ECE 369A – Fall 2023

HW2 Solutions

20 Points

Group # ?????			
Name	**Last Name**	**% Effort**	**Lab Section** **A -** 2:00-3:15pm / **B-** 3:30:4:45pm / **C-** 5:00-6:15pm
?	**?**	**?**	**?**
?	**?**	**?**	**?**
?	**?**	**?**	**?**

-2 pt per missing "Group #", "Name", "Last Name", "% Effort", "Lab Section"

Show your work in order to receive full credit.

No credit if you write the final result only (without showing how you derived it)

Grading:

- Will randomly pick a subset of the questions and scale the overall score to 20 points.
- **Show your work in order to receive full credit. If you write the final result only (without showing how you derived it), then you will receive partial credit even if the answer is correct.**

Submission:

- **Include your answers in this document and submit as a single word/pdf file.**
- 5pts per day late penalty
- 4pts penalty per skipped question
- 2pts penalty for not submitting solutions in a single file. Preferred formats are word and pdf.

Problem 1. Single-cycle Datapath (15 points): Modify the single-cycle data path given below to support an immediate type instruction called "use constant" (uc).

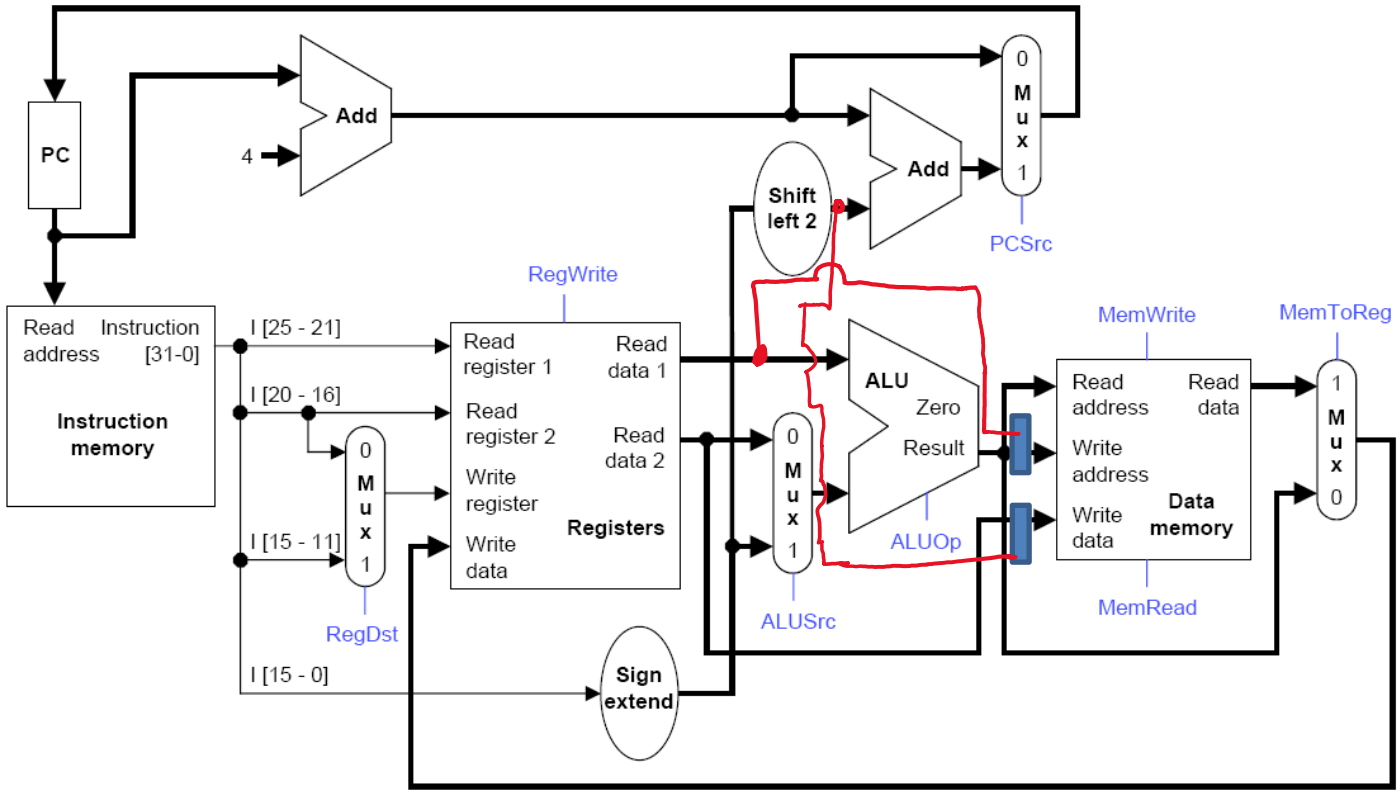
use constant **uc**

31 26	25 21	20 16	15 0
uc 110011	rs	rt	immediate

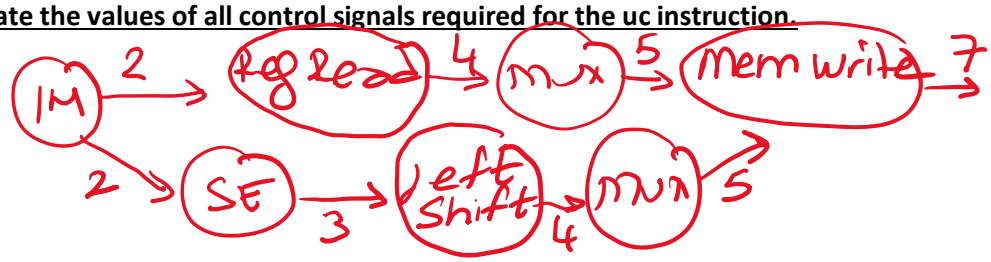
Format: `uc rs, imm # Memory[GPR[rs]] = 4*immediate`

You are allowed to add new **datapath component(s), wire(s) and/or mux(es)**. Do not modify the main functional units themselves (the memory, register file and ALU). You are **not allowed** to make a change to the "uc" specification. You can assume that any unused field in the instruction is set to 0. Try to keep your diagram neat!

Note: *While we're primarily concerned about correctness,* ***full points will only be rewarded to most efficient solution (minimum cycle time and area overhead) for the uc instruction****. Assume that the ALU, Memory, and Register file each take 2ns, and all other datapath components take 1ns.*



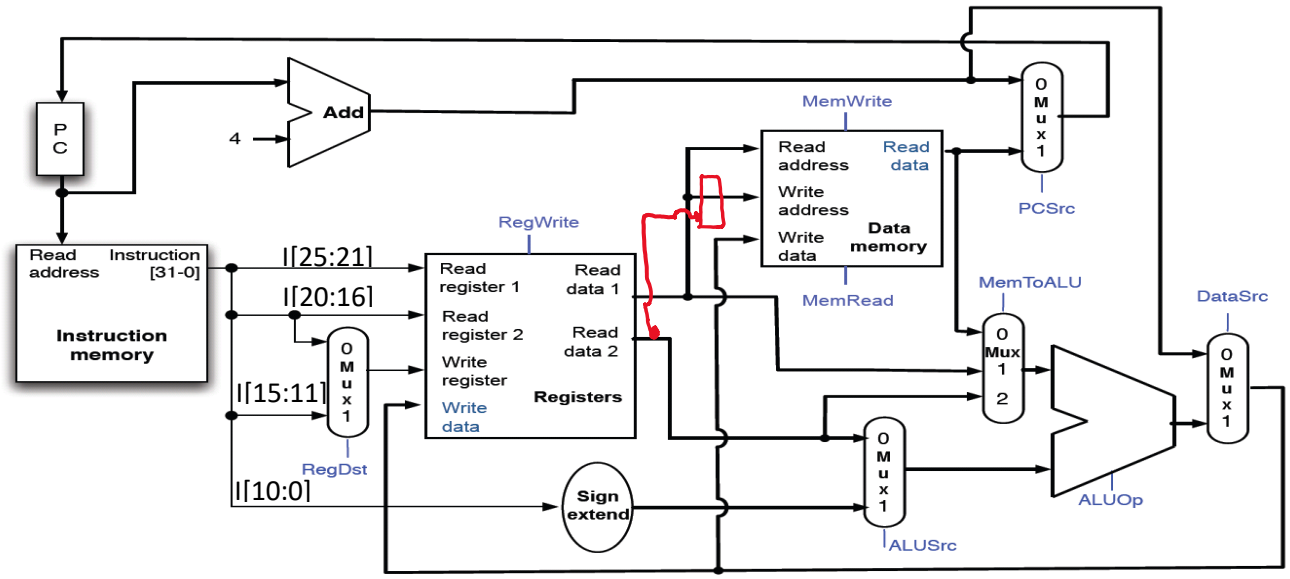
Indicate the values of all control signals required for the uc instruction.



