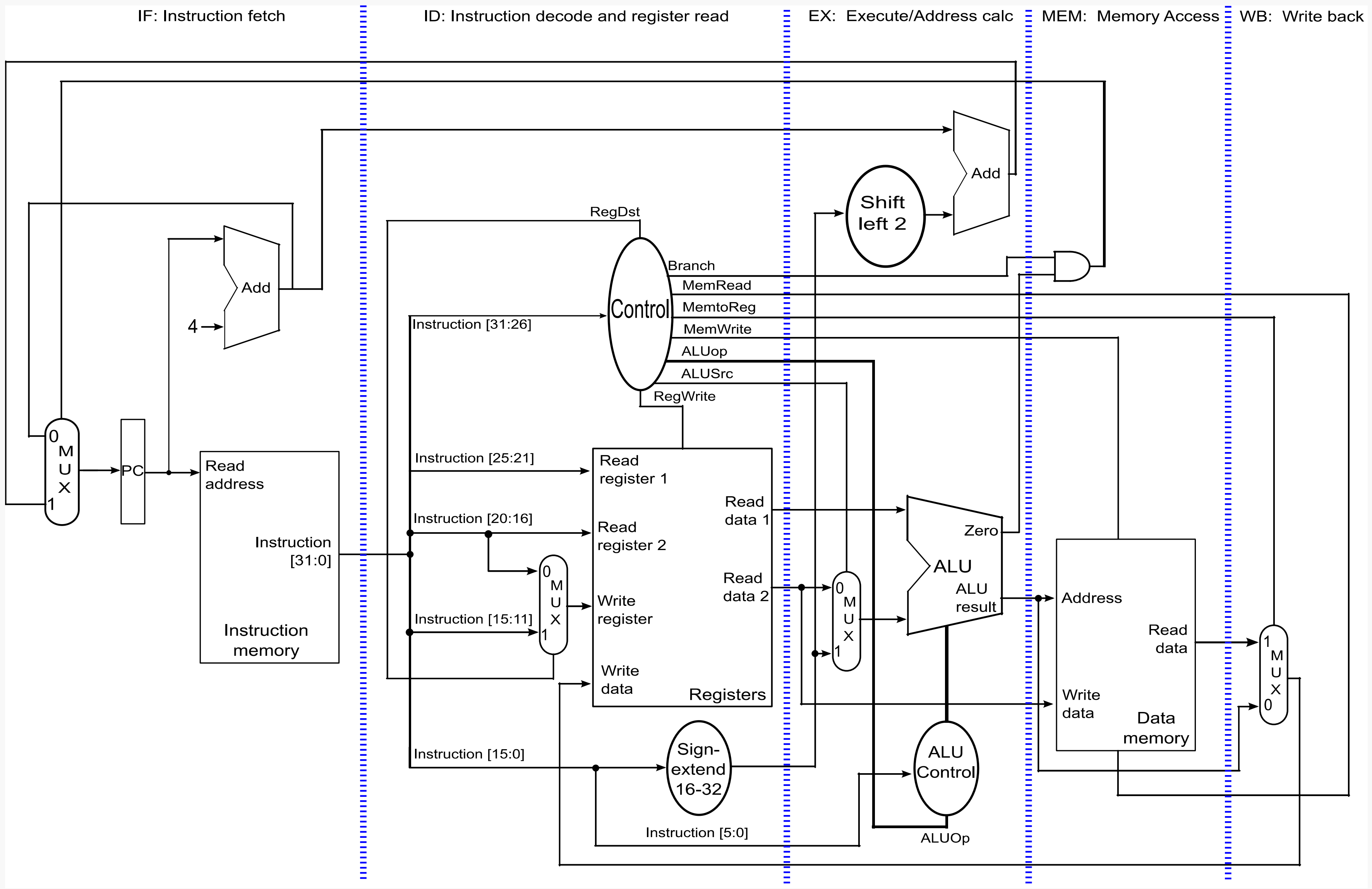
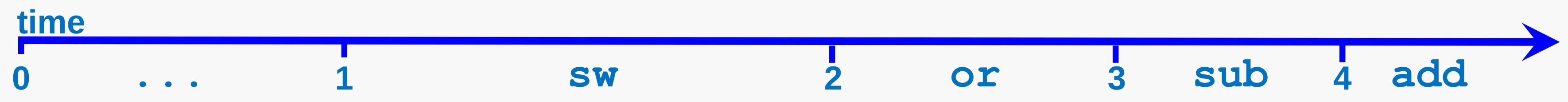
Current Design

