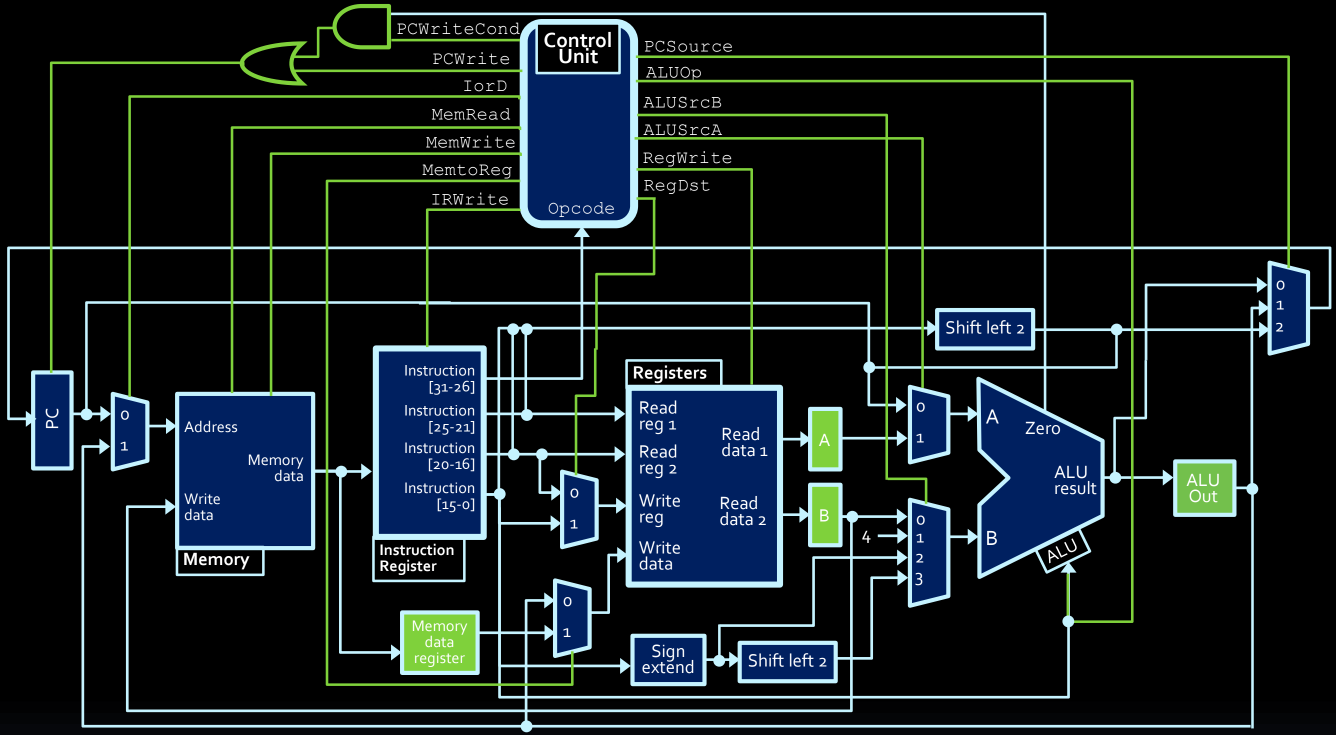
Week 9 Review

Question #1

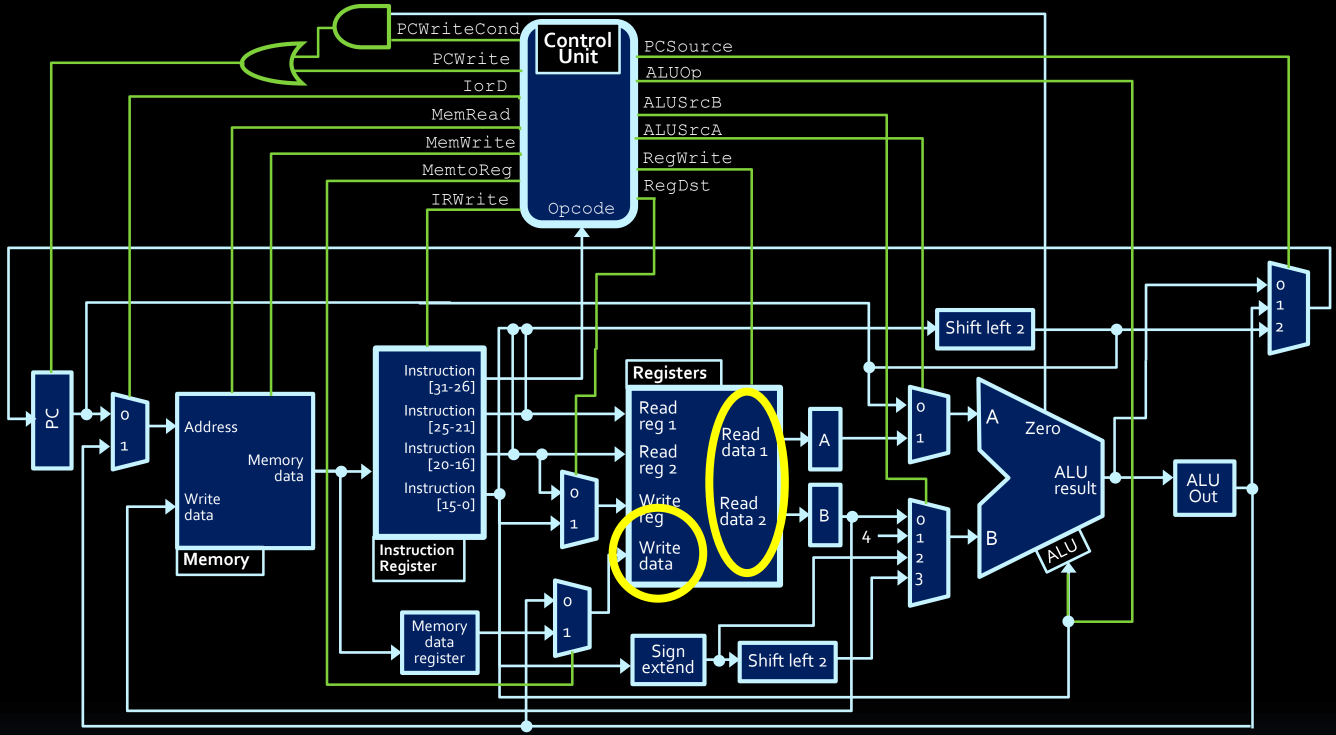


- Given the datapath above, what signals would the control unit turn on and off in order to add `$t1` to `$t2` and store the result in `$t7`?

Basic approach

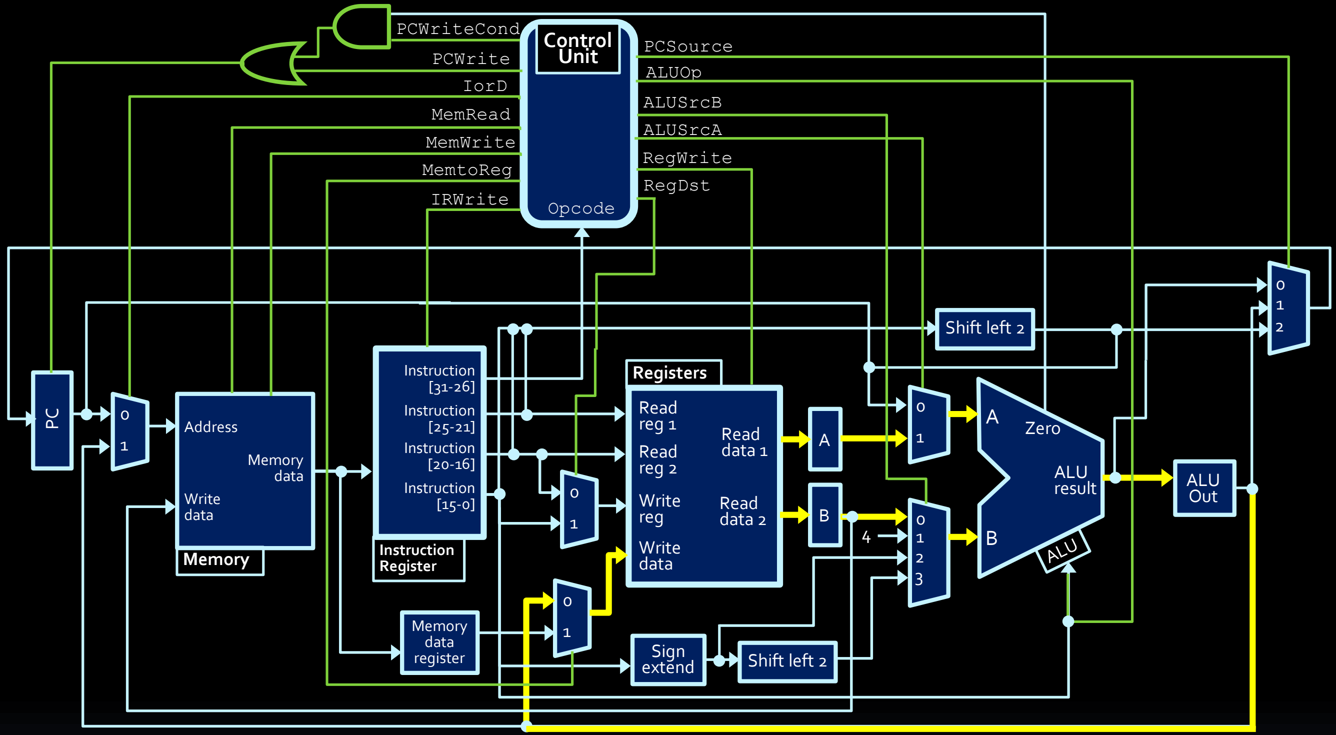
1. Figure out the data source(s) and destination.
2. Determine the path of the data.
3. Deduce the signal values that cause this path:
   a) Start with `Read` & `Write` signals (at most one can be high at a time).
   b) Then, mux signals along the data path.
   c) Non-essential signals get an `X` value.

Question #1 (cont'd)



