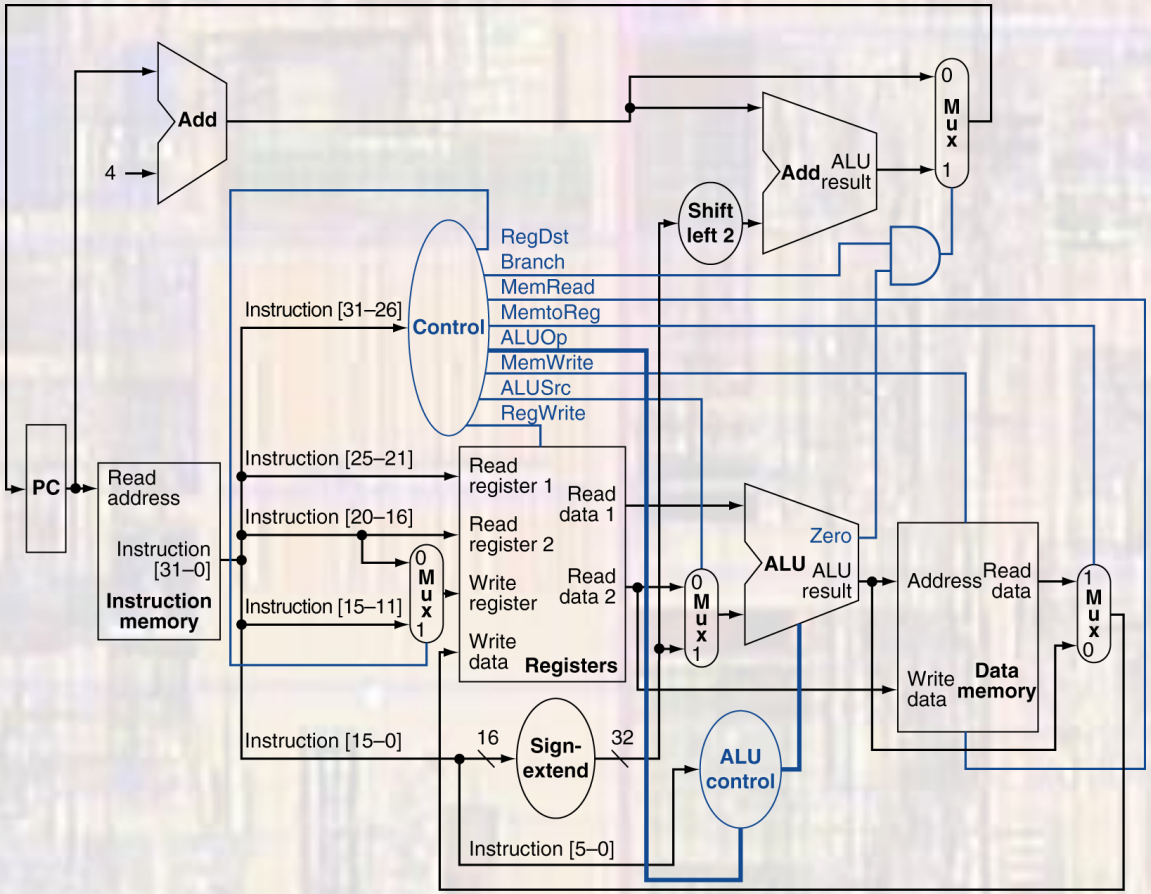
Processor Architecture Pipeline

Last modified 4/4/24

Pipelining

- Simple Datapath



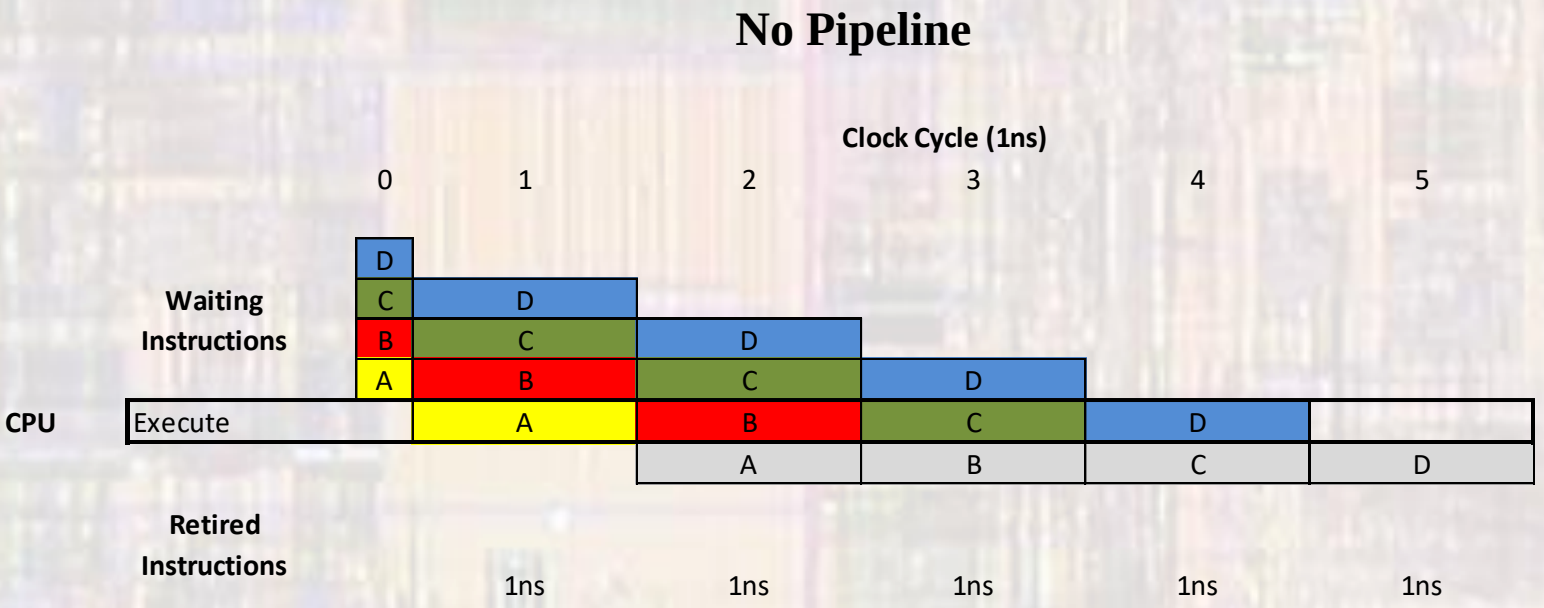
Pipelining

- 5 Stages of Instruction Execution
  - Fetch (IF)
  - Decode / Register Access (ID)
  - Execute (EX)
  - Memory Access (MEM)
  - Write Back (WB)