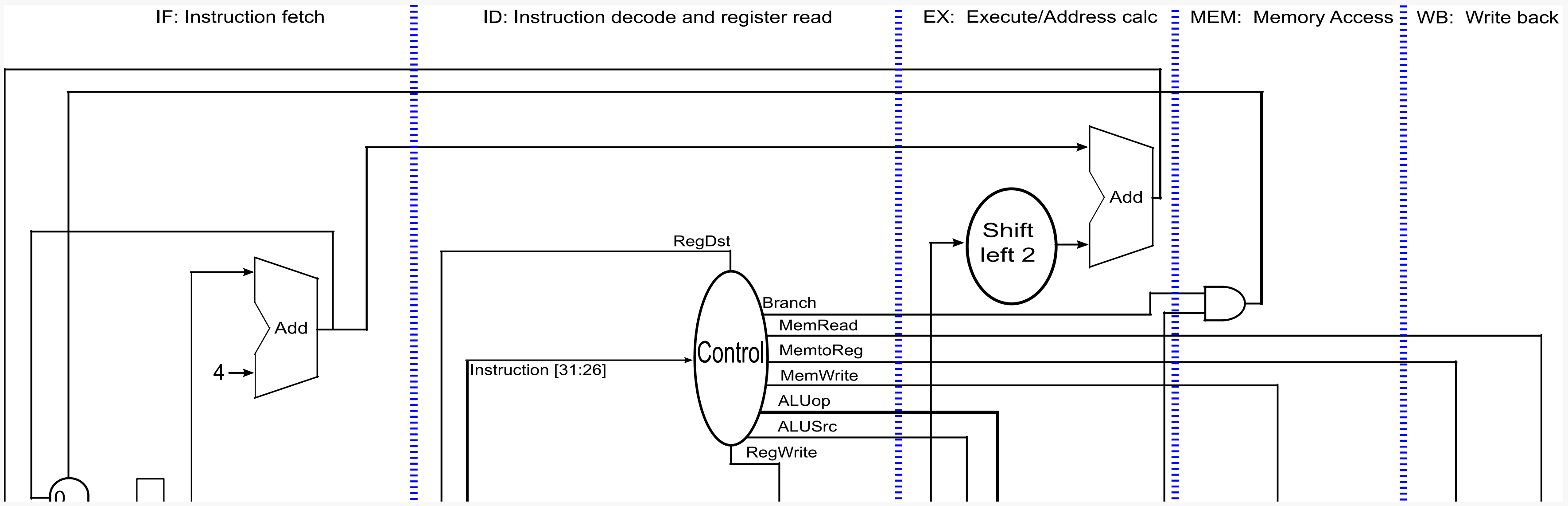


Pipeline Timing Issues

Consider executing:

```
add $t2, $t1, $t0       # needs MemWrite = 0
sub $t3, $t1, $t0       # needs MemWrite = 0
or  $t4, $t1, $t0       # needs MemWrite = 0
sw  $t2, 0($t0)         # needs MemWrite = 1
```