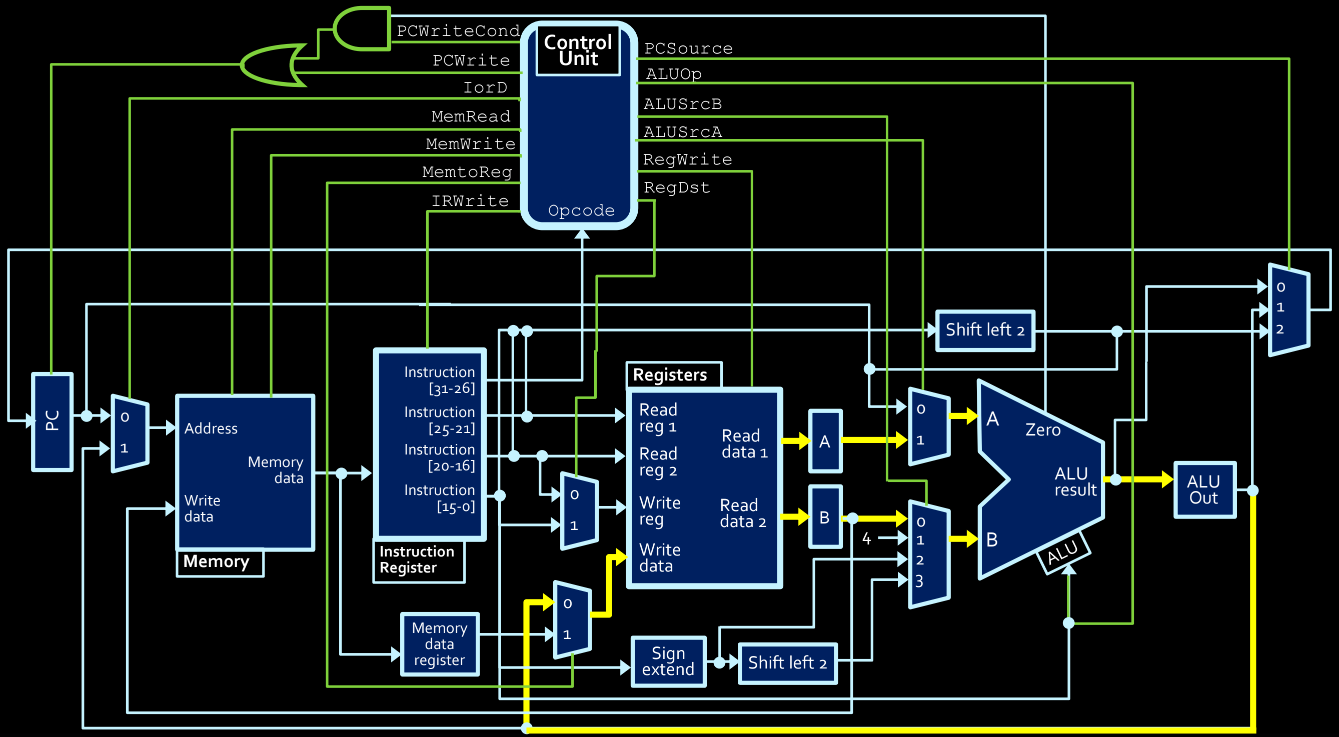
- Step #1: Data source and destination
  - Data starts in register block.
  - Data goes to register block.

Question #1 (cont’d)



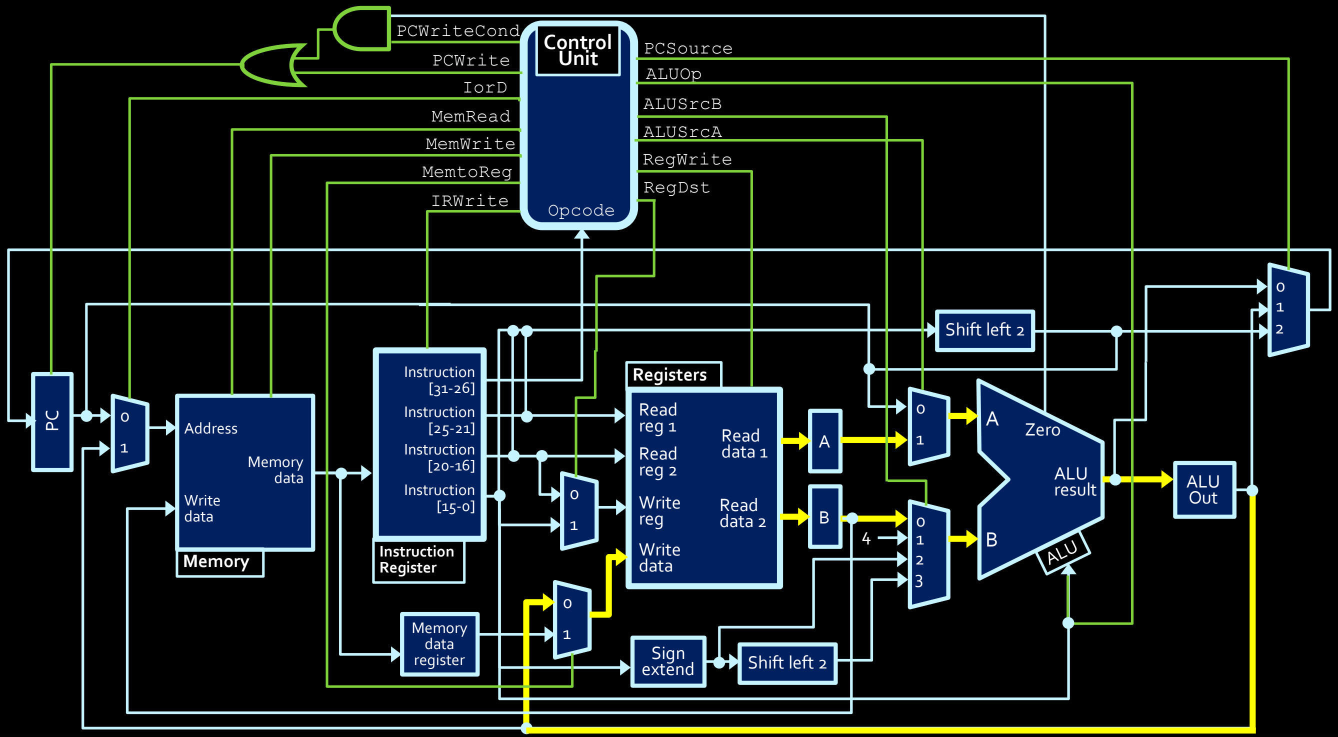
- Step #2: Determine the path of the data
  - Data needs to go through the ALU before heading back into the register file.

