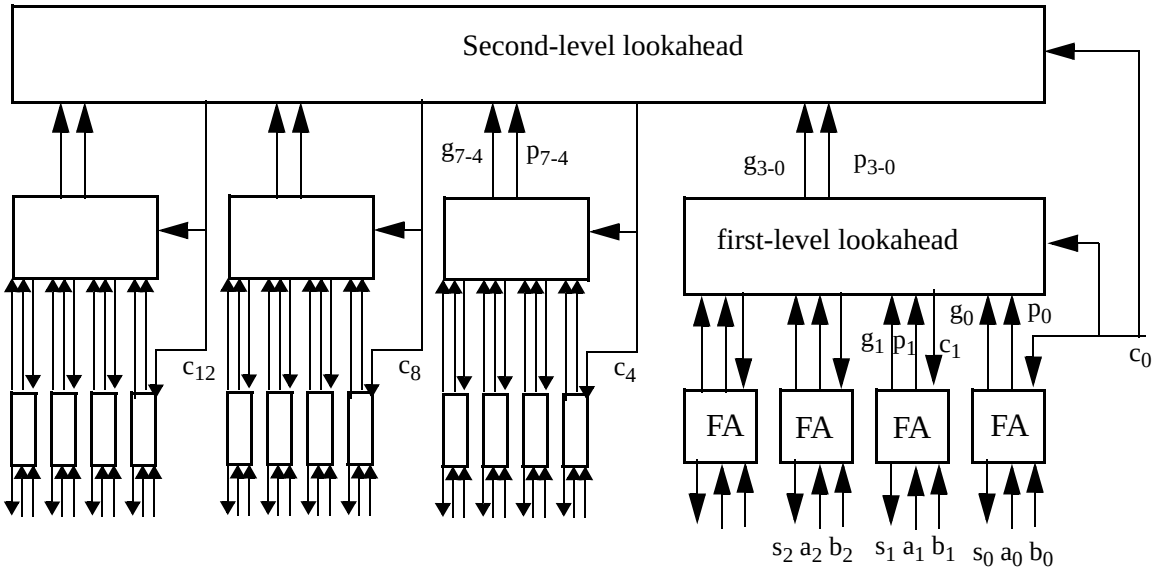
W1: 1.225 * C = 1.975 * 240

W2: 1.2775 * 400 = 2.1325 * 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.