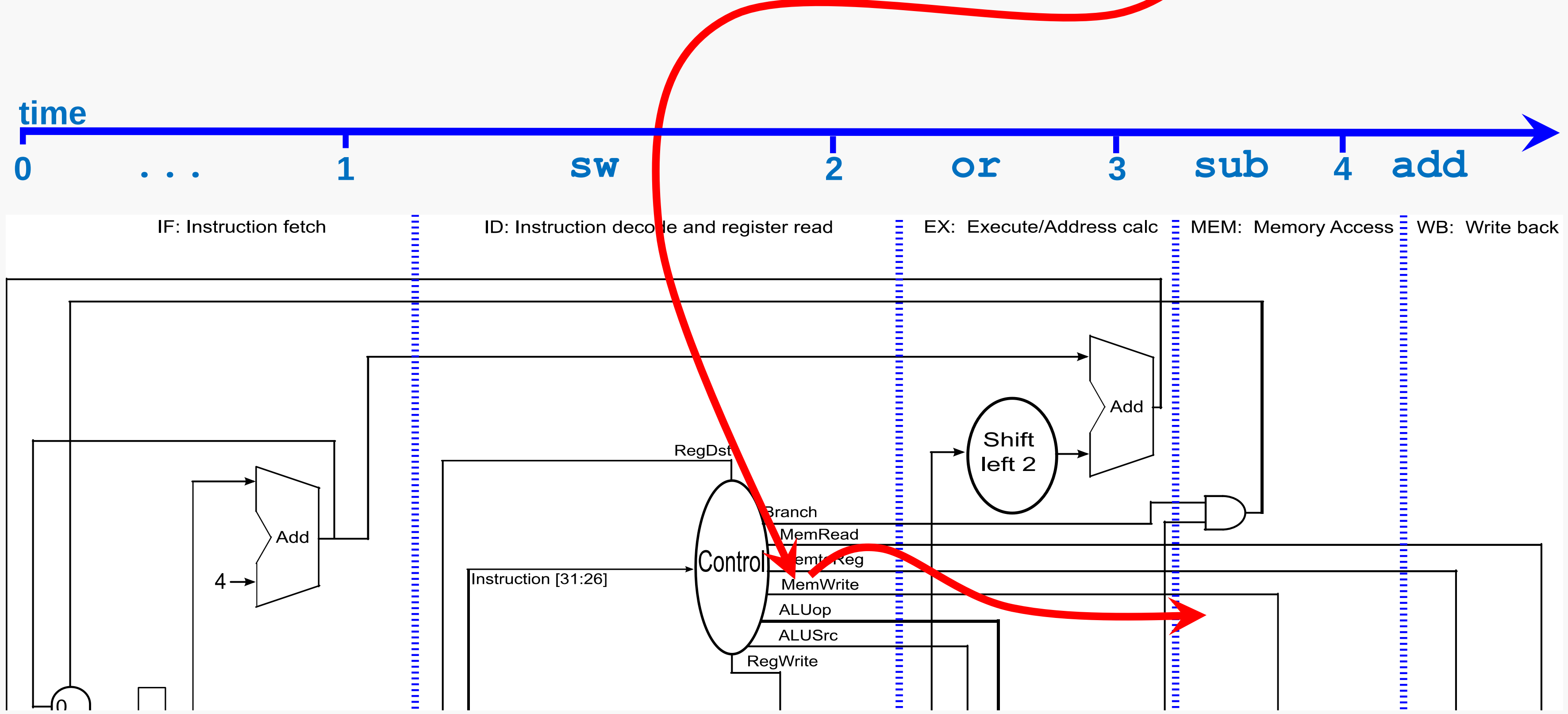




Pipeline Timing Issues

What happens during cycle 4? Among other things…

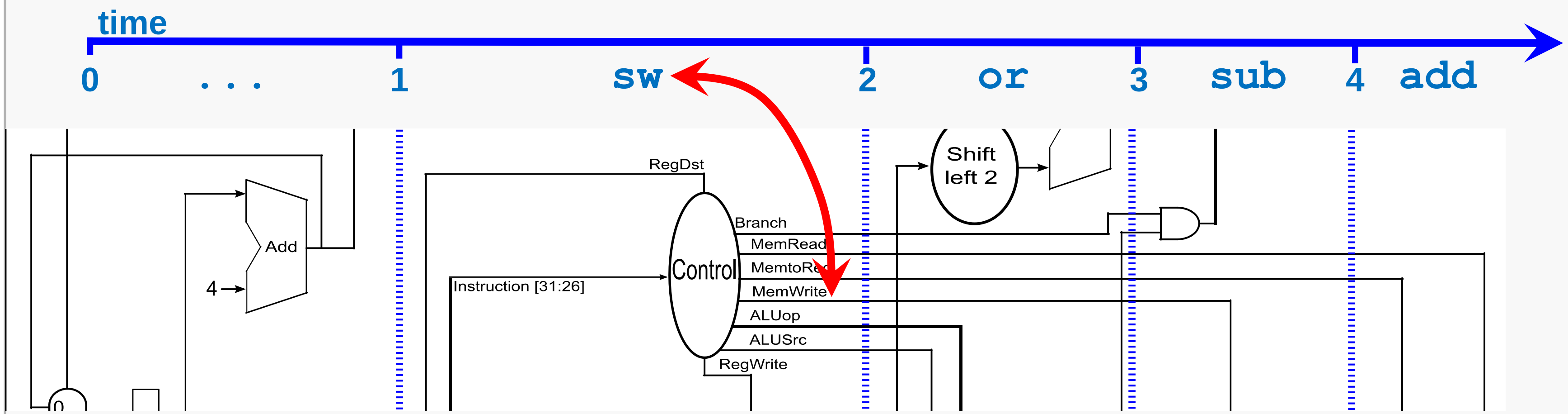
- `sw` reaches the ID stage, and Control sets MemWrite to 1
- so, a memory write will occur while `sub` is in the MEM stage
- and that's bad news…

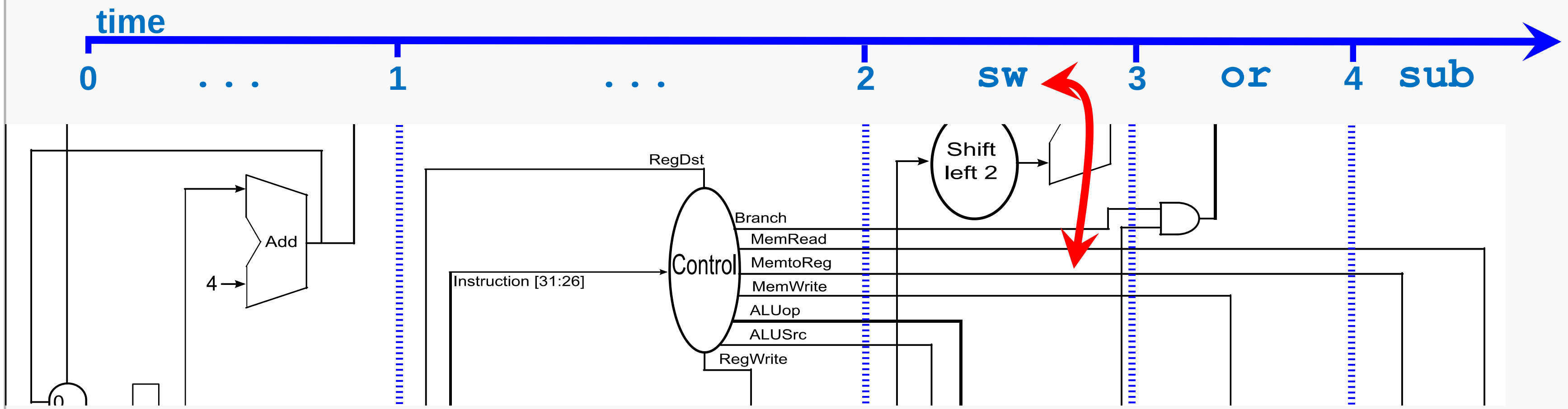


Pipeline Timing Issues

What needs to happen instead?

- the value of MemWrite that goes with `sw`…
- … needs to travel forward, stage to stage as `sw` does

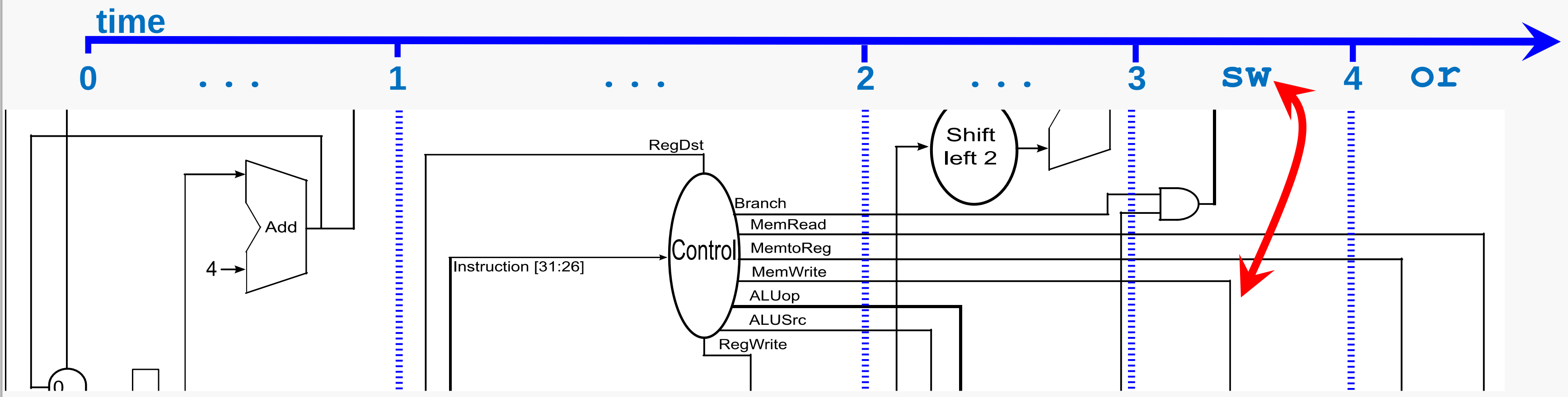




Pipeline Timing Issues

What needs to happen instead?