Pipeline these at 1 stage each

Pipelining

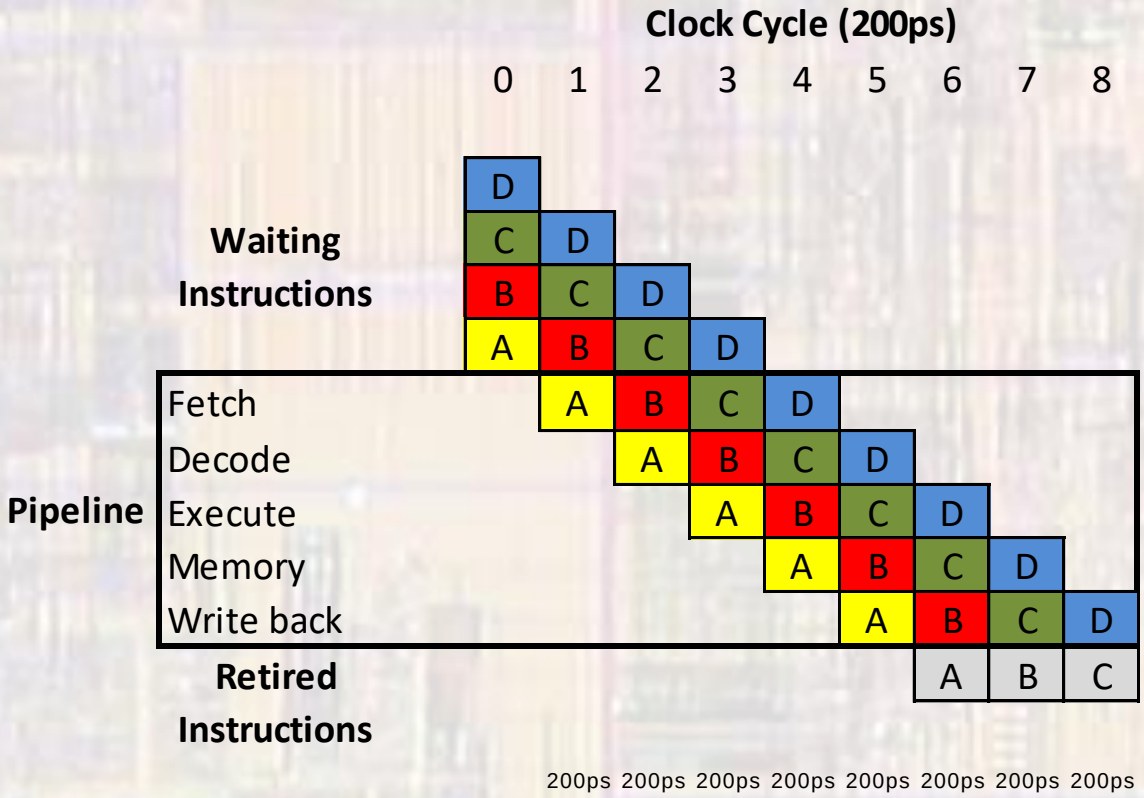
- Pipelining
  - Complete each instruction before starting the next
  - 1ns to complete each instruction



Execute = fetch instruction, decode, execute, mem, write back

Pipelining

- Pipelining
  - Break complex tasks into smaller chunks
  - Start the next instruction as soon as each subtask is complete



Pipelining

- Pipeline Performance
  - Pipelining does not reduce the time to execute an instruction (1ns in this example)
    - In fact – it usually increases the instruction execution time due to costs of implementing the pipeline
  - Pipelining does increase the instruction throughput
    - 1 instruction completes every 200ps

No Pipeline

Time	1000	1000	1000
IF/ID/EX/MEM/WB	1	2	3

1 inst

Pipeline

Time	200	200	200	200	200	200	200	200	200	200	200	200	200	200	200
IF	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ID		1	2	3	4	5	6	7	8	9	10	11	12	13	14
EX			1	2	3	4	5	6	7	8	9	10	11	12	13
MEM				1	2	3	4	5	6	7	8	9	10	11	12
WB					1	2	3	4	5	6	7	8	9	10	11

1 inst

Pipelining

- Pipeline Performance

5 stages
1000ps non-pipelined
200ps/stage pipelined

  - Non-pipelined
    - 1M Instructions → 1M * 1000ps = 1ms
  - Pipelined (5 stage)
    - 1M Instructions → Fill Time + Execute time
    - 1M Instructions → 4 * 200ps + 1M * 200ps ≅ 200us
  - Faster completion time: 1/5
  - Overall throughput improvement of 5x

Pipelining

- Pipeline Performance – with penalty

5 stages
1000ps non-pipelined
240ps/stage pipelined

  - Non-pipelined
    - 1M Instructions → 1M * 1000ps = 1ms
  - Pipelined (5 stage w/20% penalty per stage)
    - 1M Instructions → Fill Time + Execute time
    - 1M Instructions → 4 * 240ps + 1M * 240ps ≅ 240us
  - Faster completion time: 1/4.2
  - Overall throughput improvement of 4.2x

Pipelining

- Pipeline Performance
  - Not all instructions need to use all the processing stages

Instruction	IF	ID/RR	EX	MEM	WB
ADD	X	X	X		X
OR	X	X	X		X
LW	X	X	X	X	X
SW	X	X	X	X	
BEQ	X	X	X		

3, 4, or 5 stages required

  - Can't take advantage of this in either case because we need a consistent clock frequency

Pipelining

- Pipeline Performance
  - Processing stages typically do not all take the same amount of time

Stage	IF	ID/RR	EX	MEM	WB
Delay	200ps	100ps	200ps	200ps	100ps

  - Non-pipelined
    - 800ps clock period
  - Pipelined
    - Need to account for worst case cycle time
    - 200ps clock period

