EE 180 Homework 2

Schuyler Anne Tilney-Volk, Sean William Konz

TOTAL POINTS

88 / 90

QUESTION 1

1 1 9 / 10

- **0 pts** Correct
- ✓ **- 1 pts** missed exception
- **- 2 pts** missing multiple steps

QUESTION 2

2 2 4 / 5

- **0 pts** Correct
- ✓ **- 1 pts** minor error
- **- 4 pts** major error with calculations

QUESTION 3

3 3 5 / 5

- ✓ **- 0 pts** Correct

QUESTION 4

4 4 10 / 10

- ✓ **- 0 pts** Correct
- **- 5 pts** by pass ALU
- **- 1 pts** Provide more explanation
- **- 3 pts** provide additional details see solution
- **- 0 pts** Click here to replace this description.
- **- 0 pts** Click here to replace this description.

QUESTION 5

5 5 10 / 10

- ✓ **- 0 pts** Correct
- **- 4 pts** calculation error for swap instructions
- **- 1.5 pts** Case 2 for swap is wrong

QUESTION 6

6 6 15 / 15

- ✓ **- 0 pts** Correct
- **- 0 pts** Click here to replace this description.
- **- 6 pts** add mux for rs from read data 1 and pc+4
- **- 4 pts** provide more explanation see solution
- **- 5 pts** explanation not clear see solution
- **- 0 pts** connect mux for pc+4 and rs to write data and show jalrd control signal
- **- 5 pts** provide more explanation or diagram

QUESTION 7

7 7 15 / 15

- ✓ **- 0 pts** Correct
- **- 0 pts** Regwrite2 not required
- **- 2 pts** show ALU to create rs+4 and don't create another register
- **- 3 pts** no new register

QUESTION 8

8 8 20 / 20

- ✓ **- 0 pts** Correct
- ✓ **- 0 pts** look for correct signal name
- **- 2 pts** incomplete table for control signal value
- **- 1 pts** incomplete control signals for part b
- **- 3 pts** incorrect explanation or not clear
- **- 0 pts** don't need add block
- **- 3 pts** incorrect control signal values

gradescope

Sean Konz and Schuyler Tilney-Volk

HW #2

1)

```
move $t0, $zero      # i=0          a -> $a0   a_len -> $a2
lw $t1, 0($a2)       # len_a load   b -> $a1   b_len -> $a3
lw $t2, 0($a3)       # len_b load   c -> $t0   c -> $s0
                                                 sltu 1 if true
Loop:
    addu $a1, $a1, $t0       # calc b[i] location
    addu $a0, $a0, $t0       # calc a[i] location
    sltu $t3, $t2, $t0       # if b_len < i, t3 = 1   i ≤ b_len t3=0
    bne $t3, $zero, OOB      # branch if out of bounds
    lw $t4, 0($a1)           # load b[i]
    addu $t4, $t4, $s0       # t4 = b[i] + c
    sltu $t5, $t1, $t0       # if a_len < i  t5 = 1   i ≤ alen t5=0
    bne $t5, $zero, OOB      # branch if out of bounds
    sw $t4, 0($a0)           # store a[i]
    subu $a1, $a1, $t0       # get original b addr
    subu $a0, $a0, $t0       # get original a addr
    addui $t0, $t0, 16       # i = i+4
    j Loop

OOB:
```

# of instructions = 3 + 176*13 = 2,291

# of data references = 2 + 176*2 = 353

1 1 **9 / 10**

- **0 pts** Correct
- ✓ **- 1 pts** **missed exception**
- **2 pts** missing multiple steps

gradescope

Schuyler T ✓

HW #2

2) jump → 1.8% 2.4 clock
& branch → 17% 2.6 clock
arithm. → 45.7% 1 clock
load/mem → 33.8% D

$$1.5 = \frac{(1.8\%)(2.4 \text{ clock}) + (17\%)(2.6 \text{ clock}) + (45.7\%)(1 \text{ clock}) + (33.8\%)D}{(1.8 + 17 + 45.7 + 33.8)}$$

$$\boxed{D = 1.58 \text{ CPI}}$$

2 2 4 / 5

- **0 pts** Correct

✓ - **1 pts** **minor error**

- **4 pts** major error with calculations

gradescope

③ $CPI_A = 1.8$ $\quad$ $CPI_B = 2.4$

$Clock_A = 2 \times 10^9$ Hz $\quad$ $Clock_B = 2.5 \times 10^9$ Hz

100K Inst

A

$$\text{Exec. Time} = \frac{10^5 \cdot 1.8}{2 \times 10^9 \text{ Hz}} = .00009s$$

B

$$\text{Exec. Time} = \frac{X \cdot 2.4}{2.5 \times 10^9} = .00009s$$

$$\boxed{X = 93750 \text{ Instructions}}$$

∴ Computer B must be able to compile the Program with 93750 instructions.