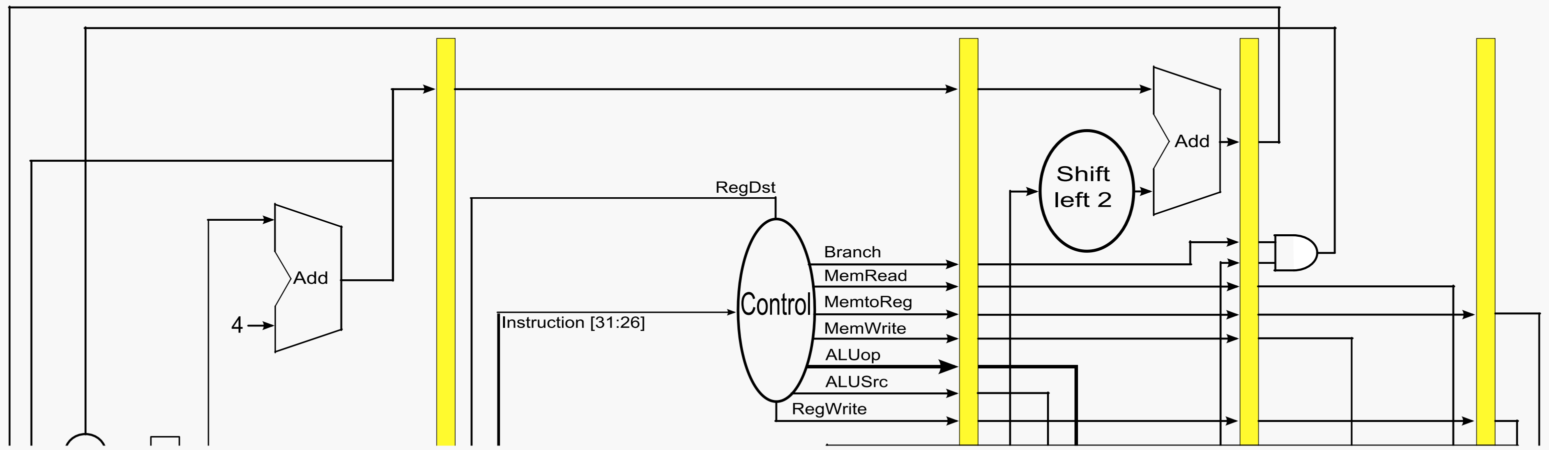
- the value of MemWrite that goes with `sw`…
- … needs to travel forward, stage to stage as `sw` does



So how do we make this happen?

Adding Buffers

Put storage buffers between adjacent stages:

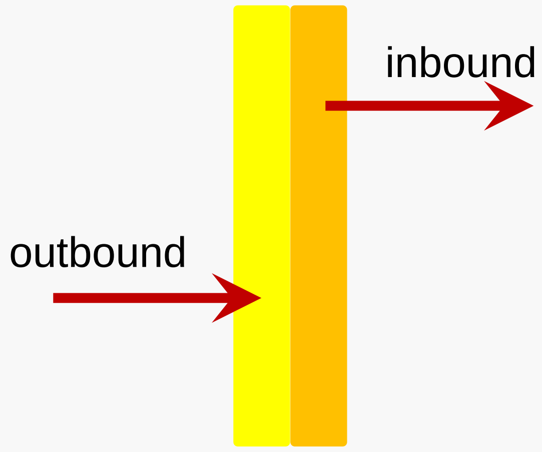


Control writes/reads on these buffers with the clock signal.

Read values entering a stage from the "inbound" buffer.

Write values exiting a stage to the "outbound" buffer.

So no signal (or data value) arrives before its time…



What now?

We have an idea: put buffers between adjacent stages and use those buffers to synchronize the operation of the pipeline stages

Are we sure we are handling all control signals properly… for every instruction?

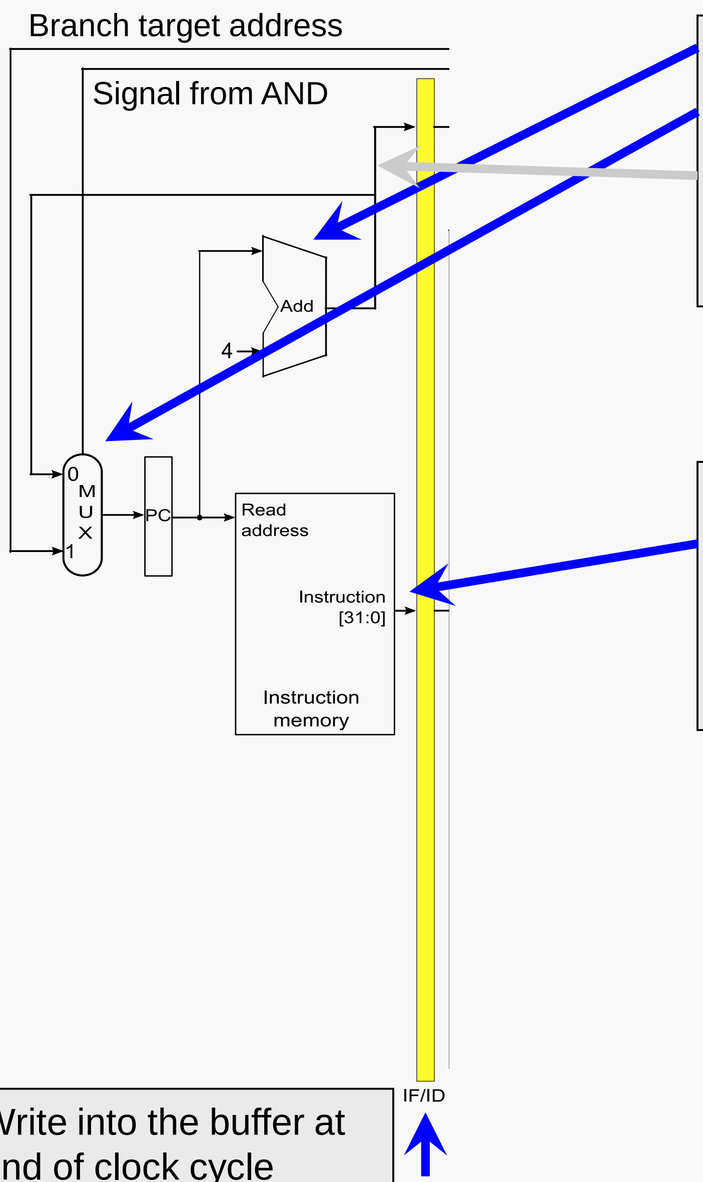
Are there any values, other than control signals that must be passed through the interstage buffers?

