Single Cycle Datapath Questions

F20: New Single Cycle Datapath Points: ___/20

LC2K++ replaces the `noop` instruction in LC2K with the following new **I-type** instruction to help implement loops.

`binc regA regB offset`

`binc's` execution semantics is as follows:

```
if (regA == regB) {
        PC = PC + 1 + offset;
}
else {
        PC = PC + 1;
        regA++;
}
```

I. Write the equivalent LC2K assembly code for "`binc` 1 2 3". You may use `reg4` as a temporary register and you may add ".fill" into the data section. [2 pts]

Text Section

```
      beq 1 2 5
      lw  0 4 one
      add 1 4 1
```

Another correct answer:

```
      lw 0 4 one
      beq 1 2 4
      add 1 4 1
```

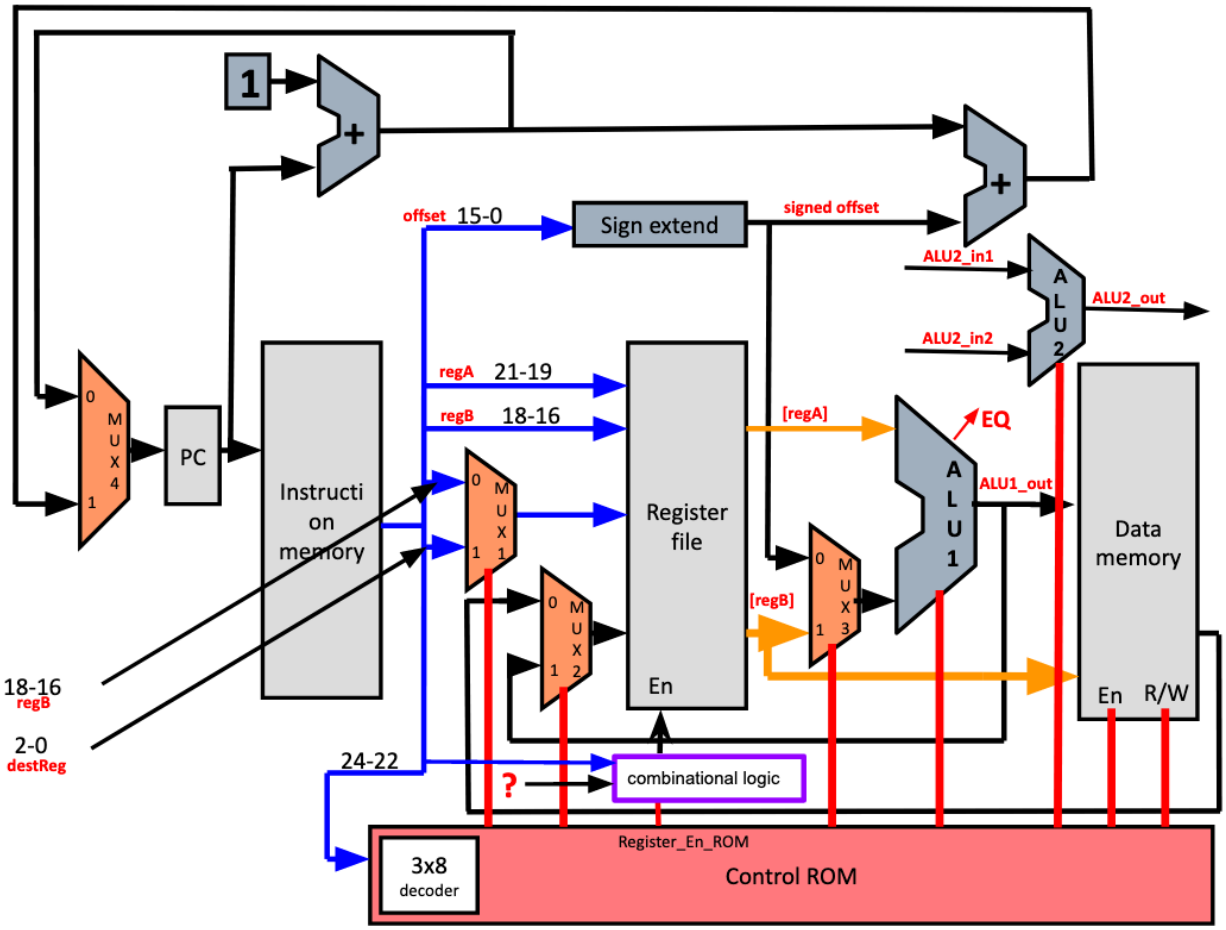
Another correct answer:

```
      beq 1 2 eq
      lw 0 4 one
      add 1 4 1
      beq 0 0 1
eq    beq 0 0 3
```

Data section

```
one   .fill      1
```

II. Modify the datapath to support `binc` and all the original LC2K instructions, except `noop`. You may use an **additional ALU2** and you can extend **two of the MUXes** with an **additional input**.