Pipelining

- Pipeline Performance
  - Non-pipelined
    - 1M Instructions → 1M * 800ps = 800us
  - Pipelined (5 stage w/20% penalty per stage)
    - 1M Instructions → Fill Time + Execute time
    - 1M Instructions → 4 * 240ps + 1M * 240ps ≅ 240us
  - Faster completion time: 240/800
  - Overall throughput improvement of 3.33x

Pipelining

- Pipeline Performance

15 stages
800ps non-pipelined
(1000ps/15)*1.2 = 84ps/stage pipelined

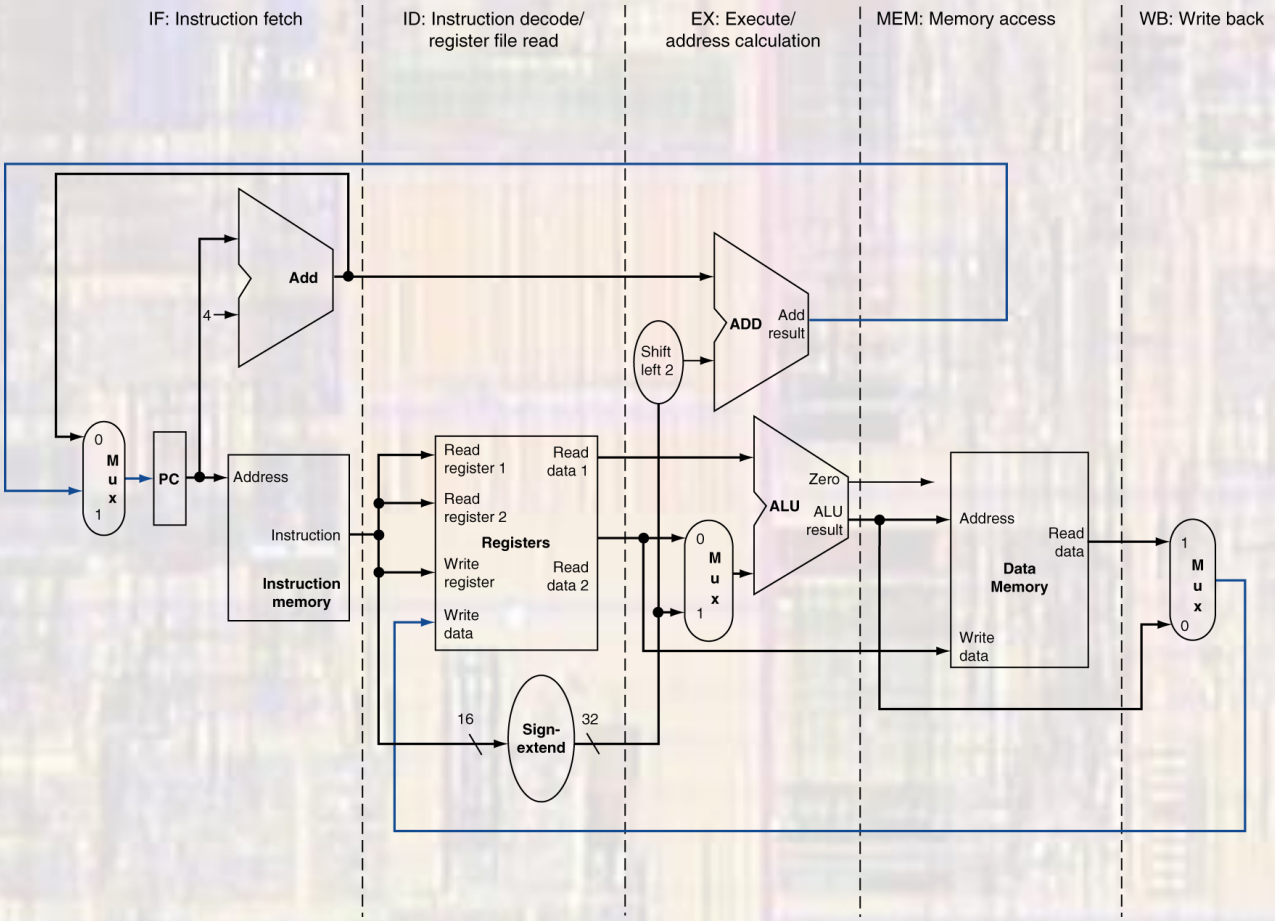
  - Non-pipelined
    - 1M Instructions → 1M * 800ps = 800us
  - Pipelined (15 stage w/20% penalty per stage)
    - 1M Instructions → Fill Time + Execute time
    - 1M Instructions → 14 * 84ps + 1M * 84ps ≅ 84us
  - Faster completion time: 84/800
  - Overall throughput improvement of 9.52x

Pipelining

- MIPS Pipeline Considerations
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
  - Few and regular instruction formats
    - R, I, J
    - Can decode and read registers in one step - why?
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands
    - Memory access takes only one cycle