Question #1 (cont'd)



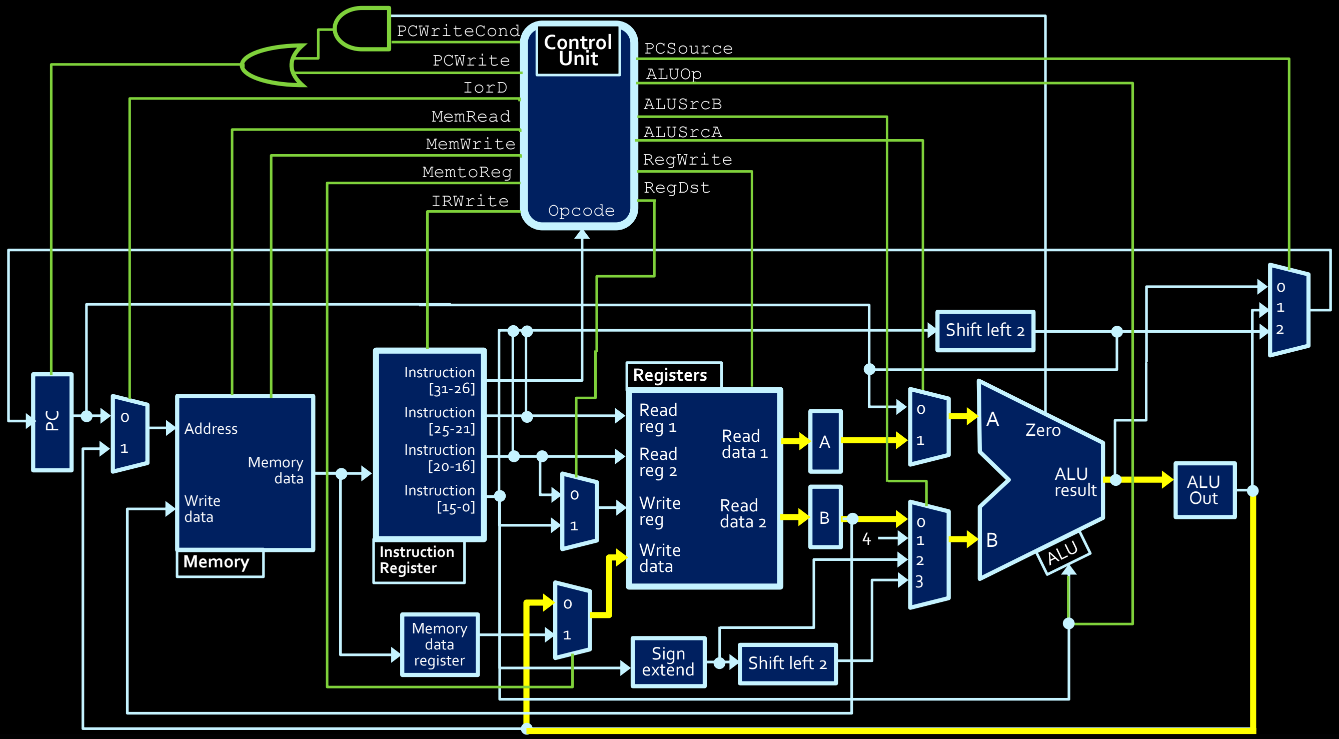
- Step #3a: Read & Write signals
  - Only RegWrite needs to be high.
  - PCWrite, PCWriteCond, MemRead, MemWrite, IRWrite would be low.

Question #1 (cont’d)



- Step #3b: Data path signals
  - Muxes before ALU: ALUSrcA → 1, ALUSrcB → 00.
  - ALUOp → 001 (Add)
  - Mux before registers: MemToReg → 0

Question #1 (cont’d)



- Step #3c: Non-essential signals
  - No writing to PC: `PCSource` → `X`.
  - No reading from memory: `IorD` → `X`.

