CS/ECE 552: Introduction to Computer Architecture

Prof. David A. Wood

Midterm Exam
October 19, 2005
7:15-9:15pm, 1221 CSS
Approximate Weight: 25%

CLOSED BOOK
ONE SHEET OF NOTES

NAME: ____SOLUTIONS________

DO NOT OPEN THE EXAM UNTIL TOLD TO DO SO!

Read over the entire exam before beginning. Verify that your exam includes all 9 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	10	
Problem 3	15	
Problem 4	30	
Problem 5	25	
Total	**90**	

NAME: ______________________________

Problem 1: (10 points)

Part A: (2 points)

Define the performance metric MIPS and explain why it is not a good metric for comparing two different machines.

MIPS = Millions of Instructions per Second

This metric ignores the "instructions/program" component of the Iron Law of performance. For example, one VAX instruction may do the same work as many MIPS instructions.

Part B: (2 points)

H&P state that "More powerful instructions mean higher performance" is a fallacy. Explain why.

A powerful instruction, such as the VAX CALLS instruction, may do a lot of work. In fact, it may do more work than required by the particular compiler's subroutine convention. A sequence of simpler instructions may be able to do the required work in less time (in fact, the VMS operating system that originally ran on the VAX did not use the CALLS instruction).

Part C: (2 points)

Instructions are stored in memory, which in MIPS requires a 32 bit byte address. Why is it OK for implementations to only store 30 bits in the program counter?

MIPS instructions are always 4 bytes and aligned to a 4-byte boundary. Thus the low-order two bits are ALWAYS zero. We can optimize these bits away.

Part D: (2 points)

When is throughput not equal to 1/latency?

Throughput = 1/latency when there is no overlap between jobs (instructions).

Throughput > 1/latency when execution overlaps, such as in a pipeline

Throughput < 1/latency when there is no overlap and there is additional overhead between servicing requests (as is the case in memory controllers).

Part E: (2 points)

Which mean should be used to average rates (e.g., jobs per hour)? Briefly explain.

Harmonic mean = N / sum (i=1 to N) 1/r_i

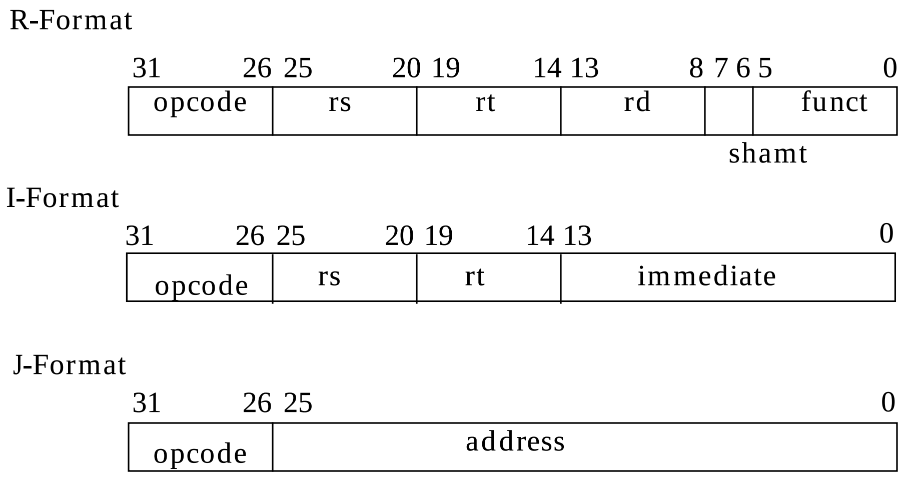
Remember that time is always what matters. Since a rate is proportional to 1/time, we want to take the recipirocal to get time, compute the arithemetic mean, and then take the recipirocal to convert back to a rate.

NAME: ______________________________

Problem 2: (10 points)

Part A: (5 points)

Most compiler writers wish the MIPS architecture had 64 or even 128 registers. Assume we want to design a new MIPS-2005 architecture with 64 registers, but preserve the general structure of the instruction formats (but not the sizes of all the fields). What might the MIPS-2005 instruction formats look like?



Part B: (5 points)

What are the most significant changes in what the new MIPS-2005 ISA can express?

Much smaller shift distance (since the shamt field shrunk from 5 to two bits)

Fewer opcodes (if you shrunk the opcode or funct field)

Smaller branch offset (since immediate field shrunk)

Smaller immediate values (since the immediate field shrunk)

Can no longer load a 32-bit immediate in 2 instructions

More registers available for the compiler, reducing loads and stores

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 multicycle datapath to reduce the cycle time to 250ps, but branch instructions take 3 cycles, load and store instructions take 5 cycles, and all other instructions take 4 cycles.