Signals and values:

- may move forward (from lower-numbered to higher-numbered stages)
- can they ever move backward?
- should they ever bypass the interstage buffers?

We will pick a particular instruction and consider its execution in detail…

IF Stage for LW



PC+4 is computed,

stored back into the PC,

also stored in the IF/ID buffer although it will not be needed in a later stage for LW

Instruction word is fetched from memory,

and stored in the IF/ID buffer because it will be needed in the next stage.

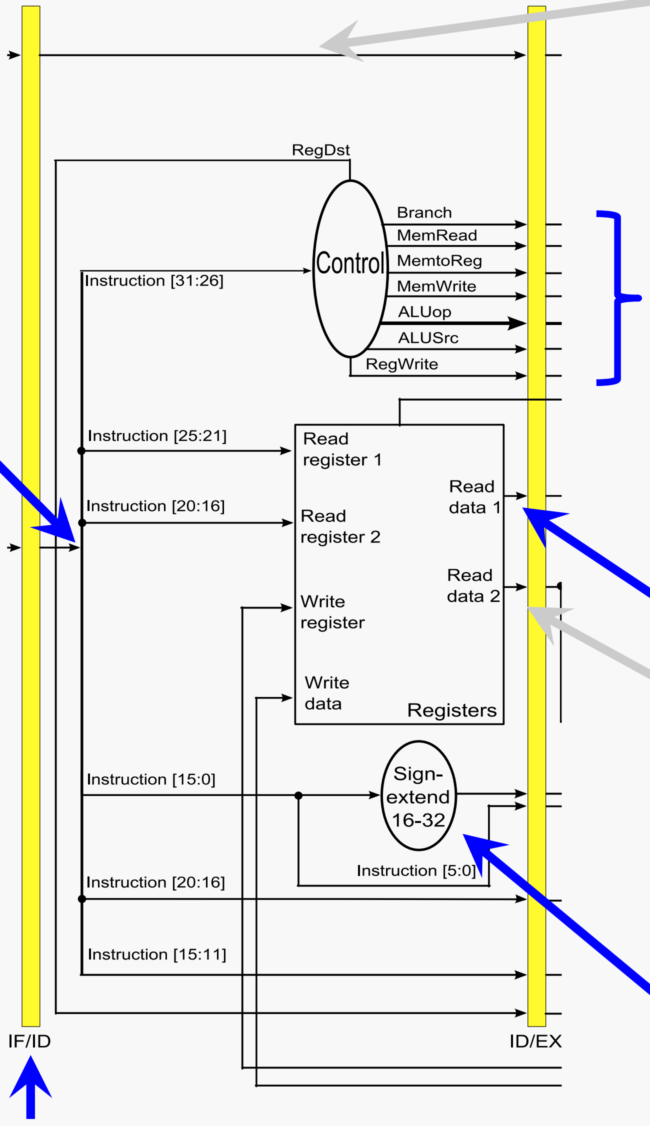
Write into the buffer at end of clock cycle

ID Stage for LW

Bits of load instruction are taken from IF/ID buffer, while

new instruction is being fetched back in stage 1.

PC+4 is passed forward to ID/EX buffer...



Control signals passed forward.

Some to be used in later stages.

Some to be passed back here.

Old design:

Set write register in ID stage, but use from WB stage.

That doesn't work now…

Read register #1 and #2 contents are fetched and stored in ID/EX buffer until needed in next stage… #2 won't be needed.

16-bit field is fetched from IF/ID buffer, then sign-extended, then stored in the ID/EX buffer for use in a later stage.

Read from the buffer at beginning of cycle

EX Stage for LW

PC+4 is taken from ID/EX buffer and added to branch offset…

Computed branch target address is stored in EX/MEM buffer to await decision in next stage... but won't be needed.

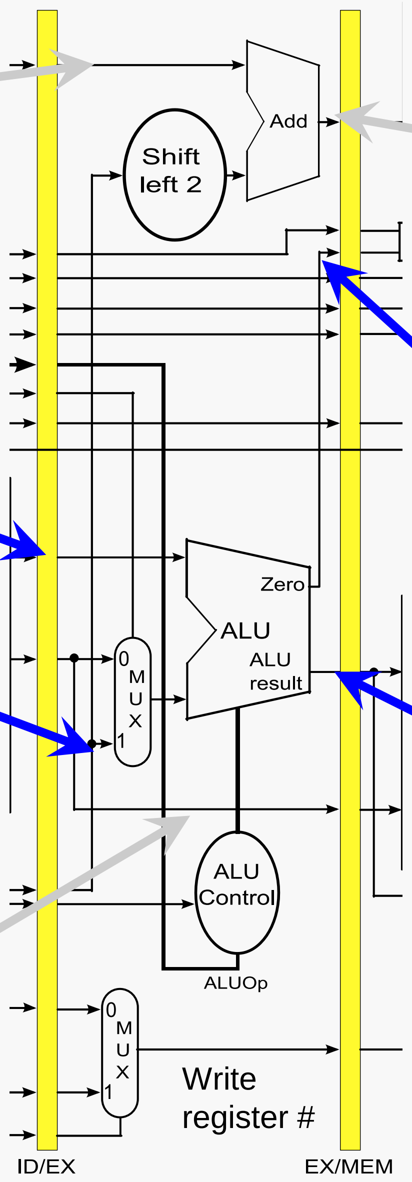
Read register #1 contents are taken from ID/EX buffer and provided to ALU.

Zero signal is stored in the EX/MEM buffer, but won't be needed.