If part a uses solution 1([regA], 1) or incorrect, part b and c will be graded based on the solution 1. If part a uses solution 2([regA], [regB]), part b and c will be graded based on the solution 2.

a. Specify the input wires to ALU2 in terms labels shown in the picture or a constant: [3 pts]

_____[regA], constant 1_____ _____[regA], [regB]_____

b. Specify the MUXes that are extended with additional inputs (MUX1, MUX2, MUX3, and/or MUX4), and the new input connected to them. Use the labels in the picture.[4 pts]

Extended MUX#	New Input Wire Connected	Extended MUX#	New Input Wire Connected
MUX1	regA	MUX1	regA

MUX2	ALU2_out	MUX3	Constant 1

c. Determine the control signals for **`binc`** instruction that is stored in ROM.
Assume the **select signal** for any new MUX input is **'10'**.
Register_En_ROM is given.
Control for ALU: `add: 0; nor: 1` [4 pts]

PC_en	MUX1	MUX2	MUX3	Register_ En_ROM	ALU1	ALU2	Data Mem_en	Data Mem_R/W
1	10	10	1	**0**	X	0	0	X
1	10	1	10	**0**	0	X	0	X

d. Describe the boolean equation for determining the following control signals for `binc`.
Register_En_ROM is a control signal stored in ROM. [4 pts]

MUX4 select = { (opcode == beq) && EQ)} OR { (opcode == binc) && EQ }

Register_En = Register_En_ROM OR { (opcode == binc) && !EQ }
(This is the logic for combination logic box shown in picture)

e. Assume the following latencies. What is the minimum delay for `binc` instruction when regA is not equal to regB? [3 pts]

Instruction Memory	10ns	15ns	20ns
Register File Read	5ns	10ns	10ns
ALU1	15ns	15ns	15ns
ALU2	15ns	15ns	15ns
Data Memory	10ns	20ns	15ns
Register File Write	15ns	10ns	10ns
	Version 1	Version 2	Version 3

For all solutions, ALU1 & ALU2 can compute in parallel

Version 1:

Instruction Mem + Register File Read + ALU1 + Register File Write = 10 + 5 + 15 + 15 = 45ns

Version 2:

Instruction Mem + Register File Read + ALU1 + Register File Write = 15 + 10 + 15 + 10 = 50ns

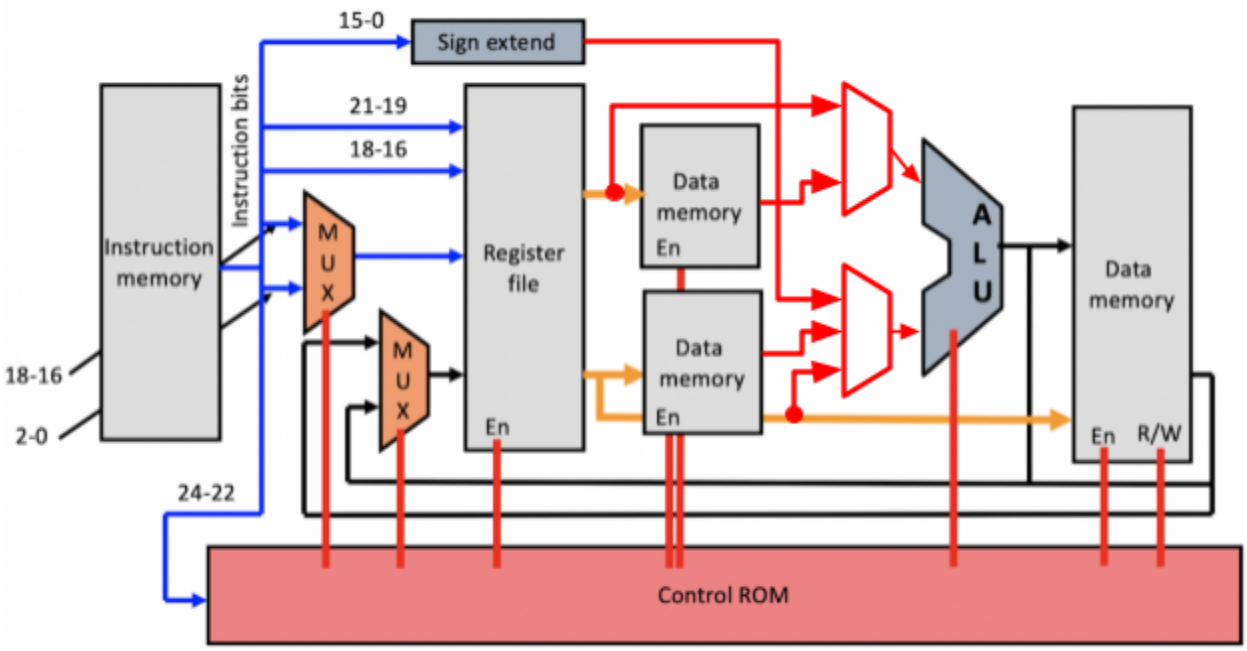
Version 3: Instruction Mem + Register File Read + ALU1 + Register File Write + Instruction Mem + Register File Read + ALU1 + Register File Write = 20 + 10 + 15 + 10 = 55ns

