CS 351 Exam 2 Wed. 4/2/2014

Name: _____

Rules and Hints

- You may use one handwritten 8.5 × 11" cheat sheet (front and back). This is the only additional resource you may consult during this exam. *No calculators.*
- Include step-by-step explanations and comments in your answers, and show as much of your work as possible, in order to maximize your partial credit.
- You may use the backs of these pages if you need more space, but make it clear where to find your answer to each question.
- You may write your answers in the form *[mathematical expression][units]*. There is no need to actually do the arithmetic.

Grade

	Your Score	**Max Score**
Problem 1: Tracing the MIPS datapath		25
Problem 2: Expanding the MIPS datapath		20
Problem 3: Pipelining and performance		30
Problem 4: AMAT and caches		25
Total		**100**

Problem 1: Tracing the MIPS datapath (25 points)

Answer the following questions about the single-cycle datapath provided on the second-to-last page of this exam. This is the datapath that executes a single instruction, from start to finish, on every clock cycle. Be sure to explain your answers if you want to receive partial credit.

Consider the execution of this instruction: `SUB $t0, $s0, $s1`

Assume the following:

- The bits of this instruction are stored in addresses 950–953 in memory.
- The initial values in $t0, $s0, and $s1 are 100, 50, and 7, respectively.

Provide the exact numerical answer to each question if possible. If not, describe the value without stating an exact number.

Part A: Instruction memory (4 points)

What value goes into the instruction memory's *Read Address* port?

What is the value of the instruction memory's *Instruction[31:0]* output?

Part B: Register file (7 points)

What values go into the register file's *Read Data 1* and *Read Data 2* ports?

What values go into the register file's *Write Register* and *Write Data* ports?

What is the value of the *RegWrite* control signal?

Part C: ALU (7 points)

What are the values of the ALU's two data inputs?

What two values go into the ALU control unit?

What are the values of the ALU's *ALU result* and *Zero* outputs?

Part D: Branch target adder (7 points)

What two values are input to the top right adder?

What is the output of the top right adder?

What is the output of the top right mux? Conceptually, what is this mux choosing between?

Problem 2: Expanding the MIPS datapath (20 points)

In x86, when you return from a function, the return address is popped from the stack instead of read from a register. Imagine that MIPS implemented a similar `ret` instruction. Here is how this instruction would work:

- It would set the next PC to the value that is currently at the top of the stack.
- It would add 4 to the stack pointer.

We want to modify the MIPS datapath to accommodate this instruction.

Part A: Register file (6 points)

Which register(s), if any, would need to be *read* by this instruction? Explain.

Which register(s), if any, would need to be *written* by this instruction? Where would the data come from? Explain.

What should be the value of the *RegWrite* control signal?

Part B: ALU (4 points)

What operation, if any, should the ALU do this instruction (e.g. add, subtract)? Explain.

What would the ALU's operands need to be?

Part C: Instruction encoding and additional logic (10 points)

Assume that the opcode is fixed, but you can encode the remaining bits of the instruction however you want. Explain how you would choose to encode them. Hint: choose an encoding that will make the rest of this problem as easy as possible.

Given your chosen encoding, explain any additional logic you would need to add to the register file, ALU, or target address computation. Draw any muxes you would add and clearly label their inputs and outputs, including any new control signals you would add.

How would you set the following control signals?

- MemWrite
- MemRead
- RegDst
- MemToReg
- Branch
- ALUSrc

Problem 3: Pipelining and performance (30 points)

Part A: Basic pipeline performance (10 points)

For this question, imagine a MIPS implementation whose pipeline stages all take 300 ps, except for the second stage, which takes 400 ps.

List the stages of the MIPS pipeline, and briefly explain the goal of each.

If you have a sequence of 1000 independent ADD instructions...

- How many *cycles* will it take on a single-cycle (non-pipelined) MIPS implementation?

- How many *cycles* will it take on a pipelined MIPS implementation?

Based on the information at the beginning of this question, which implementation will be faster for this code snippet, and what is its speedup over the slower implementation?

Part B: Data hazards (10 points)

Show a single sequence of MIPS instructions (*not* including branches) that meets *all* of the following criteria on a pipelined MIPS processor:

- Would take 8 cycles to execute if all of the instructions were independent and no data hazards existed.
- Takes 9 cycles on a processor with all possible forwarding paths enabled.
- Takes 10 cycles on a processor that has to stall until data dependencies are resolved through the register file.

Show pipeline diagrams to explain your answers.