Opcode	PCSrc	RegDst	RegWrite	ALUSrc	ALUOp	MemRead	MemWrite	MemToReg
uc	0	x	0	x	x	0	1	x

Some teams used ALU to handle addition with 0 to pass Reg[rs] tp the write address port. Correct but not optimal

Problem 2 (15pts):



Part (a) Your will implement the "ece369" instruction that will complete the process given below in one cycle.

```
ece369 $t0 $t1   # PC = Memory[GPR[t0]]
                 # Memory[GPR[t1]] = GPR[t1] + 4
                 # GPR refers to general purpose register (register file)
```

opcode I[31:26]	rs I[25:21]	rt I[20:16]	rd I[15:11]	imm I[10:0]
64	t0	t1		4

You are allowed to make changes in the "ece369" specification for utilizing the unused fields (rd and imm fields) in the instruction. Three of the instruction fields are filled for you. "opcode" is the integer value 64 for the "ece369" operation and, two source registers use "rs" and "rt" fields of "ece369 $t0 $t1" instruction. When applicable**, indicate your changes to the unused fields above with a label or value (in decimal).**

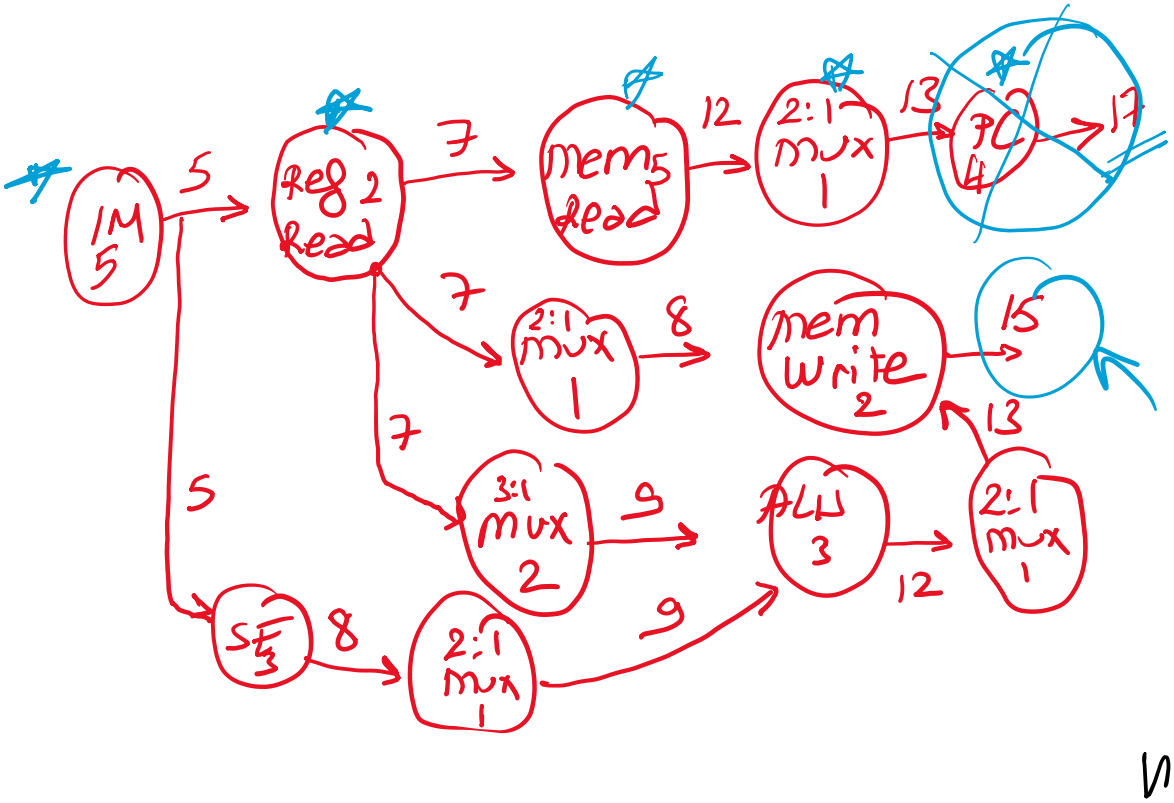
You are allowed to modify the datapath with new datapath components, wires, control signals, etc. **Show your modification(s) on the datapath given above.** While we are primarily concerned about correctness, full points will be rewarded to **solution with minimal hardware and latency overhead.** Table in the next page shows the latency for each datapath component. Adding a new input line to a mux introduces additional 1ns latency. Your modification(s) should not affect the functionally of the other types of instructions.