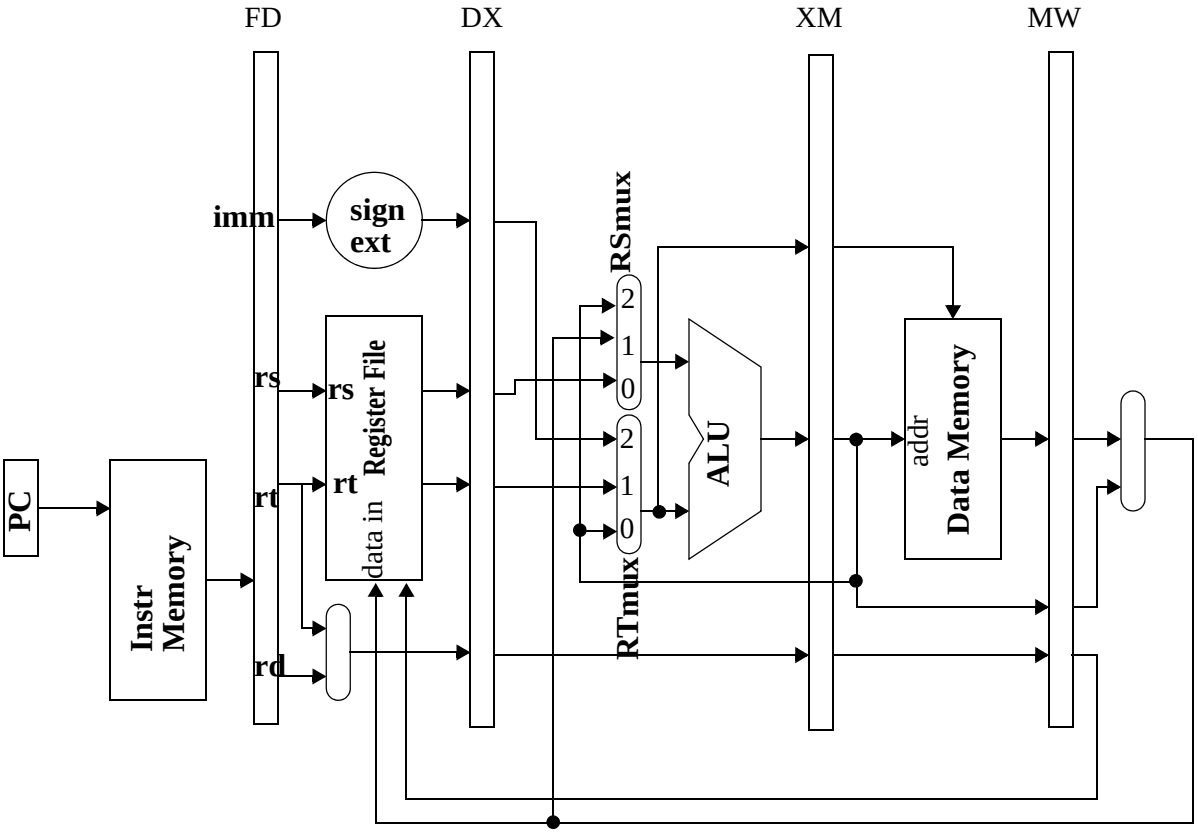


Write the boolean equation for each output signal listed in the table below. The equations should be optimized to minimize the delay from module inputs to outputs, where the modules are the full adder (FA), and the first- and second-level lookahead blocks. Compute the delays using the model below. The *worst case 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 + 4n)*τ, where n is the number of inputs to the gate. Thus a 2-input AND gate has delay 12τ and the logic function *f = ab + cde* has delay 28τ (2-input OR with delay 12τ plus a 3-input AND with delay 16τ).

Signal	Equation	Worst case module delay	Critical path delay
$p_3 =$	$a_3 + b_3$	12τ	12τ
$g_3 =$	$a_3 b_3$	12τ	12τ
$c_4 =$	$g_{3-0} + p_{3-0} c_0$	24τ	64τ
$c_8 =$	$g_{7-4} + p_{7-4} g_{3-0} + p_{7-4} p_{3-0} c_0$	32τ	80τ
$g_{11-8} =$	$g_{11} + p_{11} g_{10} + p_{11} p_{10} g_9 + p_{11} p_{10} p_9 g_8$	40τ	52τ
$p_{11-8} =$	$p_{11} p_{10} p_9 p_8$	20τ	32τ
$c_{10} =$	$g_9 + p_9 g_8 + p_9 p_8 c_8$	32τ	112τ
$s_{10} =$	$(a_{10}\overline{b}_{10} + \overline{a}_{10} b_{10})\overline{c}_{10} + (a_{10} b_{10} + \overline{a}_{10}\overline{b}_{10}) c_{10}$	48τ	136τ
$c_{12} =$	$g_{11-8} + p_{11-8} g_{7-4} + p_{11-8} p_{7-4} g_{3-0} + p_{11-8} p_{7-4} p_{3-0} c_0$	40τ	88τ
$c_{15} =$	$g_{14} + p_{14} g_{13} + p_{14} p_{13} g_{12} + p_{14} p_{13} p_{12} c_{12}$	40τ	128τ
$s_{15} =$	$(a_{15}\overline{b}_{15} + \overline{a}_{15} b_{15})\overline{c}_{15} + (a_{15} b_{15} + \overline{a}_{15}\overline{b}_{15}) c_{15}$	48τ	152τ

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																			
Instructions	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 $8, $1, $4			F	D	X	M	W													
and $7, $8, $4				F	~~D~~	D	X	M	W											
lw $8, 4($7)					~~F~~	F	D	X	M	W										
add $1, $2, $8							F	~~D~~	~~D~~	D	X	M	W							
sw $1, 4($7)								~~F~~	~~F~~	F	D	X	M	W						
RSmux select	X	X	0	2	1	X	1	2	X	X	0	0	X	X						
RTmux select	X	X	1	1	0	X	1	2	X	X	1	0/2	X	X						

For each cycle, specify the correct values for RSmux select and RTmux select (specify X for a “don’t care”).

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

Cycle 5: Register $4 in the ‘and’ instruction is dependent on the preceding ‘sub’ instruction. Because $r is the ‘rt’ register, it cannot forward from the MW latch. Instead, it must stall in decode until it can read it from the register file (only a single cycle), which also stalls the fetch of the ‘lw’ instruction.

Cycle 8: Register $8 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 stage, requiring a load-use stall for the ‘add’ in this cycle. This also stalls the ‘sw’ instruction in fetch.

Cycle 9: Register $8 in the second ‘add’ instruction depends upon the value produced by the ‘lw’ instruction. The value is ready in the MW pipeline latch, but there is no bypass path to get this from the MW latch to the ‘rt’ port of the alu.

Cycle 12: This datapath has a bug, where the RTmux must be set to both 0 and 2 in the same cycle. Up to 5 bonus points for identifying this problem and a possible solution.