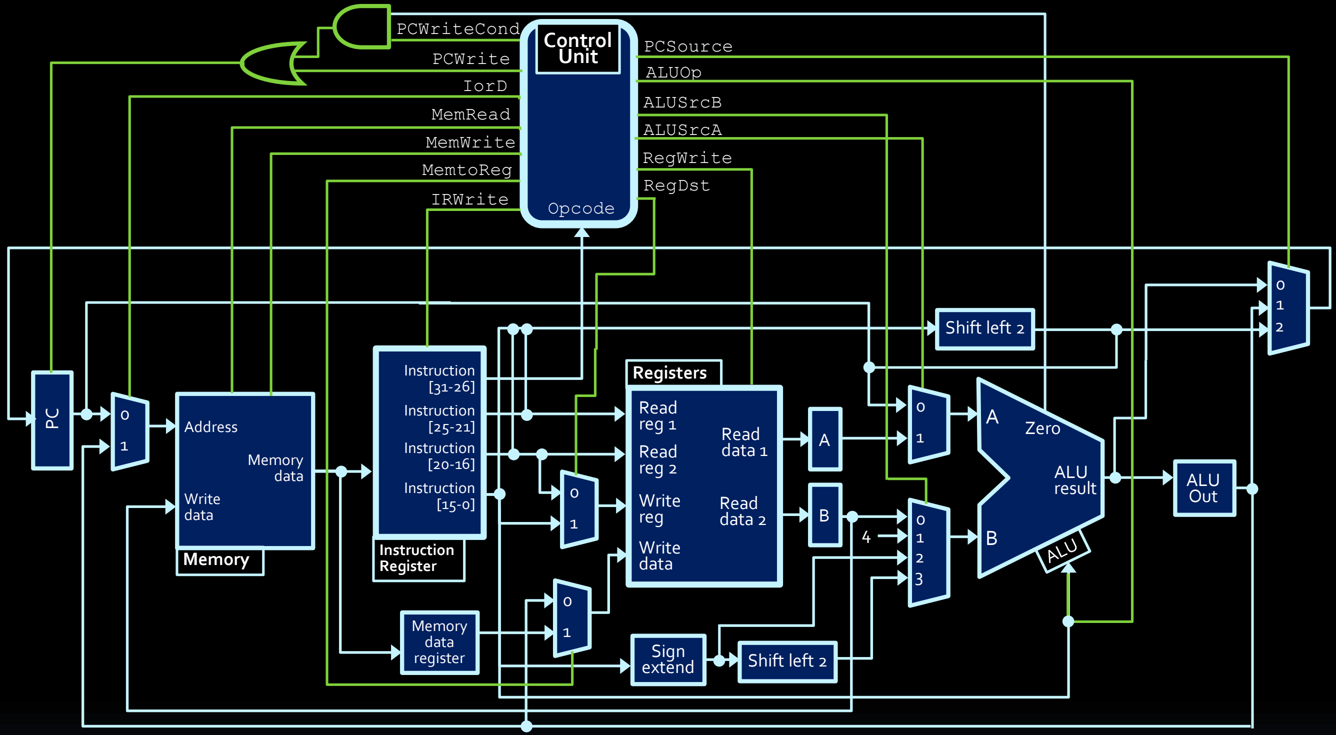
Question #1 (cont'd)

- PCWrite = 0
- PCWriteCond = 0
- IorD = X
- MemRead = 0
- MemWrite = 0
- MemToReg = 0
- IRWrite = 0
- PCSource = X
- ALUOp = 001

- ALUSrcA = 1
- ALUSrcB = 00
- RegWrite = 1
- RegDst = 1

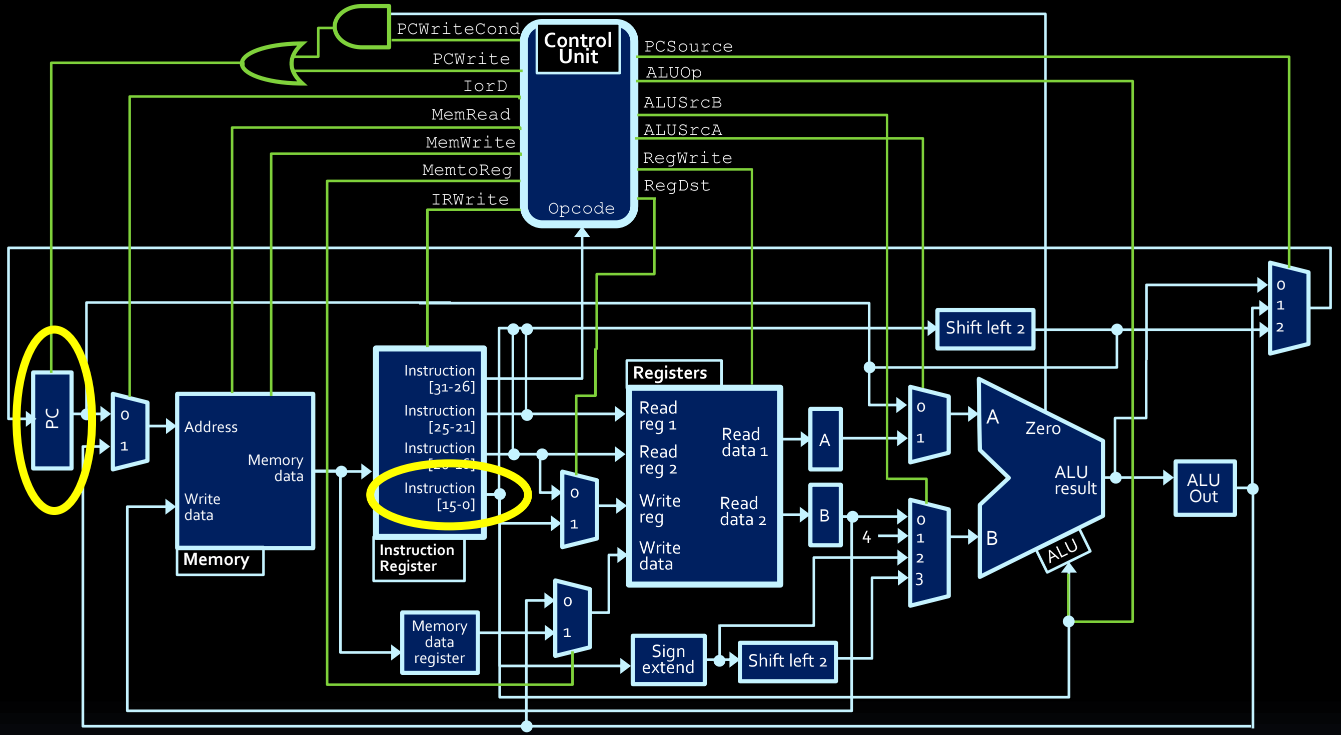
- Note: RegDst rule
  - high for 3-register operations
  - low for 2-register operations