3 3 5 / 5

✓ - 0 pts Correct

gradescope

(4) If we remove immediates from load/store instructions, then there is no need for the ALU output to be connected to the Address input of the Data Memory. This also means that memory addresses will only ever be read directly from the register file.
I would connect the output of the ALU directly to the register file, and connect the Read data 1 output of the register file directly to the read/write address input for Data Memory.

(5) swap $rs, $rt psuedoinstruction, use $at

with register available:	no register available
add $at, $rs, $zero	assume stack is free at depth d
add $rs, $rt, $zero	sw $rs, d($sp)
add $rt, $at, $zero	add $rs, $rt, $zero
	lw $rt, d($sp)

Timing Tradeoffs:

MIPS has a CPI of 1

Software implementation: CPI of 3 for swaps
Hardware: CPI of 1

Hardware implementation only makes sense if total execution time of hardware is less than total execution time of software

P = percentage of swap instructions → 1-P is percentage of all other instr

Software Time: $P \cdot 3 \cdot T + (1-P) \cdot 1 \cdot T = (2P+1)T$
Hardware: $PT + (1-P)T = T$

if clock Period increased by 12% → $T' = 1.12T$
then $1.12T < (2P+1)T \rightarrow 0.6 < P$
∴ if swap instructions make up more than 6% of instructions, then a hardware implementation is worthwhile

Period increased by 20% → $T' = .2T$
$1.2T < (2P+1)T \rightarrow .1 < P$
∴ if swap instructions make up more than 10% of instructions then a hardware implementation is worthwhile.

4 4 **10 / 10**

✓ **- 0 pts** Correct

- 5 pts by pass ALU

- 1 pts Provide more explanation

- 3 pts provide additional details see solution

- 0 pts Click here to replace this description.

- 0 pts Click here to replace this description.

gradescope

(4) If we remove immediates from load/store instructions, then there is no need for the ALU output to be connected to the Address input of the Data Memory. This also means that memory addresses will only ever be read directly from the register file.
I would connect the output of the ALU directly to the register file, and connect the Read data 1 output of the register file directly to the read/write address input for Data Memory.

(5) swap $rs, $rt psuedoinstruction, use $at

with register available:	no register available
add $at, $rs, $zero	assume stack is free at depth d
add $rs, $rt, $zero	sw $rs, d($sp)
add $rt, $at, $zero	add $rs, $rt, $zero
	lw $rt, d($sp)

Timing Tradeoffs:

MIPS has a CPI of 1

Software implementation: CPI of 3 for swaps
Hardware: CPI of 1

Hardware implementation only makes sense if total execution time of hardware is less than total execution time of software

P = percentage of swap instructions → 1-P is percentage of all other instr

Software Time: $P \cdot 3 \cdot T + (1-P) \cdot 1 \cdot T = (2P+1)T$
Hardware: $PT + (1-P)T = T$

if clock Period increased by 12% → $T' = 1.12T$
then $1.12T < (2P+1)T \rightarrow 0.6 < P$
∴ if swap instructions make up more than 6% of instructions, then a hardware implementation is worthwhile

Period increased by 20% → $T' = .2T$
$1.2T < (2P+1)T \rightarrow .1 < P$
∴ if swap instructions make up more than 10% of instructions then a hardware implementation is worthwhile.