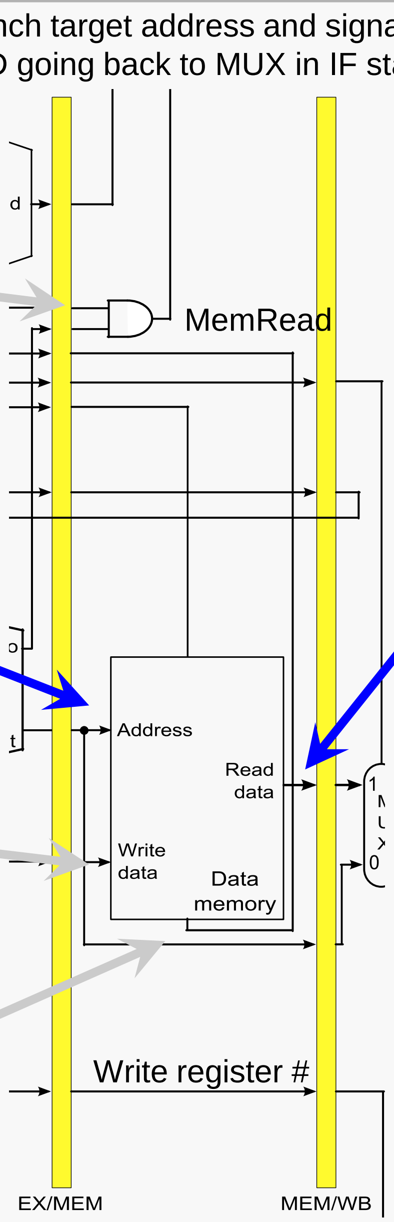
Extended 16-bit literal is provided to ALU as second operand

ALU result is stored in EX/MEM buffer for use as memory address in next stage.

Read register #2 is passed forward to EX/MEM buffer, for possible use in later stage… but won't be needed.



MEM Stage for LW



Zero line taken from EX/MEM buffer for branch control logic in this stage…

ALU result is taken from EX/MEM buffer and passed to Address port of data memory.

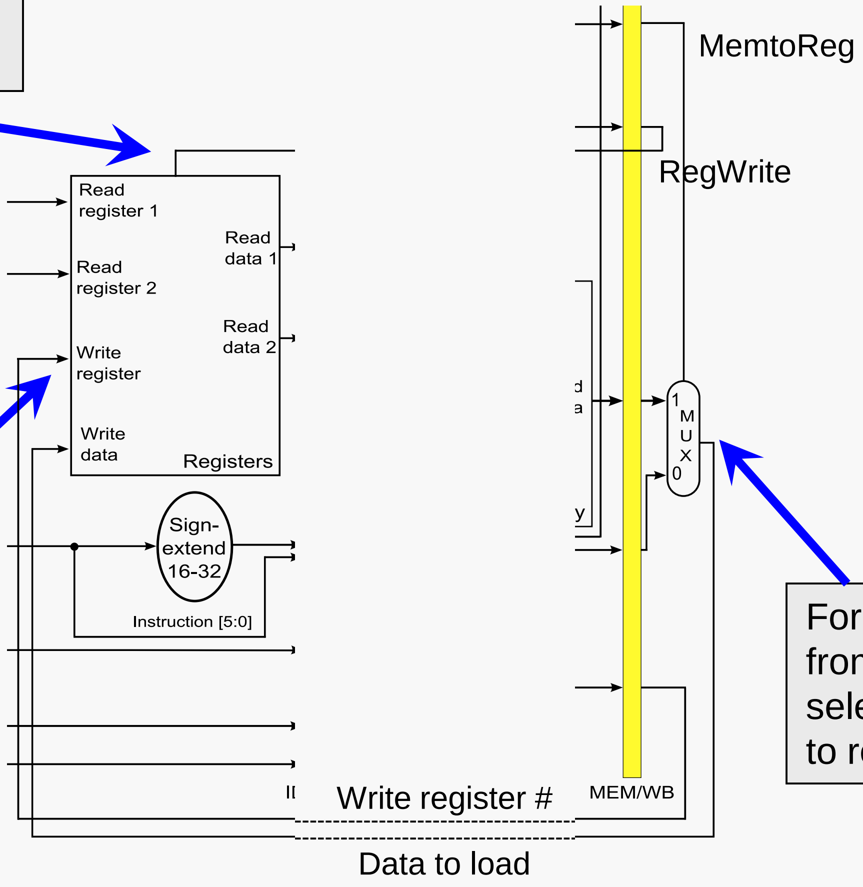
Read register #2 contents taken from EX/MEM buffer and passed to Write data port of data memory.

ALU result also stored in MEM/WB buffer for possible use in last stage…

Value on Read data port of data memory is stored in MEM/WB buffer, awaiting decision in last stage..

WB Stage for LW

The RegWrite signal is the one that goes with the load instruction (because it was sent forward through the interstage buffers).