Question #2



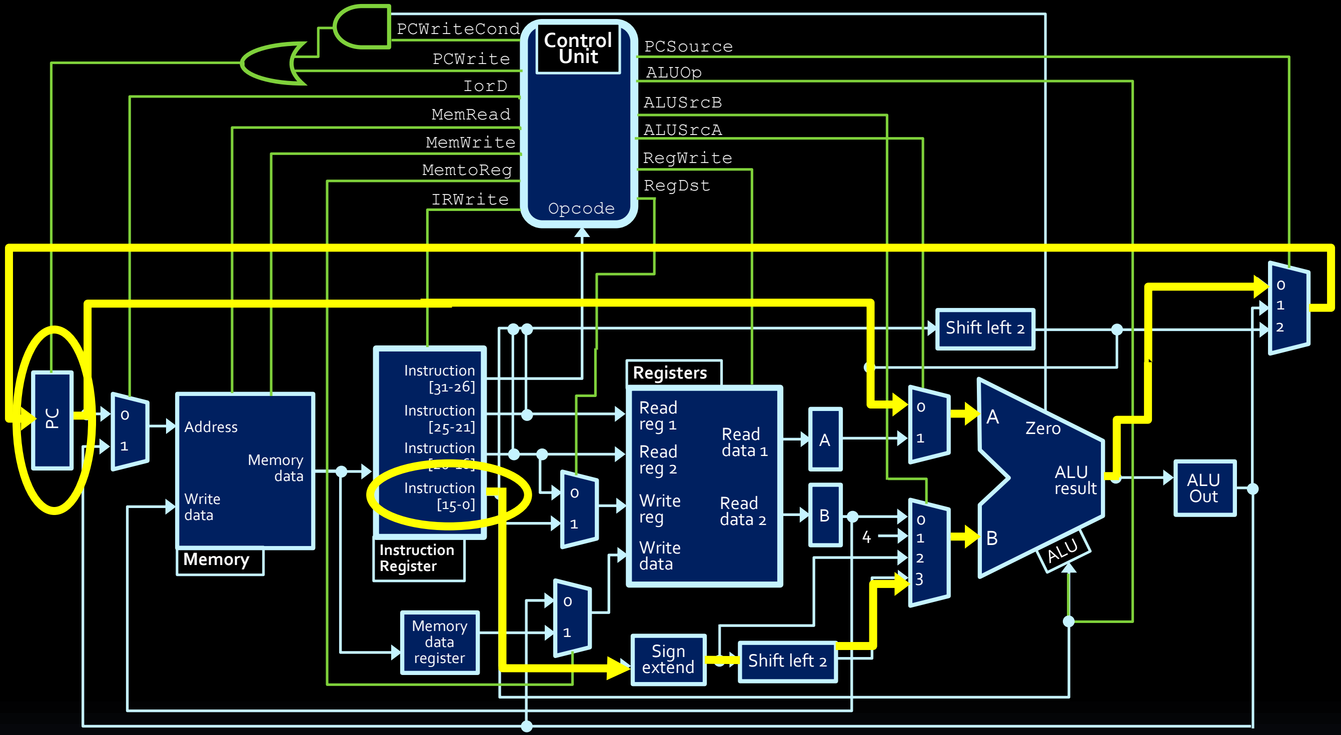
- Given the datapath above, what signals would the control unit turn on and off in order to branch to 100 instructions forward (add 100*4 to the program counter)?

Question #2



- <u>Step #1:</u> Source and destination.
  - Program counter is both source and destination.
  - Immediate value from instruction is other source.

Question #2



- Step #2: Path between source and destination.

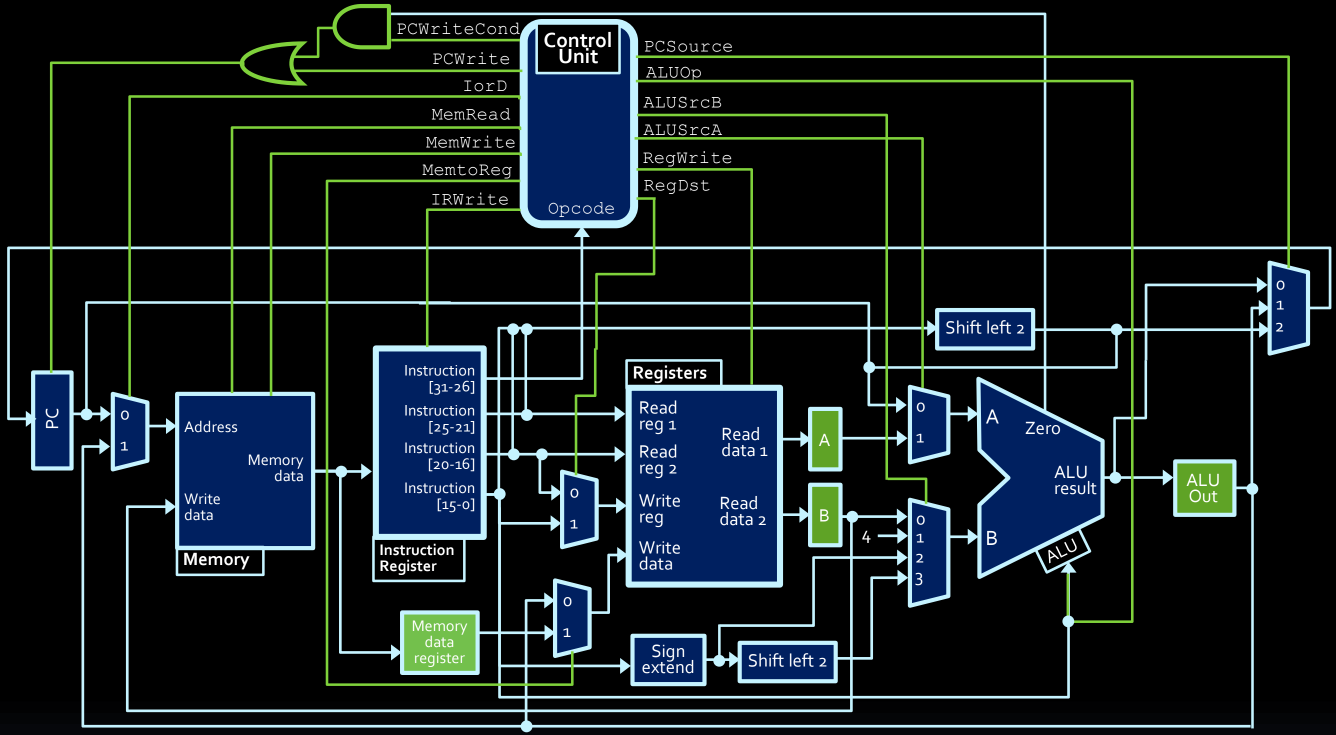
Question #2 (cont'd)