F19: The Mad Ladd [17 points]

The mad genius Finbarr Calamitous, tired of writing multiple instructions to sequentially load and add, has designed a new instruction to replace noop:

ladd = reg[destReg] = mem[regA] + mem[regB]

Unfortunately, Calamitous has not completed his modifications to the datapath to accommodate his plans. To help him implement this new instruction, you will need to add exactly **one 2:1** mux (mux A) and **one 3:1** mux (mux B) to Calamitous' schematics. Draw legibly. Note that the datapath section for branching is not shown to make the diagram easier to read. You may assume that part is unchanged. The add, lw, sw, and nor instructions must still be possible to implement.



a. Add the two MUXes to the diagram. Draw neatly and carefully. **[6]**

b. Using the original LC2K assembly language, write as short a code segment as possible that does the same thing as the instruction "`ladd 1 2 3`". You can use reg4 as a temp register **[3]**

```
        lw 1 4 0
        lw 2 5 0
        add 4 5 3
Or avoiding temps:
        lw 1 1 0
        lw 2 2 0
        add 1 2 3
```

<Problem continued on next page>

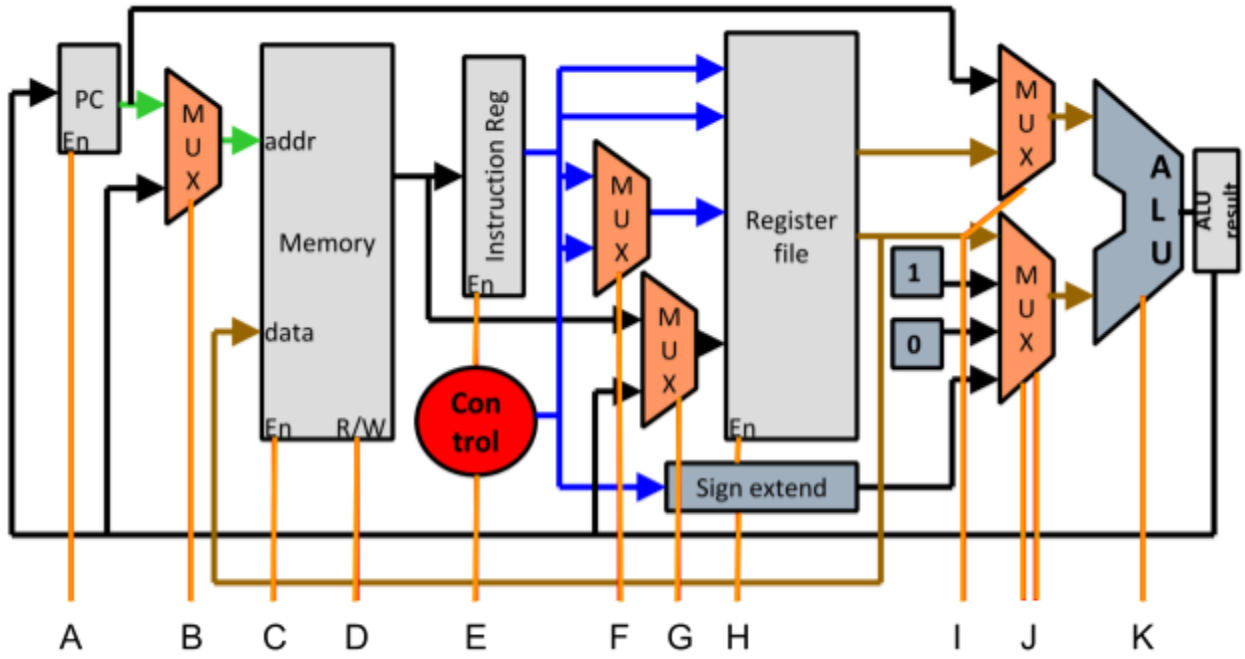
c. Say that the data path changes caused the clock period to change from 8ns (original LC2K) to 9ns (above data path). Say Finbarr Calamitous has a program where 10% of the instructions are ladd instructions. That program takes exactly 1 second to run on the new data path. How long would it take to run on the original data path if you replaced the ladd instructions with original LC2K instructions as you did in part b? Clearly show your work. **[8]**

(0.9 + 0.1 x 3) 8/9 = 1.07 seconds

Multi-Cycle Datapath Questions

W18: Multi-cycle datapath

Points: ___/ 20



Suppose we want to add support for pre- and post-incrementing load and store instructions to LC2K using the following format:

```
lw      ++1     2       100     // pre-increment: R1 += 1 ; R2 = M[R1 + 100]

sw      1++     2       100     // post-increment: M[R1+ 100] = R2; R1 += 1
```

a. Adding these instructions requires additional hardware. Add up to 2 additional wires on the above diagram to accomplish these instructions with the minimum number of additional cycles. (Do not modify the ALU, Memory, PC, ALU result, or Register File. Do not add any additional MUXes.) Circle the letter of any muxes you modify and describe what you are changing:

Changed mux(es): B **F** G **I** J

Extend mux F to pass rA as the destination register.