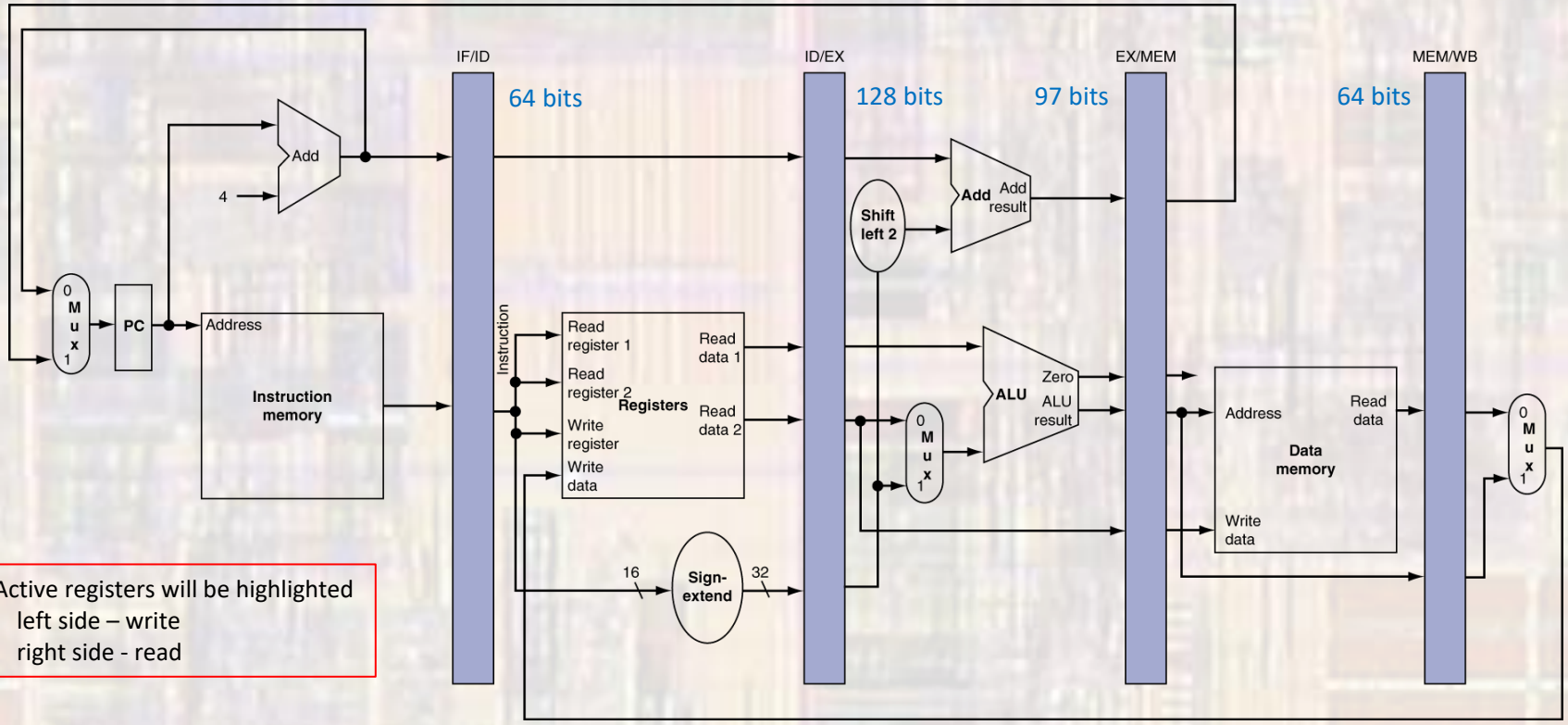
Pipelining

- Mapping the datapath to a pipeline



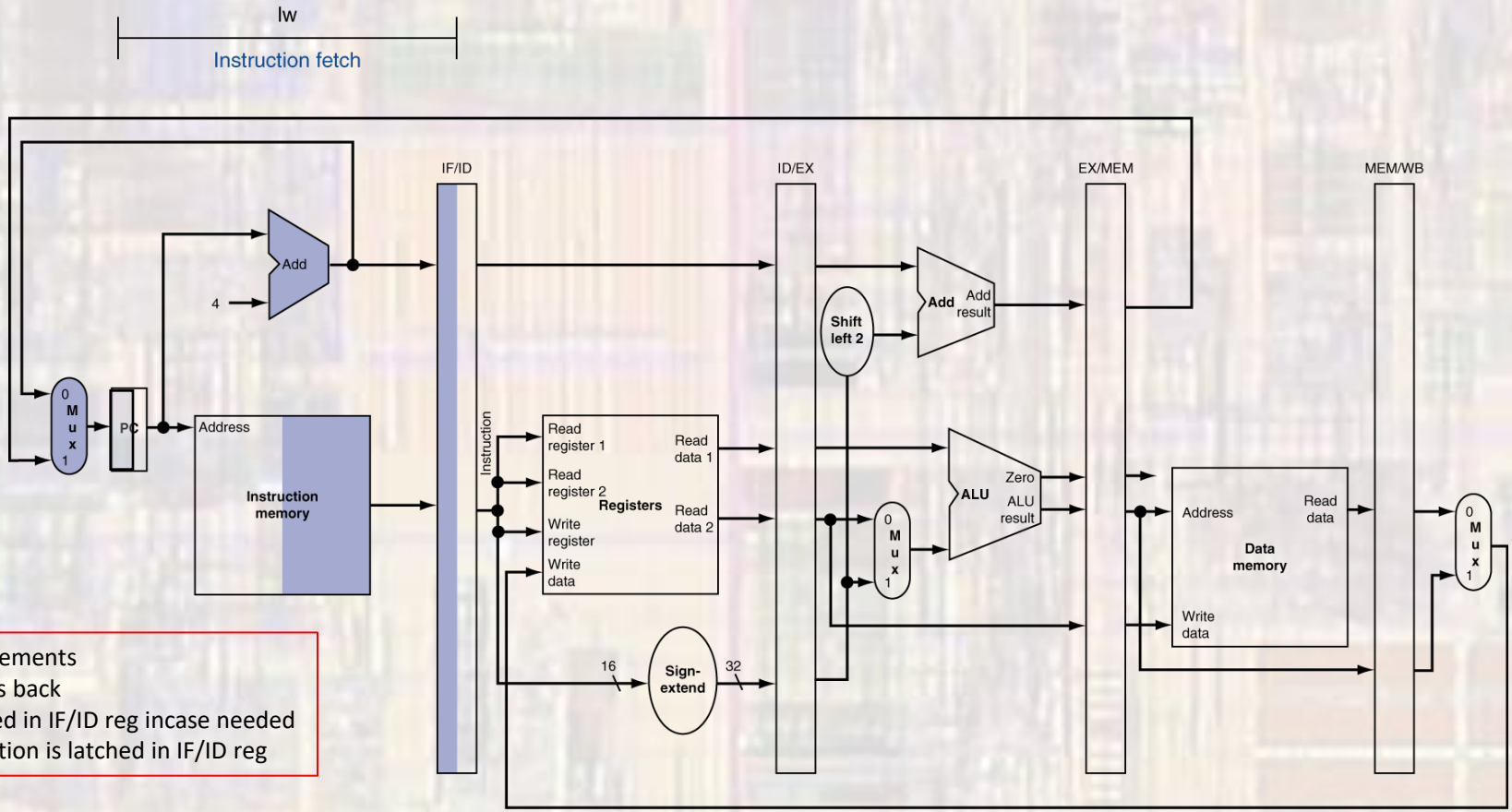
Pipelining

- Mapping the datapath to a pipeline
  - Registers are required to hold intermediate values between stages