- Read / Write signals:
  - `PCWrite` high, all others low.
    - `PCWriteCond` is `X`, when `PCWrite` is high.
- Datapath signals:
  - `ALUSrcA` → `0`
  - `ALUSrcB` → `3` (`100` is an immediate value; needs to come from the instruction, and needs to be word-aligned because it concerns 4-byte instructions.)
  - `PCSource` → `0`
- Non-essential signals:
  - `IorD`, `MemToReg`, `RegDst`

Question #2 (cont’d)

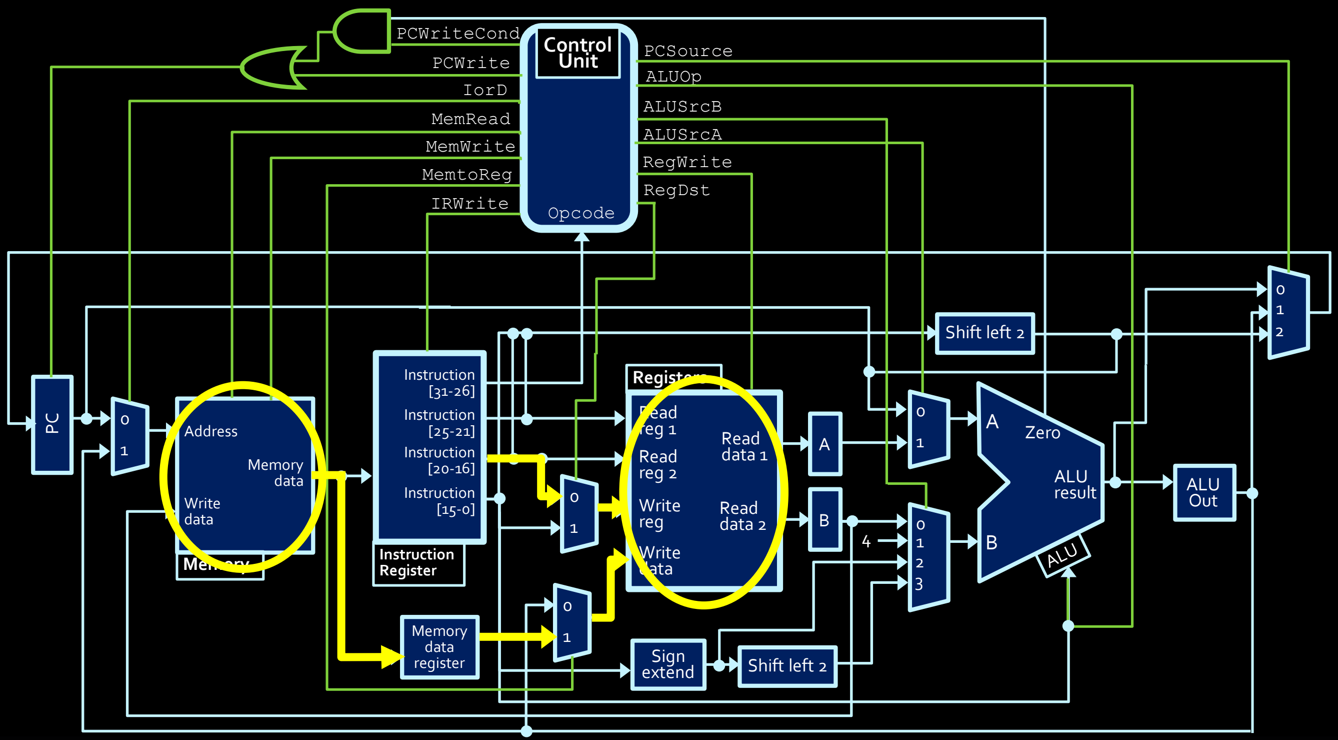
- PCWrite = 1
- PCWriteCond = X
- IorD = X
- MemRead = 0
- MemWrite = 0
- MemToReg = X
- IRWrite = 0
- PCSource = 0
- ALUOp = 001
- ALUSrcA = 0
- ALUSrcB = 11
- RegWrite = 0
- RegDst = X

Question #3



- Given the datapath above, what signals would the control unit turn on and off in order to load a memory value into `$t0`?
- Note: Assumes that the address of this memory location has already been computed and sent to the memory address input.