5 5 **10 / 10**

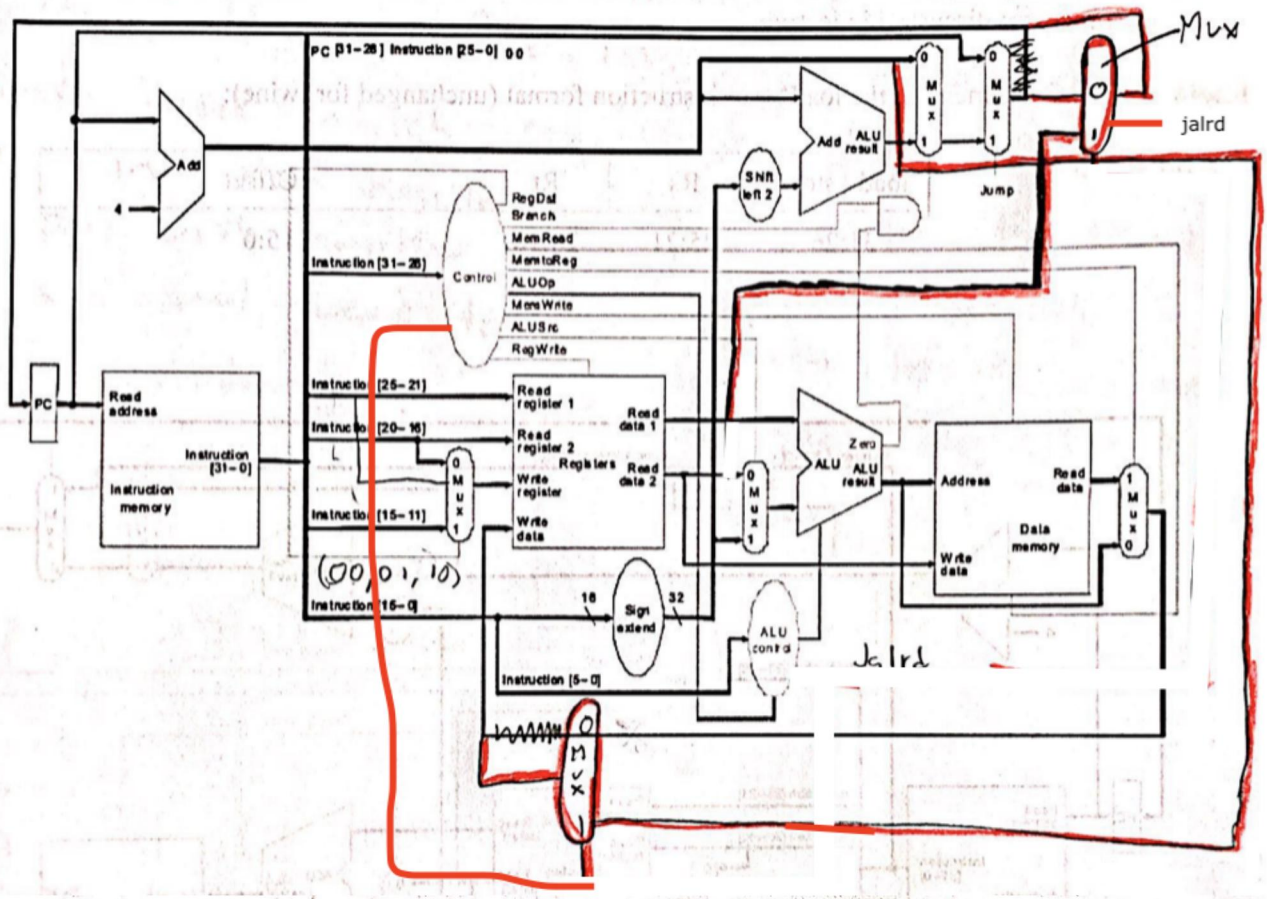
✓ **- 0 pts** Correct

- 4 pts calculation error for swap instructions

- 1.5 pts Case 2 for swap is wrong

gradescope

6. **(15 points)** Using the figure below, show all the necessary additional data and control paths to implement instruction "jalrd rd, rs" in the single-cycle MIPS processor discussed in lecture (Note: jalrd will jump to rs and store the return address in rd). Additionally, please briefly explain your additions.



Need to use the Read Data 2 output from the Reg file since is is the register to jump to, and is provided to the reg file in Read Reg 2. Output Read Data 2 is muxed with the normal instruction update, and then sent to PC. The regular update instruction address is used as the return address. This is pulled from the result of the +4 adder, and muxed with the Write Back wire and sent to the Reg file. In order to write to the rd reg, which is Bytes 25-21, we must mux the [25-21] wire with the write reg mux, assuming the use of a binary select mux.

6 6 **15 / 15**

✓ **- 0 pts** Correct

- 0 pts Click here to replace this description.

- 6 pts add mux for rs from read data 1 and pc+4

- 4 pts provide more explanation see solution

- 5 pts explanation not clear see solution

- 0 pts connect mux for pc+4 and rs to write data and show jalrd control signal

- 5 pts provide more explanation or diagram

gradescope

(7) In order to achieve this kind of behavior we need to update a register value, and write 2 registers in a single cycle. Therefore, we need an extra ALU or adder which would increment the contet of the rs register by 4 coming from Read data 1 output. The result of this operation would need to be written to a register in the reg file. This requires an additional write data input which can write to the address specified in Read Register 1. We can control the second write port with another control signal, which specifies when we want to write data to the address at Read Register 1. We call this signal RegWrite2.