Extend mux I to pass ALUresult back into it.

Partial credit: Extend mux F only and add an extra cycle

b. What operations occur in each cycle for a **pre-increment load word**? (Use the minimum number of cycles.)

Cycle 1	Fetch
Cycle 2	Decode
Cycle 3	**Compute rA + 1 and store in ALUresult**
Cycle 4	**Save ALUresult to rA ; Compute ALUresult + sign-extended offset**
Cycle 5	**Start memory load using the address from ALUresult**
Cycle 6	**Write data from memory to rB**
Cycle 7	
Cycle 8	

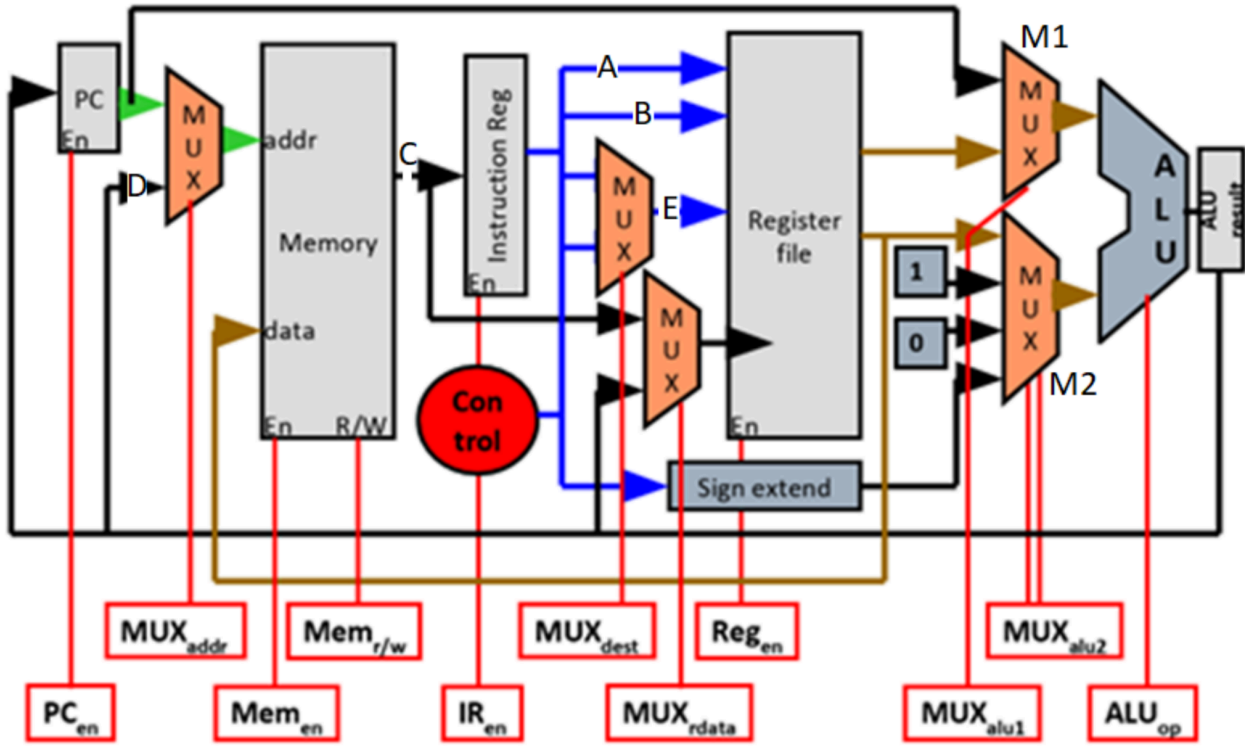
Partial credit: rA writeback and ALUresult + offset in separate cycles

c. What operations occur in each cycle for a **post-increment store word**? (Use the minimum number of cycles.)

Cycle 1	Fetch
Cycle 2	Decode
Cycle 3	**Add rA + sign-extended offset and store in ALUresult**
Cycle 4	**Write rB to memory at the address from ALUresult ; Compute rA + 1**
Cycle 5	**Save ALUresult to rA**
Cycle 6	
Cycle 7	
Cycle 8	

Partial credit: Memory write and and rA + 1 in separate cycles

F18: Improving the Multicycle Datapath Points: ___/20

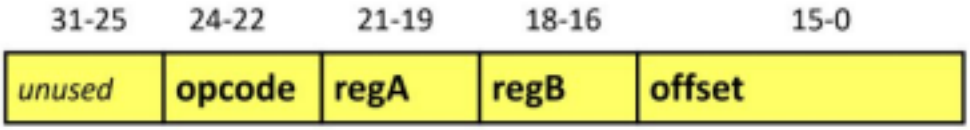


Imagine that we add a new LC2K instruction "madd", with opcode value of 0b111 (replacing noop) with following functionality:

Read memory at address [regA + offset] and increment regB by the value from memory.
`madd regA regB offset //regB = regB + Mem[regA + offset]`

Example: `madd 1 2 10 //r2 = r2 + Mem[r1 + 10]`

The "madd" instruction is an I-type instruction and will have the following machine code format:



a) Using the standard multi-cycle LC2K datapath shown above, make at most 2 additional connections between the **labeled wires (labeled A–E)** and the inputs of the **labeled muxes (labeled M1,M2)** to implement the madd instruction.