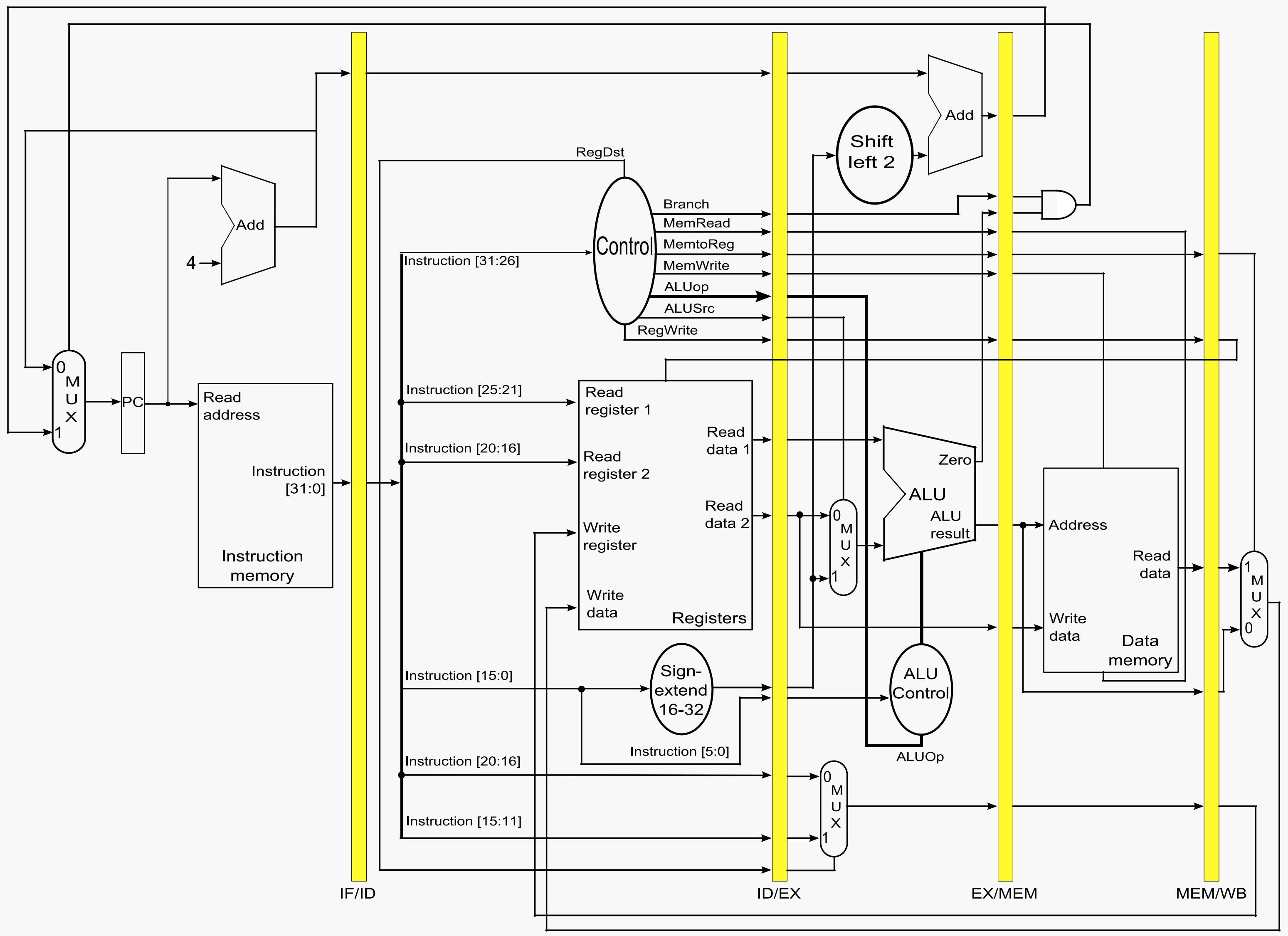


Write register port is now seeing the register number for the load instruction.

For load instructions, value from data memory is selected and passed back to register file.

Current Design

Here's our revised configuration including the buffers:



What now?

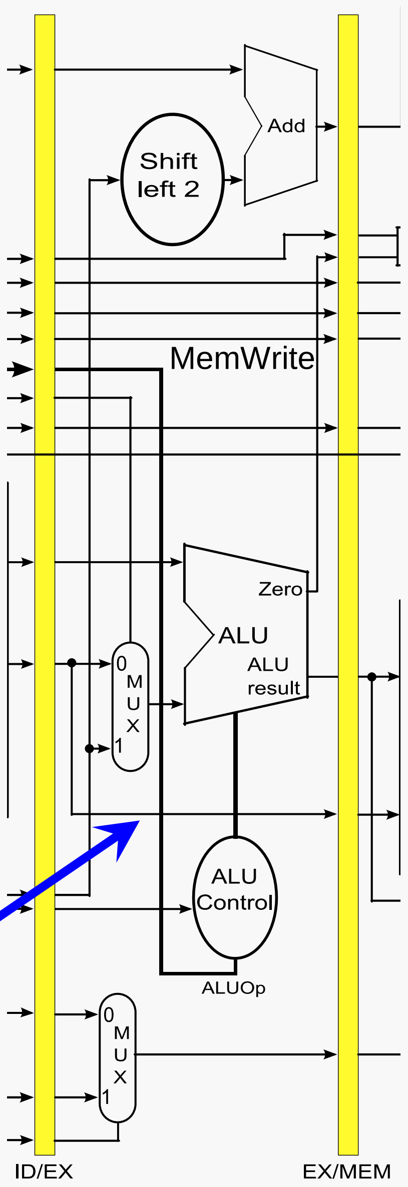
Would considering the execution of a different instruction yield new insights?

We will consider a similar instruction next: SW

Now, the IF stage is the same for all instructions, so we'll ignore it.

EX Stage for SW

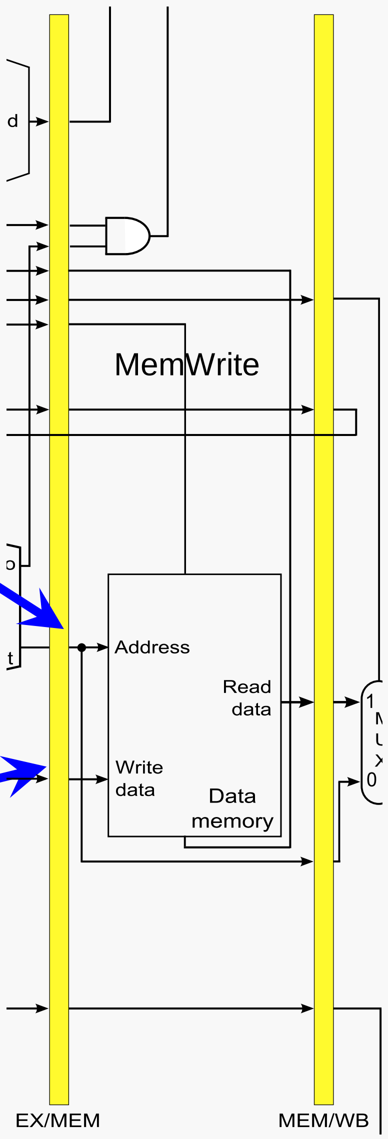
Almost the same as for LW…



Read register #2 is passed forward to EX/MEM buffer, for use in later stage… for SW this will be needed.