Part A: (4 points)

For the two workloads below, compute the CPI for the multicycle datapath.

Workload	% Branchs	% Load/ store	% Other	CPI_B
W1	15%	35%	50%	4.2
W2	10%	50%	40%	4.4

Part B: (5 points)

Which machine has the best performance for each workload? Show your work.

Workload	Better machine
W1	A
W2	A

Part C: (6 points)

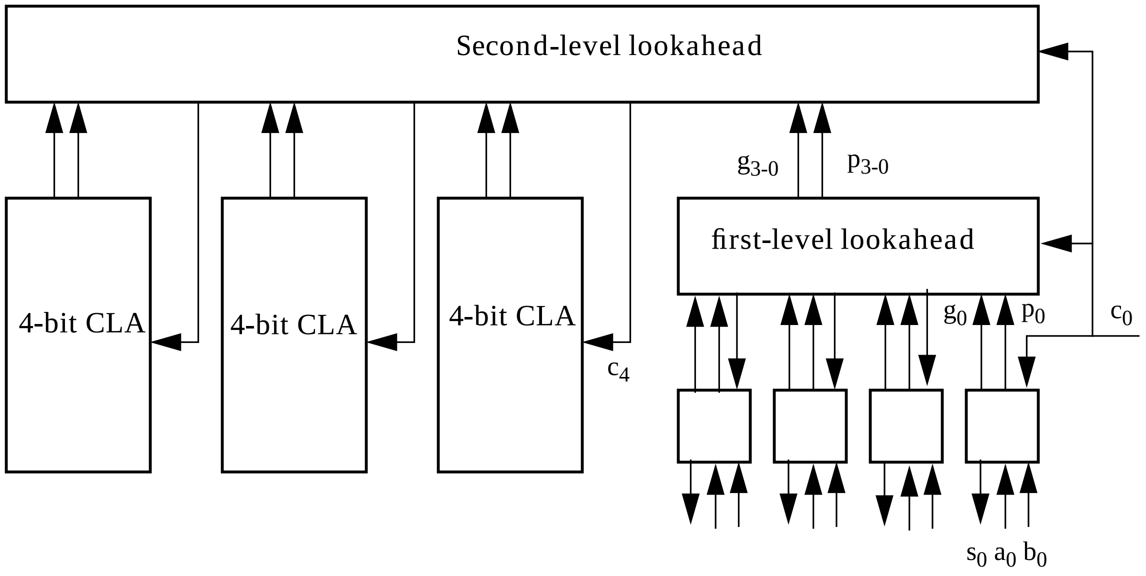
An alternative multicycle design can reduce the cycle time to 200ps, but requires increasing the delay of loads and stores to 6 cycles. Should you make this change? Does this change which machine has the best performance? Show your work.

Workload	CPI_B	Better machine
W1	4.55	new B
W2	4.9	new B

Problem 4: (30 points)

Part A: (15 points)

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



For the table below, write the boolean equation for each signal listed. Compute the delays using the model below. The *worst case delay* is the critical path from any input to that equation to the output. The *total delay* is the critical path from the basic inputs a_i and b_i, 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 $\overline{f}$. The delay is computed using the formula delay = $(8 + n^2)\tau$, 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 29τ (2-input OR with delay 12τ plus a 3-input AND with delay 17τ).

Signal	Equation	Worst case delay	Total delay
p_2	$a_2 + b_2$	12 t	12 t
c_2	$g_1 + p_1g_0 + p_1p_0c_0$	34 t	46 t
c_3	$g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$	48 t	60 t
$g_{3\text{-}0}$	$g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0$	48 t	60 t
c_8	$g_{7\text{-}4} + p_{7\text{-}4}g_{3\text{-}0} + p_{7\text{-}4}p_{3\text{-}0}c_0$	34 t	94 t
c_{12}	$g_{11\text{-}8} + p_{11\text{-}8}g_{7\text{-}4} + p_{11\text{-}8}p_{7\text{-}4}g_{3\text{-}0} + p_{11\text{-}8}p_{7\text{-}4}p_{3\text{-}0}c_0$	48 t	108 t
s_2	$(a_2\overline{b}_2 + \overline{a}_2b_2)\overline{c}_2 + (\overline{a}_2\overline{b}_2 + a_2b_2)c_2$	24 t	70 t
s_8	$(a_8\overline{b}_8 + \overline{a}_8b_8)\overline{c}_8 + (\overline{a}_8\overline{b}_8 + a_8b_8)c_8$	24 t	118 t
s_{14}	$(a_{14}\overline{b}_{14} + \overline{a}_{14}b_{14})\overline{c}_{214} + (\overline{a}_{14}\overline{b}_{14} + a_{14}b_{14})c_{14}$	24 t	166 t

Part B: (5 points)

Suppose the delay model was changed to be delay = $(1+n^2)\tau$. Would this change the optimal way to structure the carry-lookahead adder? Explain.