Pipelining

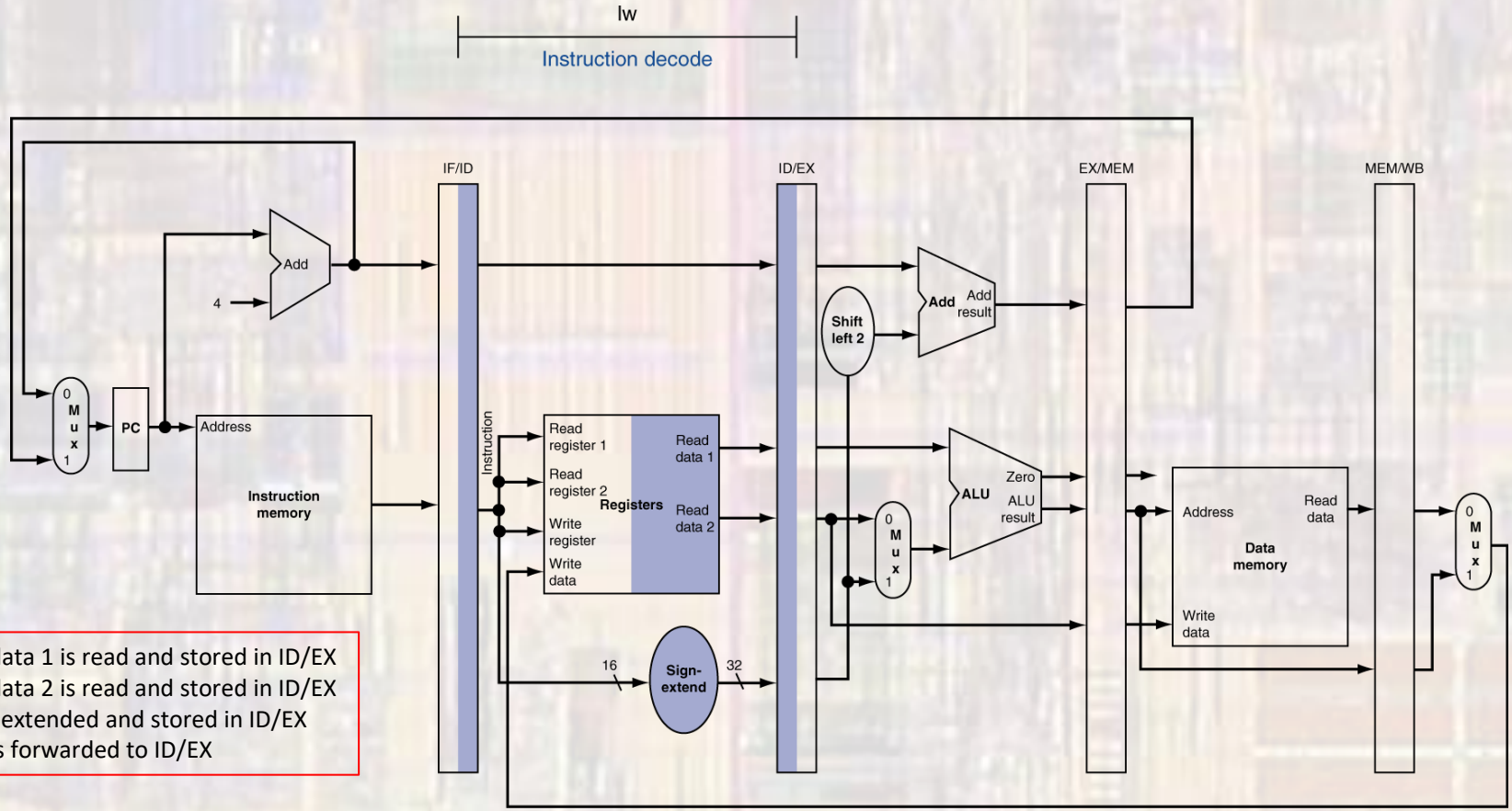
- Mapping the datapath to a pipeline
  - lw instruction - IF



PC increments
feeds back
stored in IF/ID reg incase needed
Instruction is latched in IF/ID reg

Pipelining

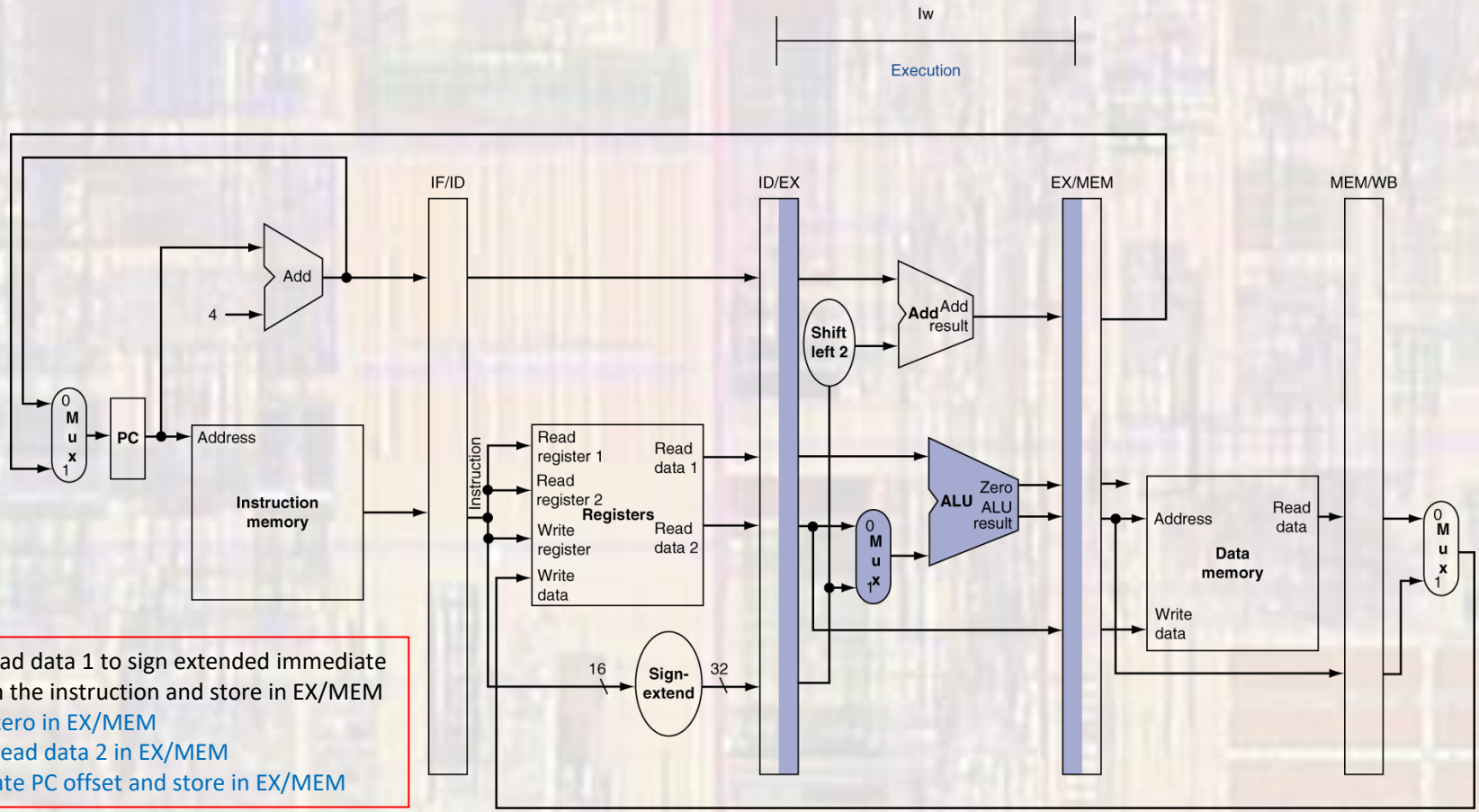
- Mapping the datapath to a pipeline
  - lw instruction – ID (instruction decode and register read)



