CS151B/EE M116C
Computer Systems Architecture
Spring 2019 Midterm Exam II Solution
Instructor: Prof. Lei He

It is a closed-book exam.
There are total TWELVE pages including this cover page. Check whether you have all pages. If not, let the TA know right now.
Good luck!

Problem 1: _____ of 18 points
Problem 2: _____ of 20 points
Problem 3: _____ of 10 points
Problem 4: _____ of 20 points
Problem 5: _____ of 24 points
Total: _____ of 92 points

Problem 1. (18 points in total, 3 points for each)

Explain terms or answer short problems.

(1) Explain the concept of speculation in the scene of pipelining, and explain how it works by using ONE example.

Guess instruction/data to be fetched when it is not decided yet.
beq rs, rt, there
add r1, r2, r3
there: add r3, r2, r1
Do add r1, r2, r3 when branching result is not settled, and flush the results if wrong.

(2) Explain the concept of loop unrolling and why we perform loop unrolling

Unroll the loop n times and rename the registers to make a big loop. Loop unrolling leads to more instructional level parallelism and therefore improve performance.

(3) Explain the concept of super pipelining, and why it improves performance.

Deep pipeline with more stages. Reduce the clock cycle time.

(4) Name three techniques **in software** to resolve or relieve data hazards.

Insert nops, renaming, unrolling loops

(5) Name three techniques (in either software or hardware) to resolve or relieve branch hazards.

stall, early detection, speculation

(6) Explain the procedures performed by a 5-stage pipelined processor when stalling an instruction for resolving data hazard.

1. For IF and ID stages, PCWrite = 0.
2. IF/ID.write = 0 for IF/ID state registers.
3. Set 0 for control signals from IF, ID stages to EX, MEM, WB stages.

Problem 2 (20 points):

we examine how data dependences affect execution in the basic 5-stage pipeline. Consider the following sequence of instructions:

lw $5, -16($5)
sw $5, -16($5)
add $5, $5, $5

Also, assume the CPU cycle time related to the forwarding as shown below:

Without forwarding	With full forwarding	With ALU-ALU forwarding
220 ps	240 ps	230 ps

(1) Assume there is no forwarding in this pipelined processor. Indicate hazards and add NOP instructions to eliminate them. Show your answer with the new sequence of instructions.

lw $5, -16($5)
Nop
Nop
sw $5, -16($5)
add $5, $5, $5

(2) Add NOP instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage). Draw the pipeline diagram to answer.

lw $5, -16($5)
Nop
Nop
sw $5, -16($5)
add $5, $5, $5

(3) What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

No forwarding: (5+4)*220ps = 1980ps
ALU-ALU forwarding: (5+4)*230ps = 2070ps
Speedup = 1980/2070 = 0.96

(4) Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them. Draw the pipeline diagram to answer.

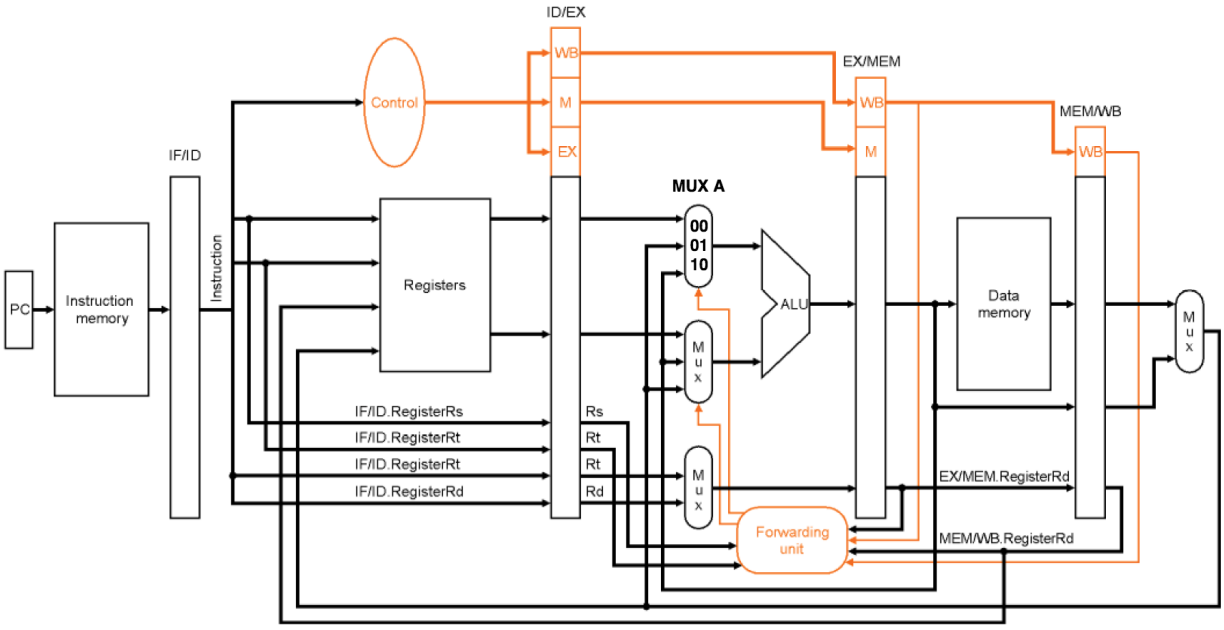
lw $5, -16($5)
Nop
sw $5, -16($5)
add $5, $5, $5