Finally, in the table below, indicate the value of each control signal in order to realize the "ece369" instruction. Use X for **don't care** when needed. ALUOp can be one of the following operations: **add, sub, mul, sll, and srl.** Assume that we can read and write the memory in the same cycle (like the register file, but this is likely not efficient to do in a real machine).

Instr	Reg Dst	Reg Write	ALU Src	ALU op	MemTo ALU	Mem Read	Mem Write	Data Src	PC Src
ece369	**x**	**0**	**1**	**add**	**2**	**1**	**1**	**1**	**1**

Part (b) Given the functional unit latencies, what is the **minimum cycle time** based on the "ece369" instruction? Assume PC has no delay. Show your critical path delay analysis using a graph where each node represents a function unit and each edge represents the data flow from one node to another. **Indicate the function units that are on the critical path.** Show your work to get credit.

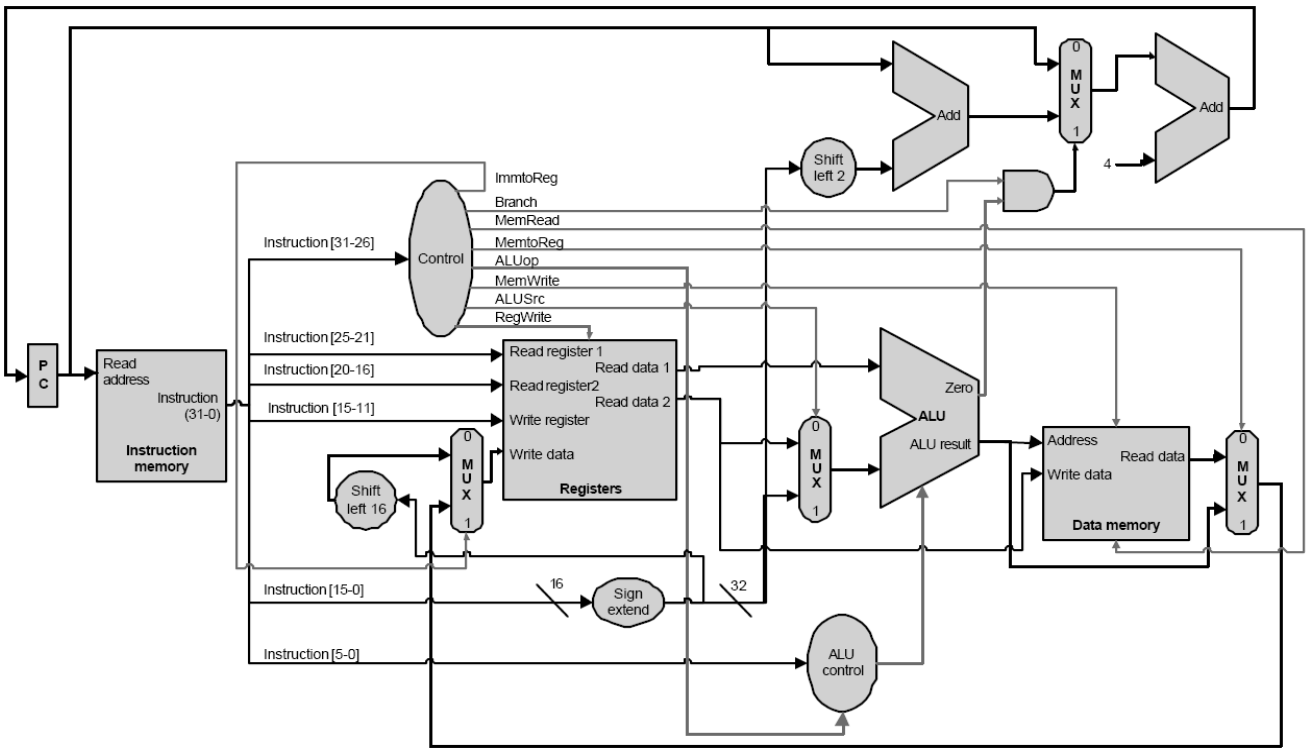
Func.Unit	Delay
Memory Read	5ns
Memory Write	2ns
ALU	3ns
Register Read	2ns
Register Write	4ns
3:1 Mux	2ns
2:1 Mux	1ns
Adder	3ns
sign extension	3ns



Problem states no delay for PC, in that case 15 is the correct answer.