Read data 1 is read and stored in ID/EX
Read data 2 is read and stored in ID/EX
Sign is extended and stored in ID/EX
PC+4 is forwarded to ID/EX

Pipelining

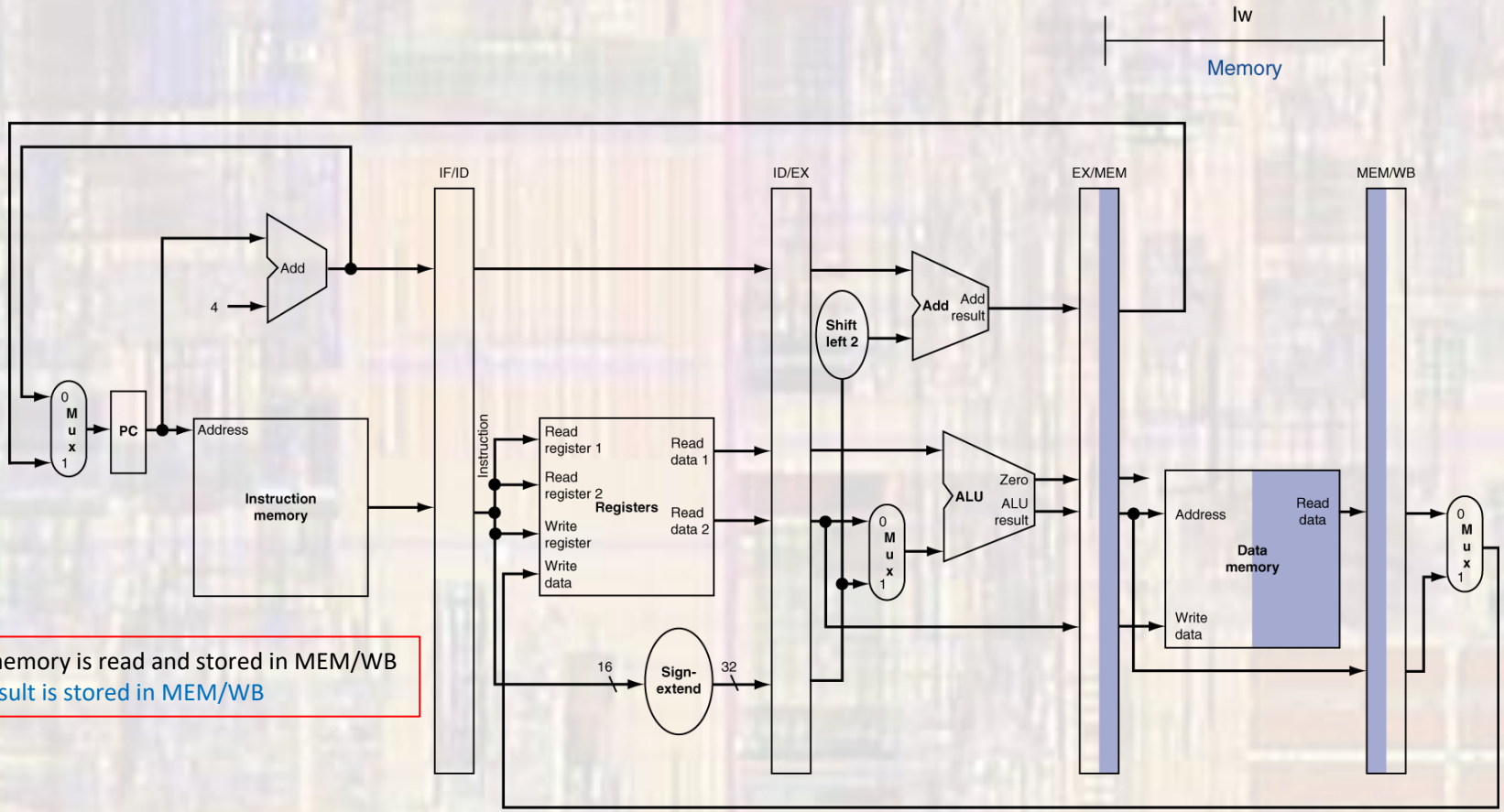
- Mapping the datapath to a pipeline
  - lw instruction – EX



Add read data 1 to sign extended immediate from the instruction and store in EX/MEM
Store zero in EX/MEM
Store read data 2 in EX/MEM
Calculate PC offset and store in EX/MEM