7 7 **15 / 15**

✓ **- 0 pts** Correct

- 0 pts Regwrite2 not required

- 2 pts show ALU to create rs+4 and don't create another register

- 3 pts no new register

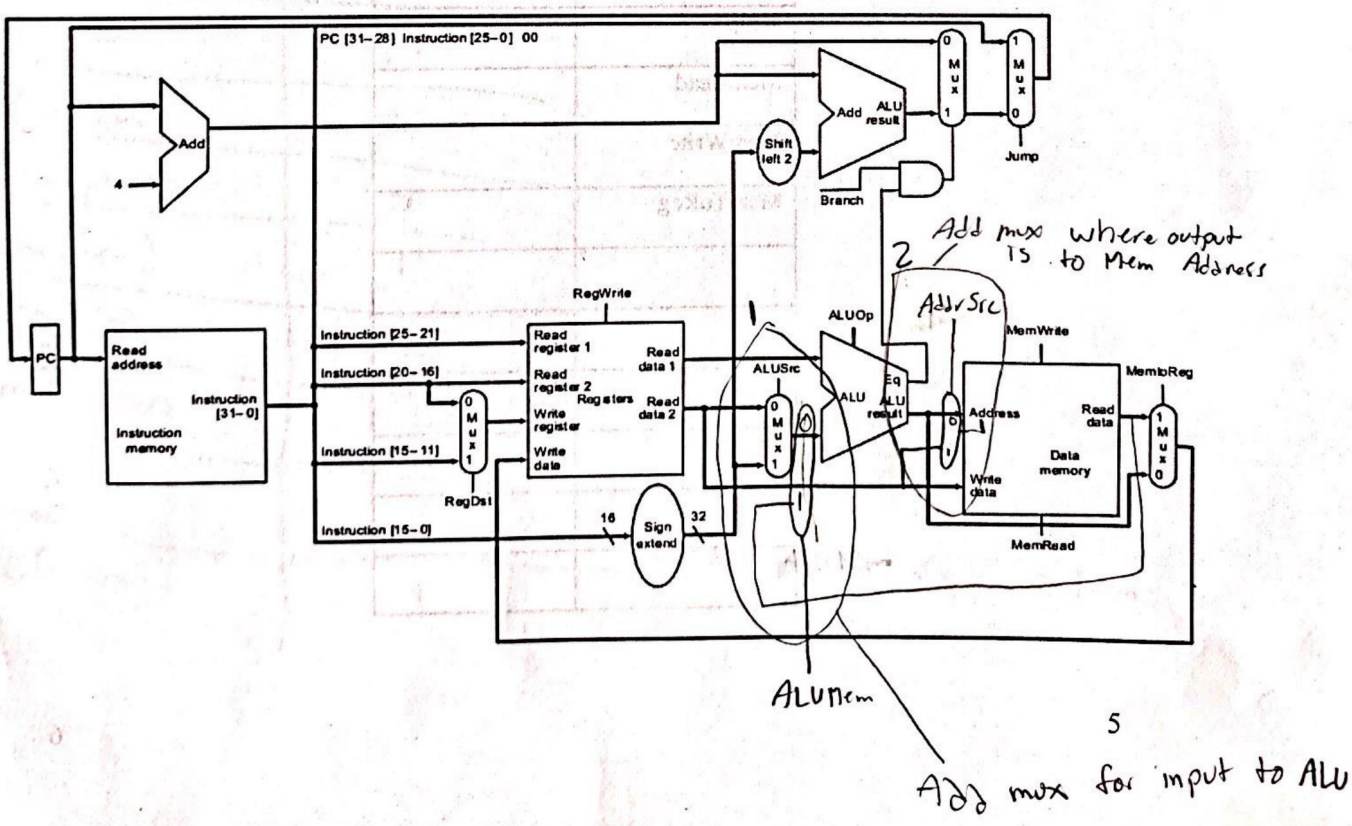
gradescope

8. **(20 Points)** The popular x86 instruction set by Intel allows arithmetic instructions to directly access memory for one of their source operands. The primary benefit is that fewer instructions will be executed because we won't have to first load that source operand into a register. The primary disadvantage is that the cycle time w i l l have to increase to account for the additional time to read memory during the arithmetic instruction.

Consider adding a new instruction to the MIPS ISA:

addm $t2, $t3, $t4 // $t2 = $t3 + Memory[$t4]

a). **(15 Points)** Consider the single-cycle MIPS processor datapath shown below. Show the datapath changes needed to implement addm. Describe each change in 1-2 sentences. Name control signals, but don't worry about their values for now.



(8)

a) Change:

1) Mux input to memory address to select which address source. 1st input (trigger is 0) is the normal ALU input, second input (trigger is 1) is directly from Read data 2, controlled by AddrSrc

2) Mux input to ALU in order to allow addition of value directly read from memory. Memory read is a direct input to the ALU here. 1st input (trigger is 0) is normal input from the existing mux. 2nd input (trigger is 1) is from the value read from memory, controlled by ALUMem.

b)

Signal	Value
RegDst	1
RegWrite	1
ALUop	add
ALUSrc	0
Branch	0
Jump	0
MemRead	1
MemWrite	0
MemToReg	0
AddrSrc	1
ALUMem	1

only 1 for addm instructions

8 8 **20 / 20**

✓ **- 0 pts** Correct

✓ **- 0 pts** look for correct signal name

- 2 pts incomplete table for control signal value

- 1 pts incomplete control signals for part b

- 3 pts incorrect explanation or not clear

- 0 pts don't need add block

- 3 pts incorrect control signal values

gradescope