CS 351 Exam 2 Mon. 4/4/2016

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.*
- You may write your answers in the form *[mathematical expression][units]*. There is no need to actually do the arithmetic.
- You may use extra scratch paper if you need more space, but make it clear where to find your answer to each question.

Grade

	Your Score	**Max Score**
Problem 1: Tracing the MIPS datapath		35
Problem 2: Pipeline performance		25
Problem 3: Data hazards and forwarding		20
Problem 4: Control hazards and branch prediction		20
Total		**100**

Problem 1: Tracing the MIPS datapath (35 points)

Answer the following questions about the single-cycle datapath provided at the end 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.

Provide the exact numeric answer (in decimal, hex, or binary) to each question if possible. If not, describe the value without stating an exact number.

Consider the execution of this instruction: `LW $v0, 0($t0)`

Assume the following:

- The bits of this instruction are stored in addresses 1000–1003 in memory.
- The initial value in $t0 is 1000.
- The initial value in $v0 is 12.
- The main control unit will set ALUOp to 00 for "add", 01 for "subtract", and 10 for "look at the function field."

Part A (1 point)

+1 extra credit point if entire class gets it right.

But before we begin: how many bits are in a byte?

Part B: Instruction memory (4 points)

What value goes into the instruction memory's *Read Address* port? How many bits is that value?

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

Part C: Register file (7 points)

What values go into the register file's *Read Register 1* and *Read Register 2* ports? How many bits are those values?

What values go into the register file's *Write Register* and *Write Data* ports? How many bits are those values?

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

Part D: ALU (8 points)

What are the values of the ALU's two data inputs? How many bits are those values?

What two values go into the ALU control unit? How many bits are those values?

What are the values of the ALU's *ALU result* and *Zero* outputs? How many bits are those values?

Part E: Branch target adder (5 points)

What two values are input to the top right adder? How many bits are those values?

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?

Part F: Modifying the datapath (10 points)

How would you need to modify this datapath to support the SLL instruction? Assume that the main ALU is capable of doing a shift operation. Draw any new hardware you would need, and explain where it should go:

How would you set the following control signals to support SLL?

- RegWrite
- MemRead
- MemWrite
- RegDst
- ALUSrc
- ALUOp
- MemtoReg
- Branch

Problem 2: Pipeline performance (25 points)

Part A: CPI (5 points)

Given a large number of instructions, what is the best-case CPI of a non-pipelined implementation, such as the one provided at the end of this exam?

What about a pipelined implementation?

Part B: Performance (20 points)

Assume that you have a MIPS processor whose stages are shown in the table:

IF	300 ps
ID	310 ps
EX	290 ps
MEM	300 ps
WB	180 ps

How long (in cycles) will a non-pipelined processor take to execute a million instructions on this datapath?

What about a pipelined implementation?

How long (in seconds) will a non-pipelined processor take to execute a million instructions on this datapath?

What about a pipelined implementation?

What is the approximate speedup of the faster one over the slower one?

Problem 3: Data hazards and forwarding (20 points)

Consider the MIPS code:

```
ADD $t0, $t0, 1
LW $t1, 0($t0)
OR $t2, $t1, $t3
```

Part A: Dependencies (4 points)

List the read-after-write dependencies in this code. For each one, state the two instructions and the register that are involved.

Part B: Stalls (8 points)

Draw a pipeline diagram to show the execution of this code, assuming that dependent instructions are stalled until they can read values from the register file.

Part C: Forwarding (8 points)

Draw a pipeline diagram to show the execution of this code, assuming that values are forwarded whenever possible. Whenever a value is forwarded, state which forwarding path is used (e.g. EX-EX).

Problem 4: Control hazards and branch prediction (20 points)

Part A: Pipeline (4 points)

In what pipeline stage does branch prediction happen? What information does the predictor have about the current instruction at that time?

Part B: Prediction accuracy (4 points)

Consider a branch that repeats the following pattern: NT, NT, NT, T, T, NT, T

What is the accuracy of predict-not-taken for this branch?

What is the long-term accuracy of a 1-bit predictor?

What is the long-term accuracy of a 2-bit predictor? Does it depend on the initial state?

Consider the following predictor, whose state table is given below. You may want to use this information to draw a state diagram.

Current State	Output	Next State if T	Next State if NT
A	NT	B	A
B	T	C	A
C	T	C	B

What is the long-term accuracy of this predictor for this branch pattern, assuming that it is initialized to State A if the branch has never been encountered before?

[intentionally left blank]

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

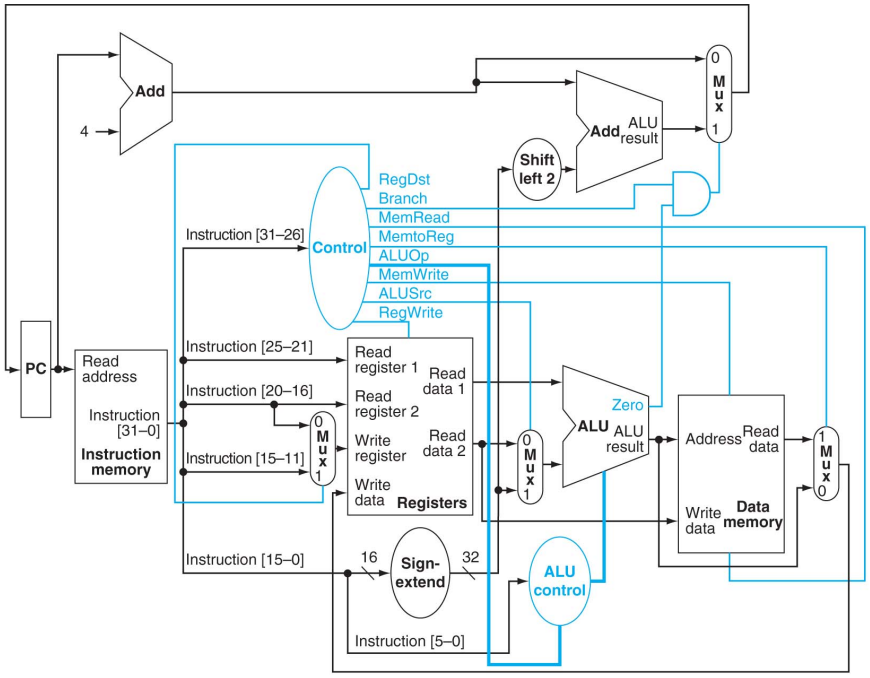

FIGURE 4.17 The simple datapath with the control unit. The input to the control unit is the 6-bit opcode field from the instruction. The outputs of the control unit consist of three 1-bit signals that are used to control multiplexors (RegDst, ALUSrc, and MemtoReg), three signals for con trolling reads 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.