Pipelining

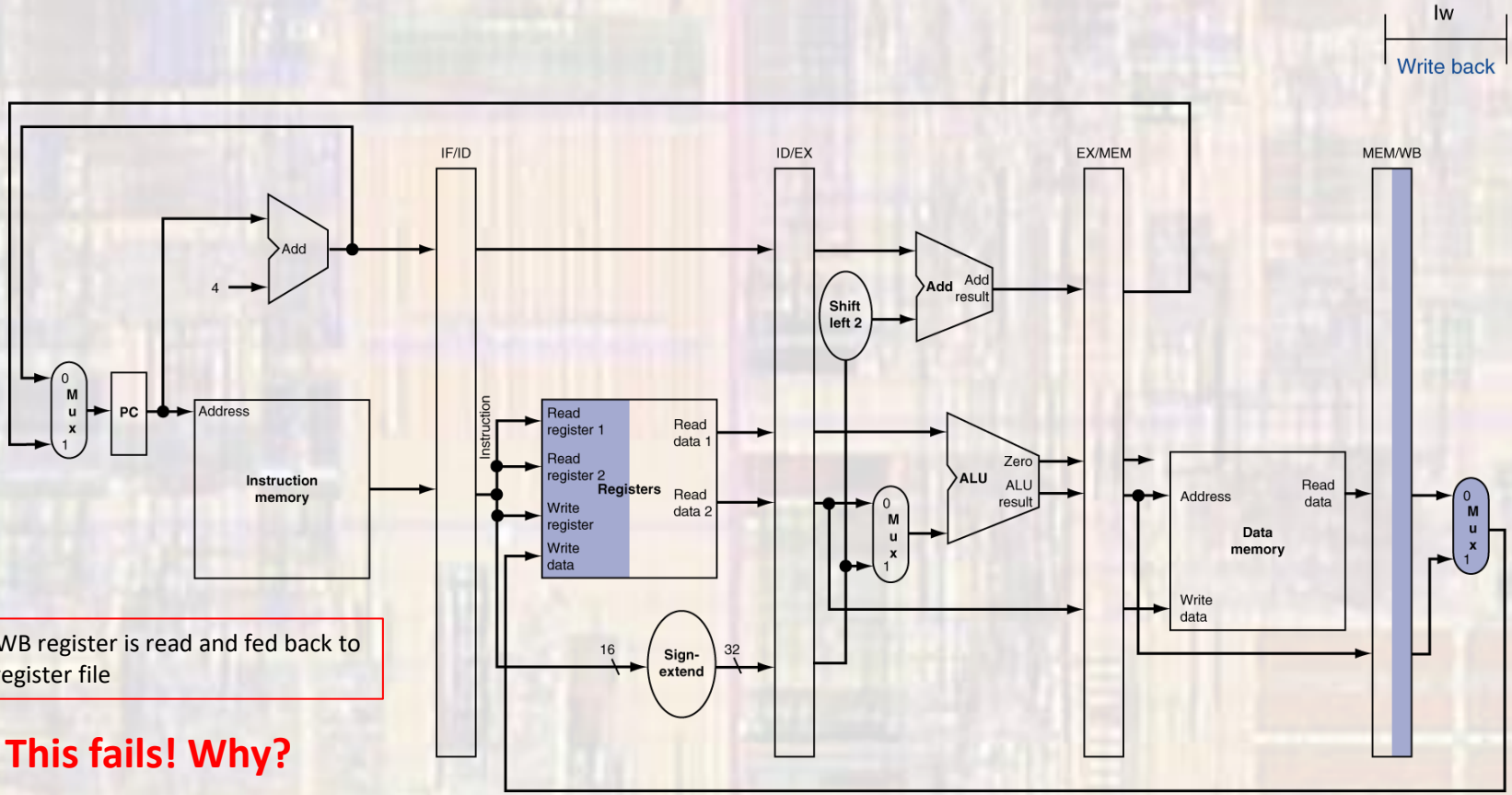
- Mapping the datapath to a pipeline
  - lw instruction – MEM



Data memory is read and stored in MEM/WB
ALU result is stored in MEM/WB

Pipelining

- Mapping the datapath to a pipeline
  - lw instruction – WB



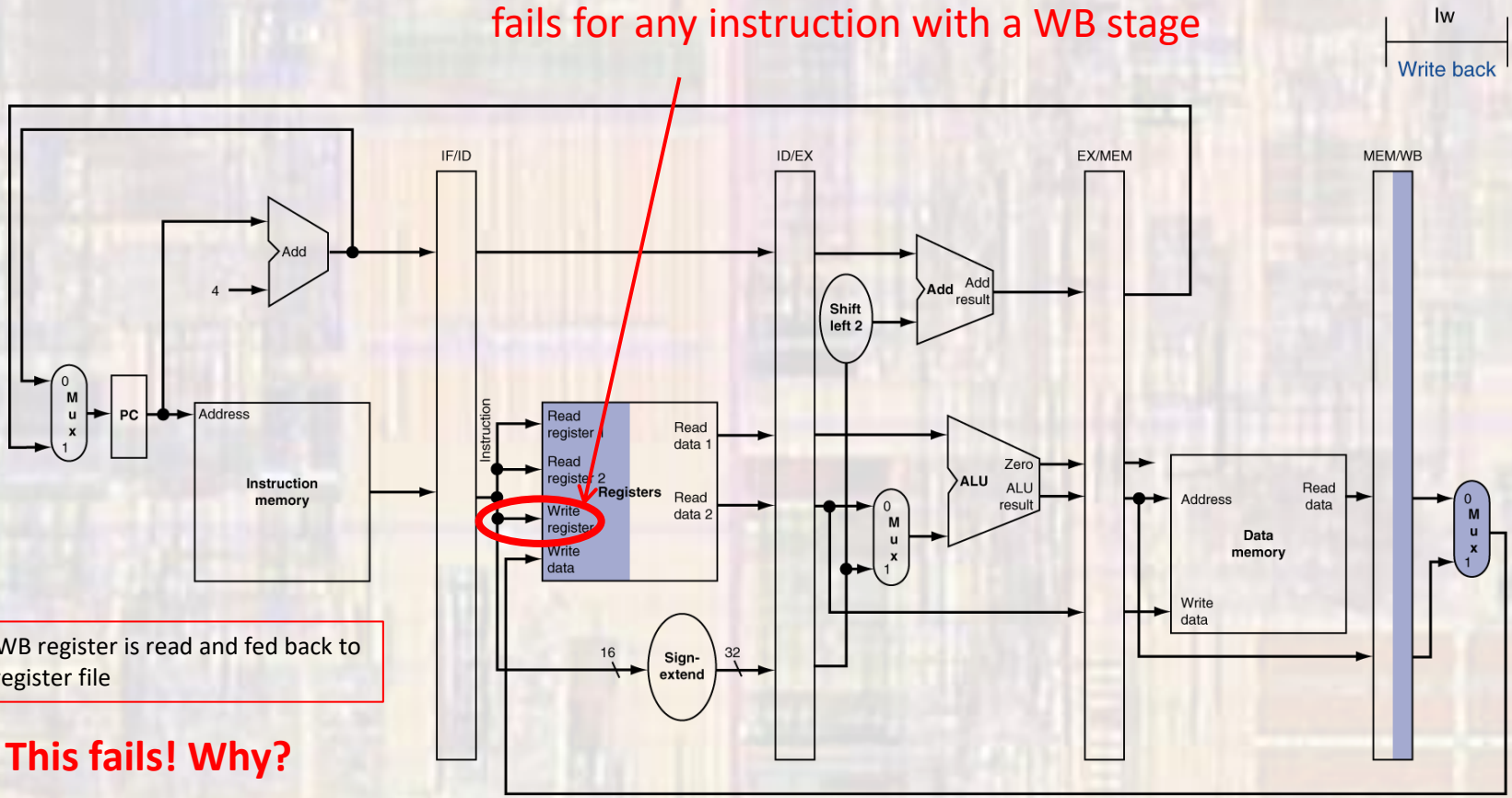
MEM/WB register is read and fed back to the register file