Please note that PC is a register and time to write should be taken into account as shown above.

Problem 3. (15pts): Some of the following instructions can *not* be carried out in the provided datapath. Explain why in the space after each instruction that can't be implemented.



1) `add rd, rs, rt`

0	rs	rt	rd	0	0x20
6	5	5	5	5	6

Works fine

2) `lw rt, offset(rs)`

0x23	rs	rt	offset
6	5	5	16

Rt not available as an address to write to register file , missing mux

3) `j target`

2	target
6	26

Can't load PC with jump address

4) `lui rt, imm`

0xf	0	rt	imm
6	5	5	16

Rt not available as an address to write to register file , missing mux

5) `bne rs, rt, label`

5	rs	rt	offset
6	5	5	16

Only beq works

Problem 4: (15 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	RegWr	ExtOp	ALUSrc	ALUCtr	MemWr	MemToReg
addu	0	0	0	1	1	X	0	0	0
subu	0	0	1	1	X	0	0	0	0
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 will 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:

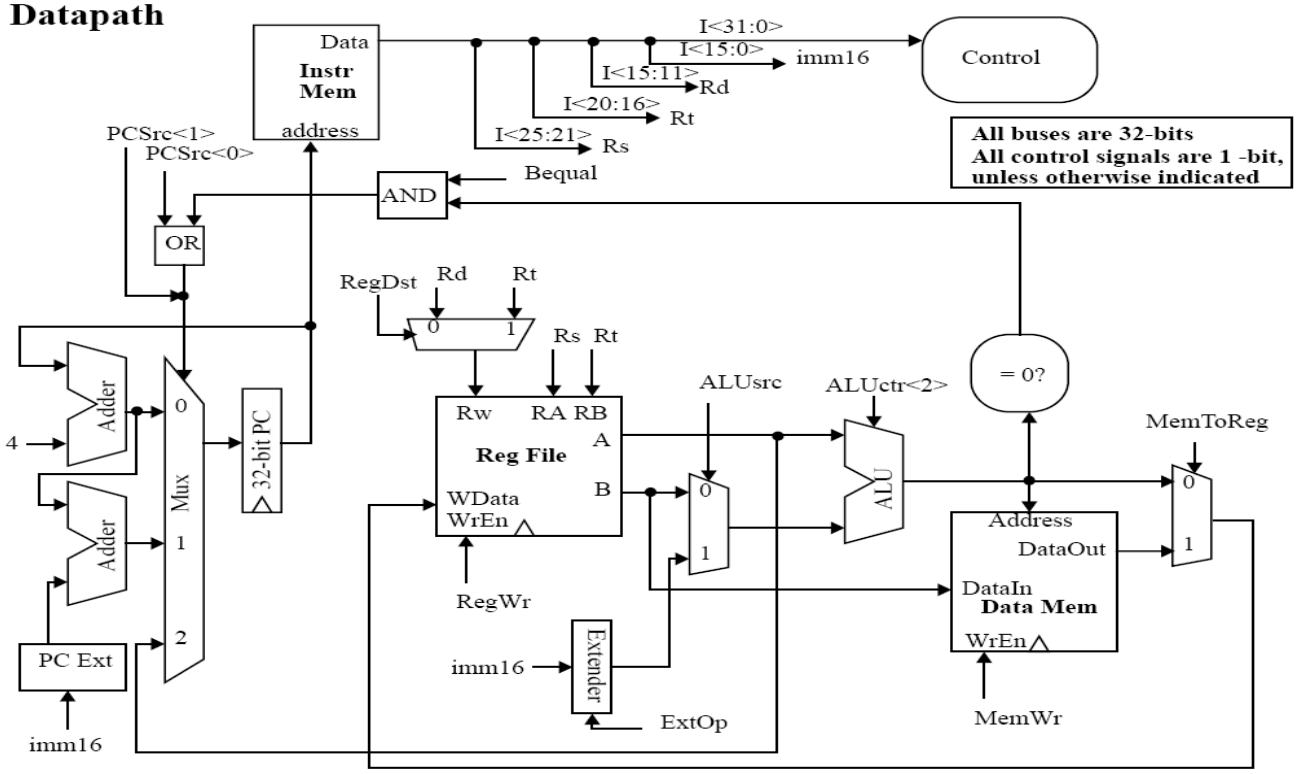
Control bits	Operation
0	add
1	sub
2	or
3	xor

For the following stream of instructions, what does your broken processor actually do? Here are some possible answers.