Fill in the blanks. If less than 2 connections are needed, then leave lines blank.

Connect wire __C__ with input of mux __M1__

Connect wire _____ with input of mux _____

b) Write the control signals for all the cycles needed to perform a “madd” operation on the modified multi-cycle datapath after your connections from part (a) have been added to the hardware.

Assumptions

- The top mux input is selected when all mux control bits are 0.
- All connections added to muxes have been added at the bottom of the mux. ● If a mux has multiple connections added to it from part (a), then the later alphabet wire will be at the bottom.
- The number of select bits for extended muxes might need to be increased ● MUX (dest): 0 selects regB and 1 selects destReg
- Mem (r/w): 0 for read and 1 for write.
- ALU (op): 0 for add and 1 for nor

Hint: If you’ve added more inputs to a mux, make sure the number of control bits for the modified mux is correct.

PC (en)	**MUX (addr)**	**MEM (en)**	**MEM (r/w)**	**IR (en)**	**MUX (dest)**	**MUX (rdata)**	**Reg (write en)**	**MUX (alu1)**	**MUX (alu2)**	**ALU (op)**
0	0	1	0	1	X	X	0	00	01	0
1	X	0	X	0	X	X	0	XX	XX	X
0	X	0	X	0	X	X	0	01	11	0
0	1	1	0	0	X	X	0	XX	XX	X
0	X	0	X	0	X	X	0	10	00	0
0	X	0	X	0	0	1	1	XX	XX	X

//cycle 3: [ALU_Result] = [regA] + offset
//cycle 4: [DATA_REG] = MEM[ALU_Result]
//cycle 5: [ALU_Result] = [DATA_REG] + [regB]
//cycle 6: [regB] = [ALU_Result]