Part C: Branch prediction (10 points)

Consider the following three branch prediction options:

- Predict not taken
- 1-bit predictor
- 2-bit predictor

You have an inner loop with the following pattern: T, T, T, T, NT, NT, T, T, NT, NT.

What are the *long-term* accuracies of each of these 3 predictors for this loop pattern? If the long-term accuracy of one of the predictors depends on its initial state, explain.

Problem 4: AMAT and caches (25 points)

A sequential program X executing on a processor P has the following memory access statistics:

- Separate L1 instruction and data caches. Both have access times of 1 cycle. The instruction cache has a 96% hit rate, and the data cache has a 90% hit rate.
- A unified L2 cache for both instructions and data, with an access time of 6 cycles and an 80% hit rate.
- A main memory with an access time of 90 cycles.

Part A: AMAT (10 points)

What is the average memory access time, in cycles, of an instruction fetch operation?

What is the average memory access time, in cycles, of a data load or store operation?

Part B: CPI (10 points)

If 25% of instructions are loads or stores, how many memory accesses does the average instruction make?

Assume that the base CPI for this program includes the access times for the L1 instruction and data caches. How many cycles do memory access *stalls* add to the average CPI?

Part C: Locality (5 points)

Explain why accesses to the *instruction cache* are likely to exhibit spatial and temporal locality. Your answer should:

- Be specific to instruction accesses and not memory accesses in general
- Demonstrate that you know what spatial and temporal locality are, and how to identify them in code.

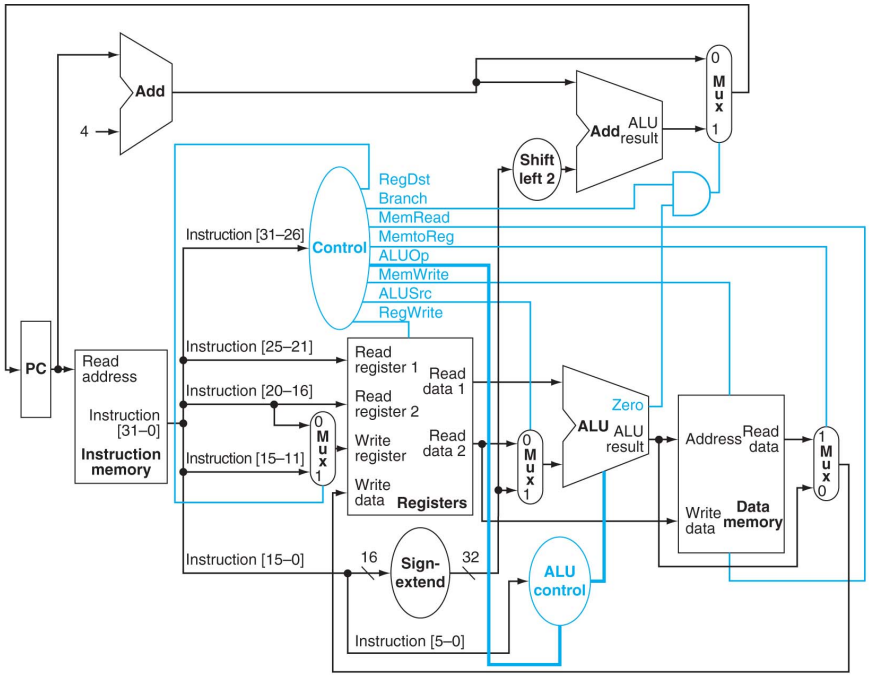

FIGURE 4.17 The simple datapath with the control unit. The input to the control unit is the 6-bit opcode field from the instruction. The outputs of the control unit consist of three 1-bit signals that are used to control multiplexors (RegDst, ALUSrc, and MemtoReg), three signals for con trolling reads and writes in the register file and data memory (RegWrite, MemRead, and MemWrite), a 1-bit signal used in determining whether to possibly branch (Branch), and a 2-bit control signal for the ALU (ALUOp). An AND gate is used to combine the branch control signal and the Zero output from the ALU; the AND gate output controls the selection of the next PC. Notice that PCSrc is now a derived signal, rather than one coming directly from the control unit. Thus, we drop the signal name in subsequent figures. Copyright © 2009 Elsevier, Inc. All rights reserved.