- **Functions correctly**
- **Incorrect functionality**: Indicate the control signals causing this error, and if it implements a different instruction then specify that instruction including the type of operation and registers used (operation, rs, rt, rd)
- **Functions correctly only under certain circumstance(s)**: Specify the circumstance

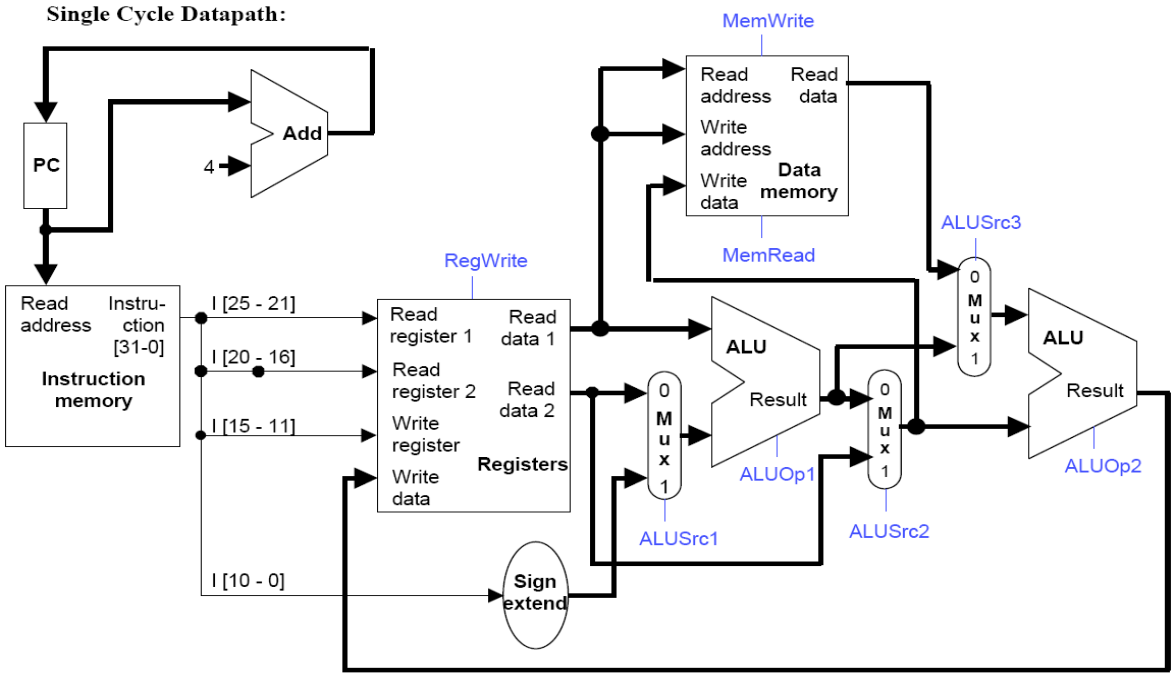
If there is more than one possibility, list all of them. For simplicity, we have used the actual register numbers rather than names.

Datapath



Instruction	Possible behavior(s) and Justification to get credit
subu $4, $5, $6	add $6, $5, $6 (incorrect behavior, both regDst and ALUctr are wrong)
addu $4, $5, $6	Alu src is wrong, functions correctly only when alusrc is 0
beq $11, $12, 24	correct behavior (even though AluCtr is xor, it still works since we are testing for equality)
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 nonexistent 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.)\

Problem 5 (15pts):



Field	op	rs	rt	rd	imm
Bits	31-26	25-21	20-16	15-11	10-0

Part (a) Assume that for the single cycle datapath shown above all instructions use the same format, but not all instructions use all of the fields. **Assume that each unused field is set to 0. ALU operation (ALUOp) can be one of the following operations: add, sub, mul, sll, and srl.** Specify how the control signals should be set for correct operation of the "mov" instruction. Use X for **don't care**. You are **not** allowed to modify the datapath.