YES!

Three two-input gates in a tree, e.g., f = (ab)(cd), would have delay 10, while a 4-input gate, e.g., f = abcd, would have delay 17. Thus it would be best to use two input gates rather than flatten into a two-level hierarchy. Specifically,

c3 = g2 + p2(g1 + p1(g0 + p0 c0)) will have delay 30, while

c2 = g2 + p2g1 + p2p1g0 + p2p1p0c0 will have delay 34

Part C: (5 points)

Overflow occurs when the result of an arithmetic operation cannot be represented. Consider the subtraction of two two's complement numbers:

S<31:0> = A<31:0> - B<31:0>

Write the boolean equation for detecting overflow.

$\text{Out1} = \overline{A}\text{<31>}B\text{<31>}S\text{<31>} + A\text{<31>}\overline{B}\text{<31>}\overline{S}\text{<31>}$

Part D: (5 points)

An alternative way to organize a carry-lookahead adder is to use propagate and kill, rather than propagate and generate. The kill signal k_i is defined to be one if and only if the carry out of bit i will be zero regardless of the carry in. Assume the inputs are a_i, b_i, and the carry in c_i. Write the boolean equations for propagate and kill signals p_i and k_i in terms of the inputs a_i and b_i. Write the boolean equations for each of the outputs sum s_i and carry out c_{i+1} in terms of p_i, k_i, and c_i.

$k_i = \overline{a_i}\overline{b_i}$

$p_i = a_i \text{ xor } b_i$

$s_i = p_i \text{ xor } c_i$

$c_{i+1} = (c_i + \overline{p_i})\,\overline{k_i}$

NAME: ____________________________________

Problem 5: (25 points)

Your single-cycle processor seems to be executing random instructions. You need to find out why. On the next page is a picture of your datapath (note that this is somewhat different from the datapath used in class) and the control table is below. You suspect that the controller is broken. You may assume that the datapath modules (e.g., the ALU, etc.) work correctly.

Opcode	PCSrc	Bequal	RegDst	Reg Wr	ExtOp	ALUsrc	ALUCtr	MemWr	Mem ToReg
addu	0	0	0	1	1	X	0	0	0
subu	0	0	1	1	X	0	0	0	0
ori	0	0	1	1	0	X	2	0	0
lw	0	0	1	1	1	1	0	1	1
sw	0	0	X	0	0	1	0	1	X
beq	0	1	X	0	X	0	3	0	X
jr	2	1	X	0	X	X	X	0	X

You may assume the following are correct:

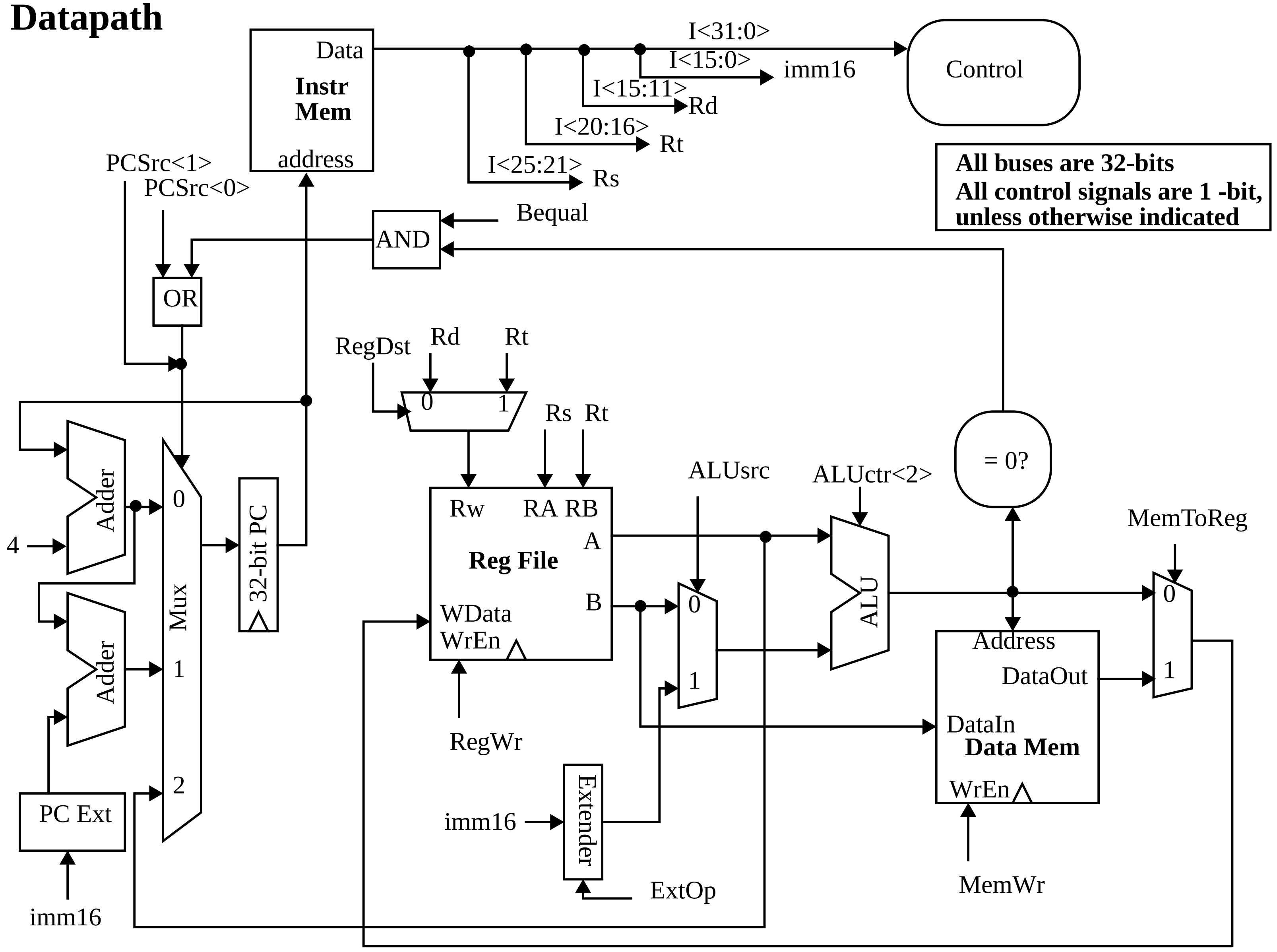
- The register file and memory both write on the rising clock edge when their respective control signals, RegWr and MemWr, are asserted.
- The extender with zero extend if the ExtOp bit is 0 and sign extend when the ExtOp bit is 1.
- The data memory reads asynchronously but has synchronous writes.
- The =0? module will output 1 if all the input bits are 0, and will output 0 otherwise.

The ALUctr encoding is as follows:

Control bits	Operation
0	add
1	sub
2	or
3	xor

For the following stream of instructions, what does your broken processor actually do? The first instruction has already been done for you as an example. If there is more than one possibility, please list all of them (note that this may be a different instruction, correct behavior, or an undefined instruction). If the incorrect result does not match a valid MIPS instruction, please give a sequence of MIPS instructions that correspond to the behavior. Also give a very brief explaintation of your possibilities. For simplicity, we have used the actual register numbers rather than names.

Datapath



All buses are 32-bits
All control signals are 1 -bit, unless otherwise indicated

NAME: ______________________________

Original Instruction	Possible behavior(s)
addu $1, $2, $0	addu $1, $2, $0 (if aluSrc = 0 —Correct behavior) addiu $1, $2, 33 (is aluSrc = 1 — Incorrect behavior)
subu $4, $5, $6	addu $6, $5, $6 (incorrect behavior, both regDst and ALUctr are wrong)
ori $7, $8, 0x0025	or $7, $8, $7 (if AluSrc = 0, incorrect behavior) ori $7, $8, 0x0025 (if AluSrc = 1, correct behavior)
beq $11, $12, 24	correct behavior (even though AluCtr is xor, it still works since we are testing for equality)
sw $10, -12($31)	sw $10, 0x0000FFF4($31) (ExtOp is wrong, so displacement is zero extended instead of sign extended -- incorrect behavior)
lw $9, -16($29)	lw $9, -16($29) and sw $9, -16($29) (Because MemWr is asserted, a load AND a store are performed in the same cycle. This is effectively an atomic swap instruction -- incorrect behavior).
EXTRA CREDIT: **jr $9**	jr $9 (most of the time this operates correctly, but depending upon the control signals and the data values in the registers, it may be undefined because it will select the non-existent port 3 of the mux. For example, if AluSrc=0, ALUctr=3, and $9=0, then it will read out register $0 on the A port (which is 0) and compare to $9, resulting in the condition being true and undefined behavior.)