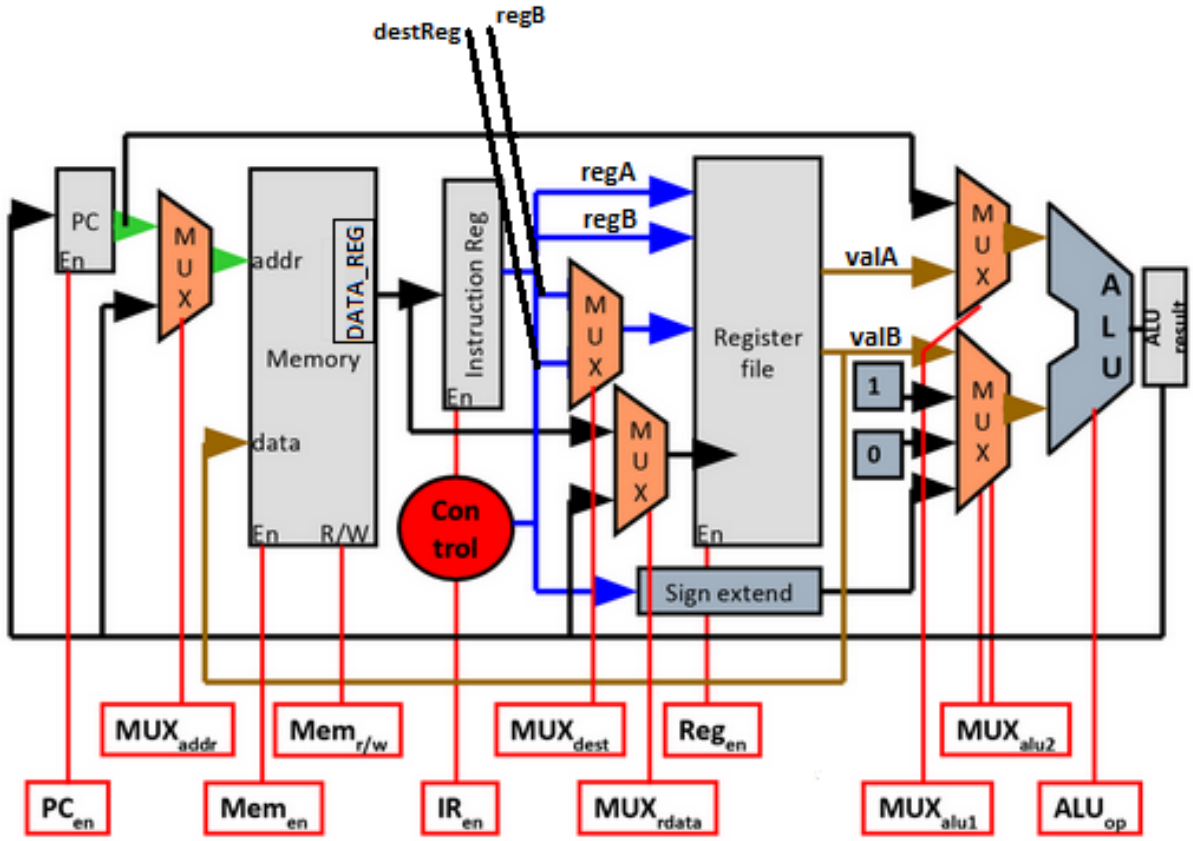
W20: The Negator Points: ___/25

A mysterious villain has appeared--going only by the name “The Negator”. The goal of this mystery figure is to flip the sign of all the values currently in memory. His first step will be to load the value he wants to change into a register. However, to finish the task, he will need a new instruction. The new instruction, **neg,** is defined as follows:

```
neg regB offset // mem[offset] = -regB
```

Read the value of regB and store **negative** regB into the memory address offset.

Example: `neg 2 4 // mem[4] = -r2`



To support this new instruction, The Negator has to modify the above multicycle datapath
// cycle 3: calculate regB nor regB, store into ALUresult
// cycle 4: calculate ALUresult + 1, store back into ALUresult
// cycle 5: store ALUresult into Memory[offset]
The below solution is derived by attaching a wire from regB to Mux (alu1)
Another solution would be to have MUX (alu2) be 10 in cycle 3 (nor with 0)
Alternate solutions include attaching a wire from a constant value of 0 to Mux (alu1) with the same control signals currently
**Two solutions possible: Routing “0” or “RegB” to MUX_ALU1, please give both solutions full points.
**Watch out for MUX_ALU1 select bits in part (b) table - because any order of ALU_result and RegB is correct (select bits will be different) and should get full points.

(a) To support **neg** instruction, you can **only modify the MUXes** (and add any necessary connections to them) **in the table below** and **add only *one new MUX***. Note that none of the MUXes should have more than four inputs to them. Describe your changes to datapath below. Note neg takes 5 cycles to complete in the new datapath.

MUX name	**Modified (Yes/ No)**	**Connections added/removed**
MUX_addr	Yes	Add input from "sign extend" wire, which represents offset
MUX_dest	No	N/A
MUX_rdata	No	N/A
MUX_alu1	Yes	Add inputs from regB/0 (ctrl 10) and AluResult (11)
NEW_MUX	Added New	Add input from regB (0) and AluResult (1) and feed output in to the data field in mem

(b) Write the control signals for all the cycles needed for the "neg" instruction on the modified datapath after your connections have been added.

You can assume the following things:

- The top mux input is selected when all mux control bits are 0
- All connections added to any mux are added at the bottom of the mux
- The select bits for extended muxes may need to be increased from the original mux
- Mem (r/w): 0 for read, 1 for write
- ALU (op): 0 for add and 1 for nor

Hint. If you've added more inputs to the MUX, make sure the number of control bits for the modified mux is correct

PC (en)	**MUX (addr)**	**MEM (en)**	**MEM (r/w)**	**IR (en)**	**MUX (dest)**	**MUX (rdata)**	**Reg (write en)**	**MUX (alu1)**	**MUX (alu2)**	**ALU (op)**	**NEW_ MUX**
0	0	1	0	1	X	X	0	0	01	0	X
1	X	0	X	0	X	X	0	X	XX	X	X
0	XX	0	X	0	X	X	0	10/11	00/10	1	X
0	XX	0	X	0	X	X	0	11/10	01	0	X
0	10	1	1	0	X	X	0	XX	XX	X	1