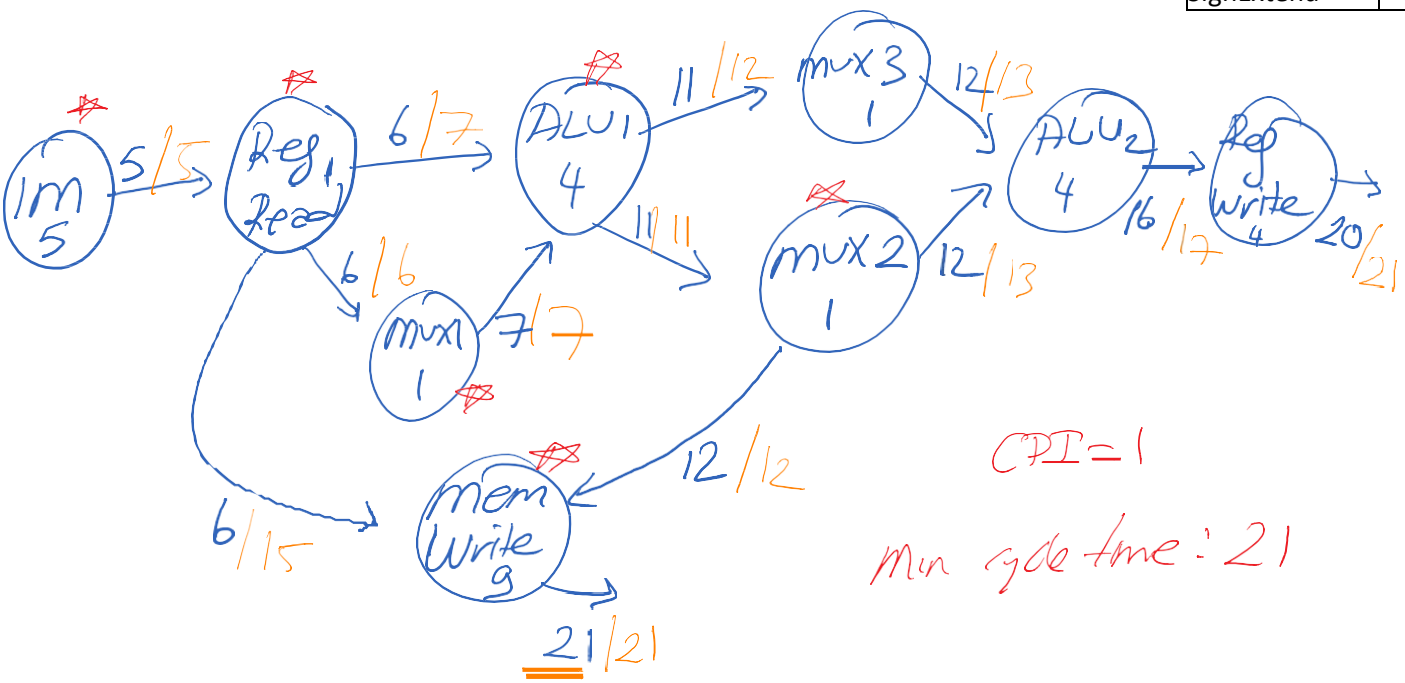
```
mov rs, rt, rd   # Memory[R[rs]] = R[rs] + R[rt]; R[rd] = 2*R[rs] + 2*R[rt]
```

Instr	RegWrite	ALUSrc1	ALUop1	ALUSrc2	ALUSrc3	ALUOp2	MemRead	MemWrite
mov	**1**	**0**	**add**	**0**	**1**	**add**	**0**	**1**

Part (b) Given the functional unit latencies, what is the **minimum CPI** and **minimum cycle time** based on the "mov" instruction? Assume PC has no delay. Show your critical path delay analysis using a graph where each node represents a function unit and each edge represents the data flow from one node to another. Show your work to get credit.