(5) What is the total execution time of this instruction sequence WITHOUT any forwarding and WITH full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

No forwarding: (5+4)*220ps = 1980ps
FULL forwarding: (4+4)*240ps = 1920ps
Speedup = 1980/1920 = 1.03

Problem 3 (10 points)

Considering data forwarding for the pipeline below, write down the Register-transfer level (RTL) model for the control signal of MUX A. Use plain English and logic function to explain when the control signal for MUX A should be 01 and 10 respectively. Represent control/register states of each instruction by specifying its intermediate register and control/register name. ***For example, branch control bit of an instruction between the ID and EX state can be represented as ID/EX.branch.***



Control signal = 01

Forward from MEM/WB register
1. MEM/WB.RegWrite
2. IF/ID.RegisterRs != EX/MEM.RegisterRd && EX/MEM.RegWrite
3. ID/EX.RegisterRs == MEM/WB.RegisterRd
4. ID/EX.RegisterRs != 0

Control signal = 10

Forward from EX/MEM register
1. EX/MEM.RegWrite
2. ID/EX.RegisterRs == EX/MEM.RegisterRd
3. ID/EX.RegisterRs != 0

Problem 4 (20 points):

Consider the two-issue superscalar processor we covered in class. It has a five stage pipeline where we can issue one ALU/branch instruction and one load/store instruction every cycle.

Suppose the branch delay penalty is two clock cycles (or say, the branch hazard is resolved in MEM stage), and it uses the full forwarding. How long would the following code take to execute on this processor assuming the loop is executed 200 times? Assume the pipeline is initially empty.

```
Loop:   lw $t0, 0 ($s0)
        lw $t1, 0 ($t0)
        add $t1, $s1, $t1
        sw $t1, 0 ($t0)     # you may assume that this store never goes to the same address as the first load
        addi $s0, $s0, 4
        bne $s0, $s2, Loop
```

(1) Suppose the loop is not unrolled for scheduling. Schedule the code (use the table below) to minimize the total number of cycles required. **(8 points)**

Total # of cycles for 200 iterations: _____**1204**_____

(Hint – schedule the code first for one iteration, then figure out how long it will take the processor to run 200 iterations of this scheduled code)

Cycle	1st Issue Slot (ALU or Branch)	2nd Issue Slot (LW or SW)
1	addi $s0, $s0, 4	lw $t0, 0($s0)
2		
3		lw $t1, 0($t0)
4	bne $s0, $s2, loop	
5	add $t1, $s1, $t1	
6		sw $t1, 0($t0)
7		
8		
9		
10		
11		
12		
13		

(2) Now unroll the loop once to make two copies of the loop body.
Schedule it again to minimize the total number of cycles required. **(12 points)**

Total # of cycles for 200 iterations: _____**804**_____

Cycle	1st Issue Slot (ALU or Branch)	2nd Issue Slot (LW or SW)
1	addi $s0, $s0, 8	lw $t0, 0($s0)
2		lw $t2, -4($s0)
3		lw $t1, 0($t0)
4		lw $t3, 4($t2)
5	add $t1, $s1, $t1	
6	bne $s0, $s2, loop	sw $t1, 0($t0)
7	add $t3, $s1, $t3	
8		sw $t3, 0($t2)
9		
10		
11		
12		
13		

Problem 5 (24 points):

Consider the single-cycle processor implementation. Your task will be to augment this data path with a new instruction: the cai instruction (conditional assign immediate). This instruction will be an I-type instruction, and will have the following effect:

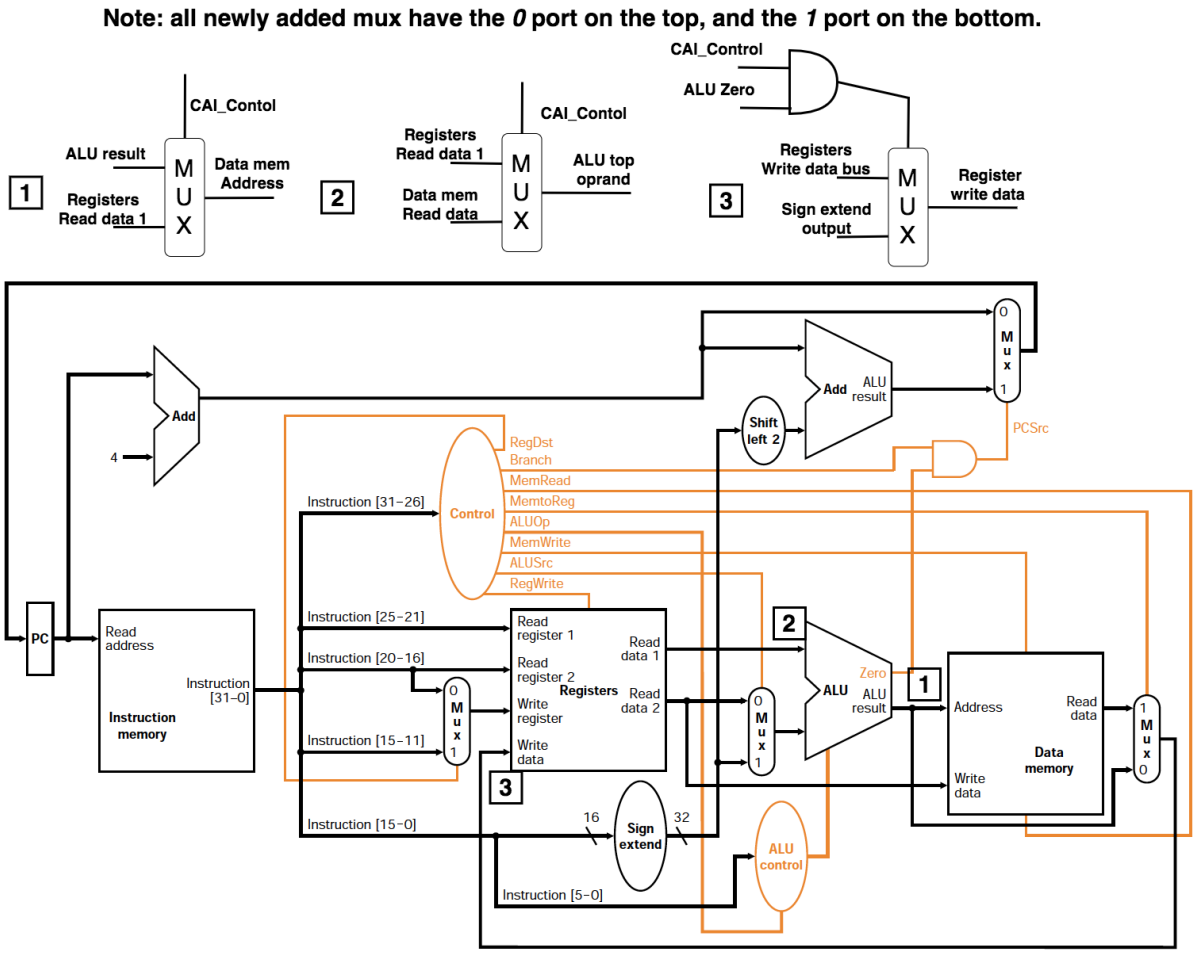
```
if (MEM[R[rs]] == R[rt])
        R[rt] = SE(I)
else
        R[rt] = MEM[R[rs]] - R[rt]
```

Note: MEM represent for data memory. SE represent for sign extension. R represent for register.

(1) **Complete the main controller control table**. Fill in with 1, 0, or X for does not matter). **(8 points)**

	R-type	lw	sw	beq
Opcode	00 0000	10 0011	10 1011	00 0100
RegDst	1	0	X	X
Branch	0	0	0	1
MemRead	0	1	0	0
MemtoReg	0	1	X	X
MemWrite	0	0	1	0
ALUSrc	0	1	1	0
RegWrite	1	1	0	0
ALUop	10	00	00	01

(2) **Implement cai on the single cycle datapath on the following two pages (for datapath and control table).** Use the I-type instruction format. All other instructions must still work correctly after your modification. You should not add new ALUs, register file ports, or ports to memory. However, simple logic gates, i.e. AND, OR, XOR gates are allowed. The number of newly added logic gates should be as small as possible. **(16 points)**



Main Controller modified as below.