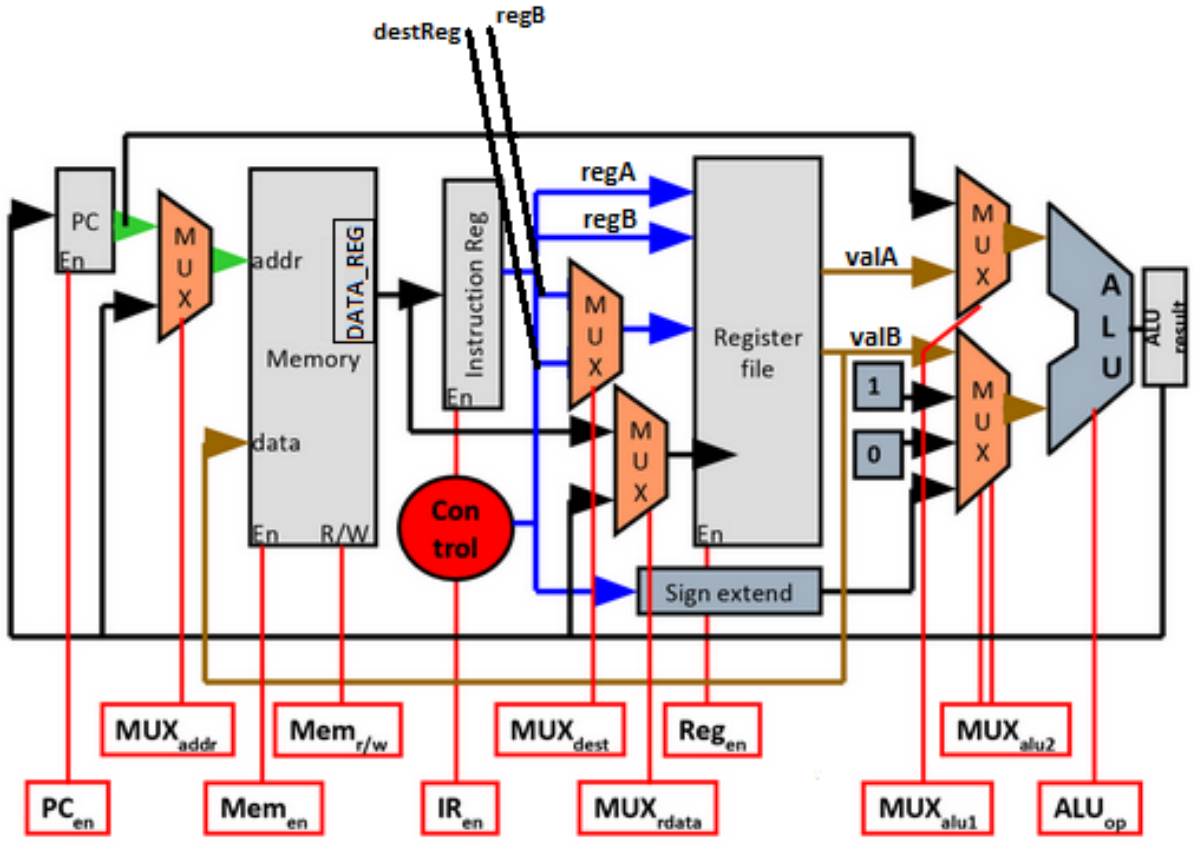
F20: Multicycle Datapath Design Points: ___/20

Consider a new LC2K I-type swap instruction:

```
swp   regA  regB offset
```

Execution semantics of this instruction is as follows:

tmp	= Mem[regA+ offset]
Mem[regA + offset]	= regB
regB	= tmp



a. Implement **`swp`** in the multi-cycle data-path in five or fewer cycles. Specify the operations for **`swp`** in each cycle. Provide an informal description (few words) followed by exact changes to the multicycle state. Assume [data_reg] stores the value read from memory. [10 pts]

You may not need to fill all the blanks.

Cycle 1: Fetch
[Instruction_Reg] = Mem[PC]
[ALU_Result] = PC + 1

Cycle 2: Decode
[PC] = [ALU_Result]
Read register values

Cycle 3:
[ALU_Result] = [regA] + offset

_____ = _____

Cycle 4:
[DATA_REG] = Memory[ALU_Result]

[ALU_Result] = [regA] + offset

Cycle 5:
Memory[ALU_Result] = [regB]

[regB] = [DATA_REG]

b. Specify the control signals for each cycle for **`swp`**: [10 pts]

Cycle	PC (en)	MUX (addr)	MEM (en)	MEM (r/w)	IR (en)	MUX (dest)	MUX (rdata)	Reg (en)	MUX (alu1)	MUX (alu2)	ALU (op) 0: add 1: nor
1	0	0	1	0	1	X	X	0	0	01	0
2	1	X	0	X	0	X	X	0	X	XX	X
3	0	X	0	X	0	X	X	0	1	11	0
4	0	1	1	0	0	X	X	0	1	11	0
5	0	1	1	1	0	0	0	1	X	XX	X