Question #3



- Loading a memory value into `$t0`
  - Step #1: Determine source and destination.
  - Step #2: Path between source and destination.
- Note: This assumes that the address of this memory location has already been sent to the memory unit by some previous micro-operations.

Question #3 (cont'd)

- Read / Write signals:
  - `RegWrite` and `MemRead` high, all others low.
- Datapath signals:
  - `MemToReg` → 1
  - `RegDst` → 0
- Non-essential signals:
  - `IorD`, `PCSource`, `AluSrcA`, `AluSrcB`

Question #3 (cont'd)

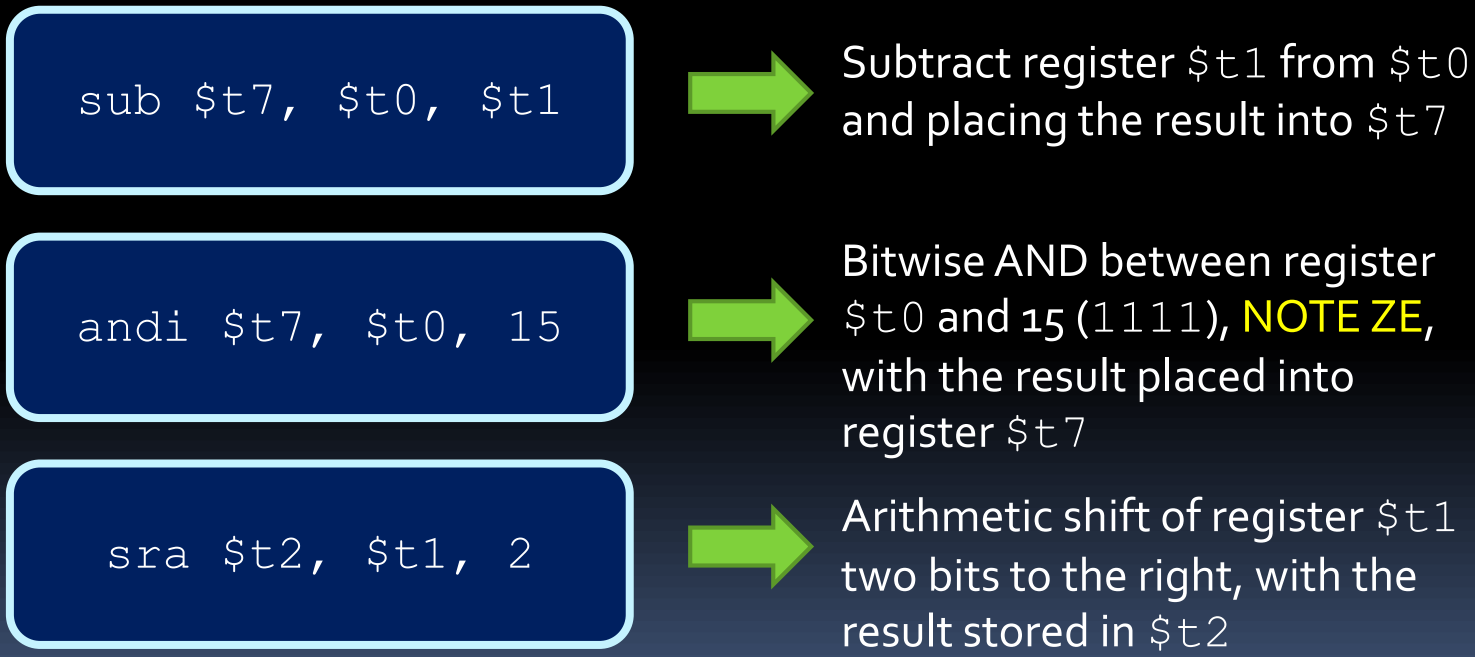
- PCWrite = 0
- PCWriteCond = 0
- IorD = X
- MemRead = 1
- MemWrite = 0
- MemToReg = 1
- IRWrite = 0

- PCSource = X
- ALUOp = XXX
- ALUSrcA = X
- ALUSrcB = XX
- RegWrite = 1
- RegDst = 0

Now do the same question, but for entire execution phase of lw $t0, 10($t1) instruction; i.e. Produce the address for memory as well.
Note: The highlighted signals cannot be left "don't care" now.

Question #4

- What are the following assembly language instructions doing?



```
sub $t7, $t0, $t1
```

Subtract register `$t1` from `$t0` and placing the result into `$t7`

```
andi $t7, $t0, 15
```

Bitwise AND between register `$t0` and 15 (`1111`), NOTE ZE, with the result placed into register `$t7`

```
sra $t2, $t1, 2
```

Arithmetic shift of register `$t1` two bits to the right, with the result stored in `$t2`

Question #5

- How do you translate the following assembly language instruction into machine code?

```
add $t7, $t0, $t1
```

R-type instruction!

opcode	rs	rt	rd	shamt	funct
6	5	5	5	5	6

Question #5

```
add $t7, $t0, $t1
```

- Step #1: The opcode
  - Arithmetic operations start with six 0's, and have the function identifier at the end.

```
000000 sssss ttttt ddddd XXXXX 100000
```

- Step #2: The register numbers
  - Remember that $t0 does not translate to register 0
  - The temporary registers start at register 8, so $t0 → 8, $t1 → 9 and $t7 →15

```
000000 01000 01001 01111 XXXXX 100000
```