	R-type	lw	sw	beq	cai
Opcode	00 0000	10 0011	10 1011	00 0100	10000 (any unused one)
RegDst	1	0	X	X	0
Branch	0	0	0	1	0
MemRead	0	1	0	0	1
MemtoReg	0	1	X	X	0
MemWrite	0	0	1	0	0
ALUSrc	0	1	1	0	0
RegWrite	1	1	0	0	1
ALUop	10	00	00	01	01
CAI_Control	0	0	0	0	1

ALU Controller should not be modified.

MIPS Reference Sheet

Arithmetic and Logical Instructions

Instruction	Operation
add $d, $s, $t	`$d = $s + $t`
addu $d, $s, $t	`$d = $s + $t`
addi $t, $s, i	`$t = $s + SE(i)`
addiu $t, $s, i	`$t = $s + SE(i)`
and $d, $s, $t	`$d = $s & $t`
andi $t, $s, i	`$t = $s & ZE(i)`
div $s, $t	`lo = $s / $t; hi = $s % $t`
divu $s, $t	`lo = $s / $t; hi = $s % $t`
mult $s, $t	`hi:lo = $s * $t`
multu $s, $t	`hi:lo = $s * $t`
nor $d, $s, $t	`$d = ~($s | $t)`
or $d, $s, $t	`$d = $s | $t`
ori $t, $s, i	`$t = $s | ZE(i)`
sll $d, $t, a	`$d = $t << a`
sllv $d, $t, $s	`$d = $t << $s`
sra $d, $t, a	`$d = $t >> a`
srav $d, $t, $s	`$d = $t >> $s`
srl $d, $t, a	`$d = $t >>> a`
srlv $d, $t, $s	`$d = $t >>> $s`
sub $d, $s, $t	`$d = $s - $t`
subu $d, $s, $t	`$d = $s - $t`
xor $d, $s, $t	`$d = $s ^ $t`
xori $d, $s, i	`$d = $s ^ ZE(i)`

Constant-Manipulating Instructions

Instruction	Operation
lhi $t, i	`HH($t) = i`
llo $t, i	`LH($t) = i`

Comparison Instructions

Instruction	Operation
slt $d, $s, $t	`$d = ($s < $t)`
sltu $d, $s, $t	`$d = ($s < $t)`
slti $t, $s, i	`$t = ($s < SE(i))`
sltiu $t, $s, i	`$t = ($s < SE(i))`

Branch Instructions

Instruction	Operation
beq $s, $t, label	`if ($s == $t) pc += i << 2`
bgtz $s, label	`if ($s > 0) pc += i << 2`
blez $s, label	`if ($s <= 0) pc += i << 2`
bne $s, $t, label	`if ($s != $t) pc += i << 2`

Jump Instructions

Instruction	Operation
j label	`pc += i << 2`
jal label	`$31 = pc; pc += i << 2`
jalr $s	`$31 = pc; pc = $s`
jr $s	`pc = $s`

Load Instructions

Instruction	Operation
lb $t, i($s)	`$t = SE (MEM [$s + i]:1)`
lbu $t, i($s)	`$t = ZE (MEM [$s + i]:1)`
lh $t, i($s)	`$t = SE (MEM [$s + i]:2)`
lhu $t, i($s)	`$t = ZE (MEM [$s + i]:2)`
lw $t, i($s)	`$t = MEM [$s + i]:4`

Store Instructions

Instruction	Operation
sb $t, i($s)	`MEM [$s + i]:1 = LB ($t)`
sh $t, i($s)	`MEM [$s + i]:2 = LH ($t)`
sw $t, i($s)	`MEM [$s + i]:4 = $t`

Data Movement Instructions

Instruction	Operation
mfhi $d	`$d = hi`
mflo $d	`$d = lo`
mthi $s	`hi = $s`
mtlo $s	`lo = $s`

Exception and Interrupt Instructions

Instruction	Operation
trap 1	Print integer value in $4
trap 5	Read integer value into $2
trap 10	Terminate program execution
trap 101	Print ASCII character in $4
trap 102	Read ASCII character into $2