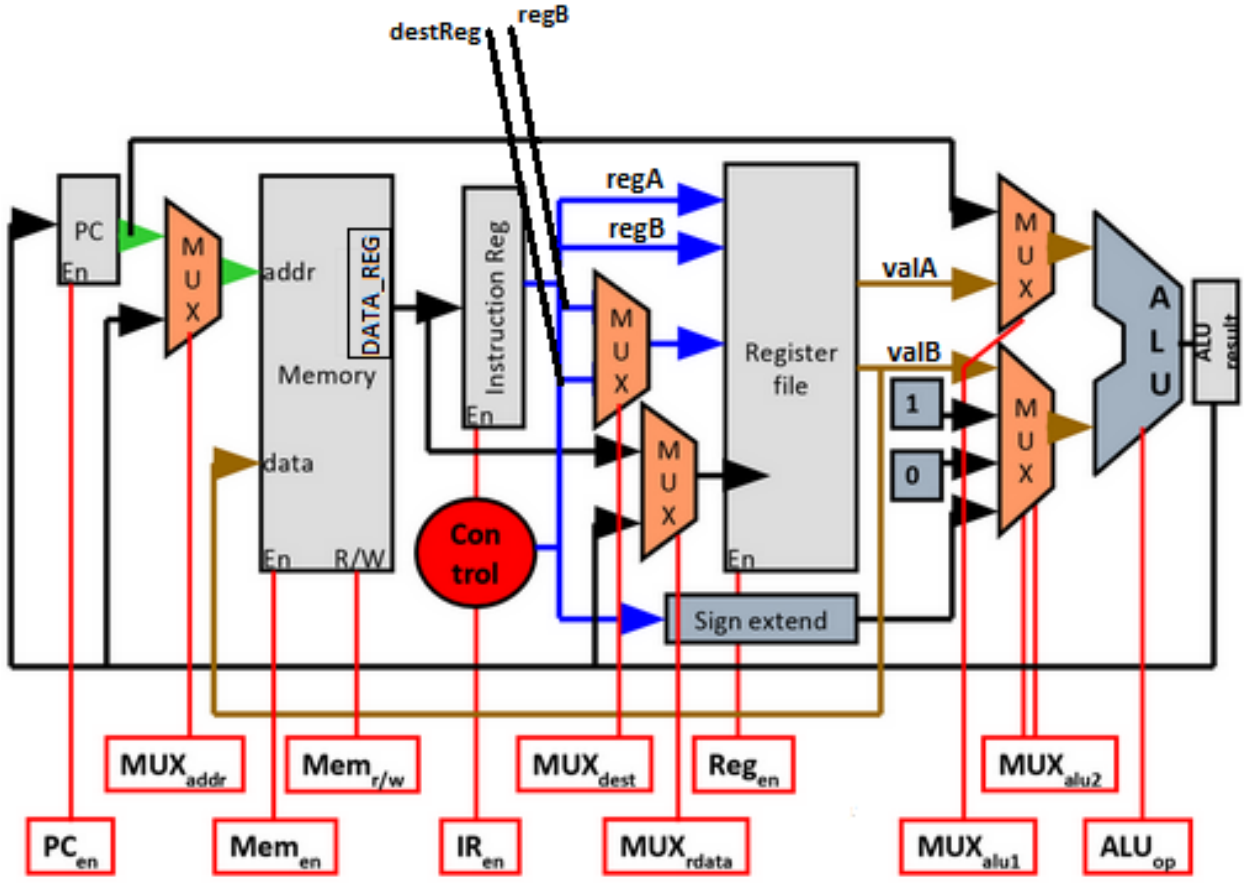
W21 Multi-cycle Datapath Design Points: ___/20

Consider a new LC2K I-type instruction:

```
accmems    regA  regB stride
```

The goal of this instruction is to facilitate strided accumulation of the data in the memory. Execution semantics of this instruction is as follows:

```
regB = regB + Mem[regA];
regA = regA + stride;
```



a) You have been asked to change the LC2K multi-cycle datapath to support the new instruction. We can **only modify the MUXes** (and add any necessary connections to them). Note that none of the MUXes should have more than four inputs to them. Describe your changes to datapath below. (`accmems` **should take 5 cycles or less** to complete) (5pts)

Example:

MUX_example1:	MUX_example2:
DATA_REG added to input 10.	NA // Leave blank or type NA if unchanged.

- MUX_addr:

  ___Add valA to input 10__

  MUX_dest:

  __Add regA (bits 19-21) to input 10__

- MUX_rdata:

  _____N/A_____

  MUX_alu1:

  ___Add DATA_REG to input 10___

- MUX_alu2

  _____N/A_____

b) Implement `accmems` in the multi-cycle data-path in five or fewer cycles. Specify the operations for each cycle. Provide an informal description (few words) followed by exact changes to the multicycle state. Assume [data_reg] stores the value read from memory. (10 pts) *You may not need to fill all the blanks.*

Cycle 1: Fetch

[Instruction_Reg] = Mem[PC]

[ALU_Result] = PC + 1

Cycle 2: Decode

[PC] = [ALU_Result]

Read register values

Cycle 3:

___[DATA_REG]___ = _____MEM[valA]_____

___[ALU_Result]___ = ___valA + offset/stride__

Cycle 4:

_____[regA]_____ = _____[ALU_Result]____

___[ALU_Result]___ = __[DATA_REG] + valB__

Cycle5:

_____[regB]_____ = _____[ALU_Result]____

_____ = _____

c) Write the control signals for all the cycles needed for the "`accmems`" instruction on the modified datapath after your connections have been added. (5 pts)

You can assume the following things:

- All connections added to any mux are added at the bottom of the mux
- The select bits for extended muxes may need to be increased from the original mux

PC (en)	MUX (addr)	MEM (en)	MEM (r/w)	IR (en)	MUX (dest)	MUX (rdata)	Reg (write en)	MUX (alu1)	MUX (alu2)	ALU (op)
0	0	1	0	1	X	X	0	0	01	0
1	X	0	X	0	X	X	0	X	XX	X
0	10	1	0	0	X	X	0	01	11	0
0	XX	0	X	0	10	1	1	10	00	0
0	XX	0	X	0	00	1	1	XX	XX	X