MEM Stage for SW

Almost the same as for LW…

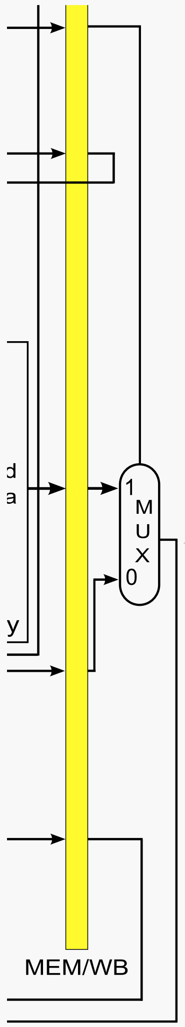


ALU result taken from EX/MEM buffer and passed to Address port of data memory.

Read register #2 contents taken from EX/MEM buffer and passed to Write data port of data memory.

WB Stage for SW

Not relevant for SW …



For SW instructions, no value will be written to the register file… doesn't really matter which value we send back…

Questions to Ponder

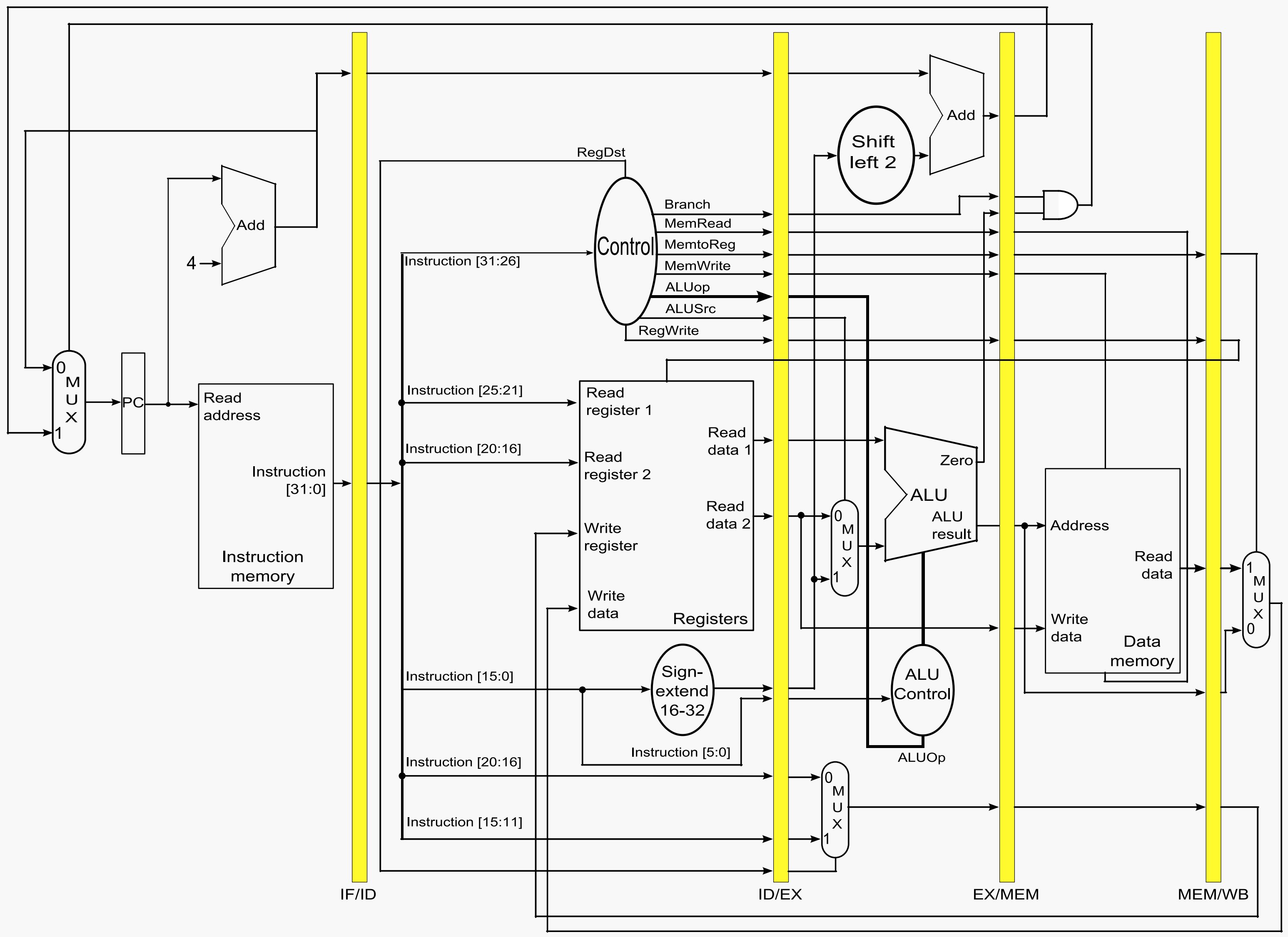
Can you repeat this analysis for other sorts of instructions, identifying in each stage what's relevant and what's not?

How much storage space does each interstage buffer need? Why?

Could adding interstage buffers affect the clock cycle? Why?

Summary

Here's our preliminary configuration for the buffers:



RegDst
Control
Branch
MemRead
MemtoReg
MemWrite
ALUop
ALUSrc
RegWrite
Add
4
Shift left 2
Add
0 MUX 1
PC
Read address
Instruction [31:0]
Instruction memory
Instruction [31:26]
Instruction [25:21]
Instruction [20:16]
Instruction [15:0]
Instruction [20:16]
Instruction [15:11]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Sign-extend 16-32
Instruction [5:0]
ALU Control
ALUOp
ALU
Zero
ALU result
Address
Read data
Write data
Data memory
IF/ID
ID/EX
EX/MEM
MEM/WB