MIPS Arithmetic Instructions

Instruction	Operation	Fmt	Opcode	Funct
ADD $rd, $rs, $rt	reg[rd] = reg[rs] + reg[rt]	R	0	0x20
ADDI $rt, $rs, imm	reg[rt] = reg[rs] + SignExt(imm)	I	0x08	
ADDU $rd, $rs, $rt	reg[rd] = reg[rs] + reg[rt]	R	0	0x21
ADDIU $rt, $rs, imm	reg[rt] = reg[rs] + SignExt(imm)	I	0x09	
SUB $rd, $rs, $rt	reg[rd] = reg[rs] - reg[rt]	R	0	0x22
SUBU $rd, $rs, $rt	reg[rd] = reg[rs] - reg[rt]	R	0	0x23
LUI $rt, imm	reg[rt] = (imm << 16) | 0	I	0x0f	

The signed instructions (ADD, ADDI, SUB) can cause overflow exceptions.

MIPS Logical Instructions

Instruction	Operation	Fmt	Opcode	Funct
AND $rd, $rs, $rt	reg[rd] = reg[rs] & reg[rt]	R	0	0x24
ANDI $rt, $rs, imm	reg[rt] = reg[rs] & ZeroExt(imm)	I	0x0c	
NOR $rd, $rs, $rt	reg[rd] = ~(reg[rs] | reg[rt])	R	0	0x27
OR $rd, $rs, $rt	reg[rd] = reg[rs] | reg[rt]	R	0	0x25
ORI $rt, $rs, imm	reg[rt] = reg[rs] | ZeroExt(imm)	I	0x0d	
SLL $rd, $rt, shamt	reg[rd] = reg[rt] << shamt	R	0	0x00
SRL $rd, $rt, shamt	reg[rd] = reg[rt] >> shamt	R	0	0x02

The SLL and SRL instructions fill in the "shifted-out" bits with 0.

MIPS Branch and Jump Instructions

Instruction	Operation	Fmt	Opcode	Funct
BEQ $rs, $rt, label	if (reg[rs] == reg[rt]) PC=BrAddr	I	0x04	
BNE $rs, $rt, label	if (reg[rs] != reg[rt]) PC=BrAddr	I	0x05	
J label	PC = JumpAddr	J	0x02	
JAL label	$ra = PC+4; PC = JumpAddr	J	0x03	
JR $rs	PC = reg[$rs]	R	0	0x08

BrAddr = PC + 4 + SignExt (imm << 2)
JumpAddr = (PC+4)[31:28] | (26-bit imm << 2)

MIPS Memory Access Instructions

Instruction	Operation	Fmt	Opcode	Funct
LW $rt, imm($rs)	reg[rt] = Mem[rs + SignExt(imm)]	I	0x23	
SW $rt, imm($rs)	Mem[rs + SignExt(imm)] = reg[rt]	I	0x2b	
LH $rt, imm($rs)	Loads 16b from memory	I	0x21	
LHU $rt, imm($rs)	Loads 16b from memory	I	0x25	
SH $rt, imm($rs)	Stores 16b to memory	I	0x29	
LB $rt, imm($rs)	Loads 8b from memory	I	0x20	
LBU $rt, imm($rs)	Loads 8b from memory	I	0x24	
SB $rt, imm($rs)	Stores 8b to memory	I	0x28	

LH and LB sign-extend the values read from memory in order to fill the leftmost bits of the 32-bit register. LHU/LBU zero-extend.

MIPS Comparison Instructions

Instruction	Operation	Fmt	Opcode	Funct
SLT $rd, $rs, $rt	reg[rd] = ($rs < $rt)	R	0	0x2a
SLTI $rt, $rs, imm	reg[rt] = ($rs < SignExt(imm))	I	0x0a	
SLTU $rd, $rs, $rt	reg[rd] = ($rs < $rt)	R	0	0x2b
SLTIU $rt, $rs, imm	reg[rt] = ($rs < SignExt(imm))	I	0x0b	

Instruction Formats

R-format

6b: opcode	5b: rs	5b: rt	5b: rd	5b: shamt	6b: funct

I-format

6b: opcode	5b: rs	5b: rt	16b: immediate operand

J-format

6b: opcode	26b: JumpAddr[28:2]

Register List

Name	Number	Use	Preserved across function call?
$zero	0	Constant value 0	Y
$at	1	Assembler temporary	N
$v0-$v1	2-3	Function return values	N
$a0-$a3	4-7	Function arguments	N
$t0-$t7	8-15	Temporary values	N
$s0-$s7	16-23	Saved values	Y
$t8-$t9	24-25	More temporary values	N
$k0-$k1	26-27	Reserved for OS kernel	N
$gp	28	Global (heap) pointer	Y
$sp	29	Stack pointer	Y
$fp	30	Frame pointer	Y
$ra	31	Return address	Y