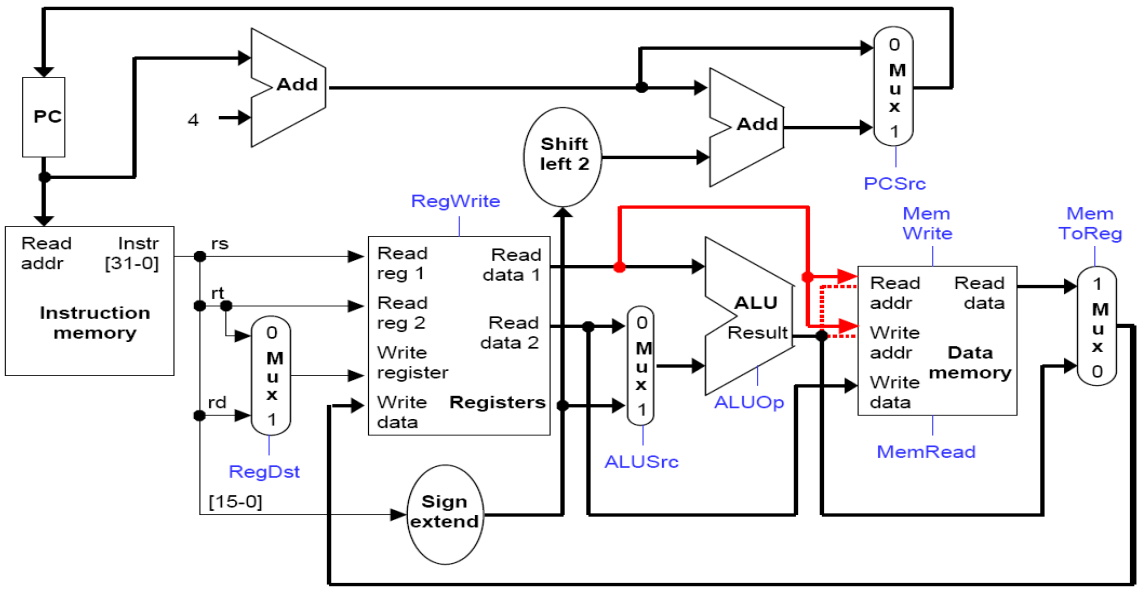
```
mov rs, rt, rd #Memory[R[rs]] = R[rs] + R[rt]; R[rd] = 2*R[rs] + 2*R[rt]
```

Func.Unit	Delay
Inst. Memory	5ns
Data Memory (Write)	9ns
Data Memory (Read)	6ns
ALU	4ns
Register (Read)	1ns
Register(Write)	4ns
Mux	1ns
Add	2ns
SignExtend	2ns



Path delay analysis requires slack calculations as labeled above.

Need to clearly indicate the time on each wire and label each component that is on the critical path.



Problem 6 (15pts):

Your teammate proposes to simplify the MIPS instruction set architecture for the competition phase of the project by removing the support for original lw and sw instructions (all memory accesses) and replacing them with ones that do not contain a constant offset. In the simplified instruction set , new loads and stores will have the following general forms:

lw rt, rs # GPR[rt] = Memory[GPR[rs]]

sw rt, rs # Memory[GPR[rs]] = GPR[rt]

Your teammate modifies the datapath to support the new lw and sw instructions as shown above. The dotted red lines show wires that are removed from the original design and solid red lines with arrow indicate the new wires. Assume that memory accesses (reads/writes) and ALU each take 2ns, register file and adders each take 1ns, and all other components have negligible delay. Considering all instruction types, your teammate claims that minimum cycle time will be shorter than the original datapath because lw/sw instructions no longer need the ALU for address calculation.

How much time is saved in ns compared to the minimum cycle time of the original datapath executing the standard R-type, I-type MIPS instructions? Your teammate claims that due to reduction in cycle time for this datapath, the clock rate will be faster, therefore revised datapath will execute programs faster compared to the original datapath. True/False. Justify your answer.

Original cycle time IM + RegRead + ALU + Data memory + RegWrite = 8ns

Bypassing ALU reduces path delay to 6ns.

This makes the datapath run faster by a factor of 8/6 but requires all offsets to be 0. That means compiler needs to invoke an additional instruction to calculate the exact address to prepare for lw/sw instructions. This will increase the number of instructions. For a code that is heavy on memory addressing, we will observe longer time to complete. If the code is compute intensive rather than memory access intensive, then the code will benefit from faster clock rate resulting with reduced execution time.

Therefore, not all programs will execute faster, it really depends on how much that program uses the data memory; the more times it has to access memory with this design, the worse it becomes since we need 2 cycles each 6ns (total of 12ns) vs 1 cycle of 8ns.