This fails! Why?

Pipelining

- Mapping the datapath to a pipeline
  - lw instruction – WB

register address for 3 instructions after lw fails for any instruction with a WB stage

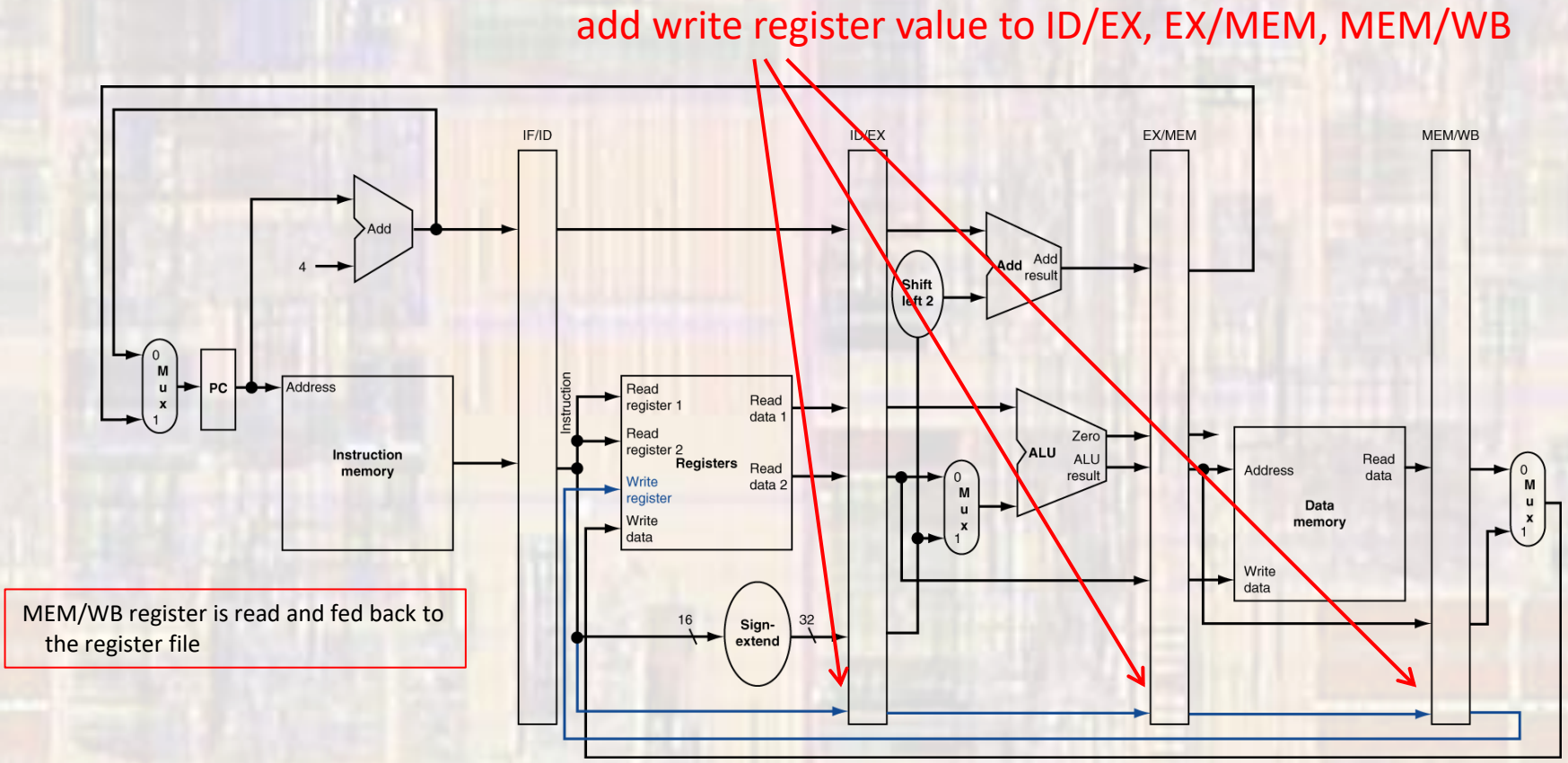


MEM/WB register is read and fed back to the register file

This fails! Why?

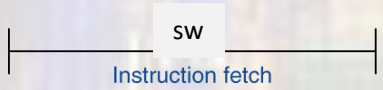
Pipelining

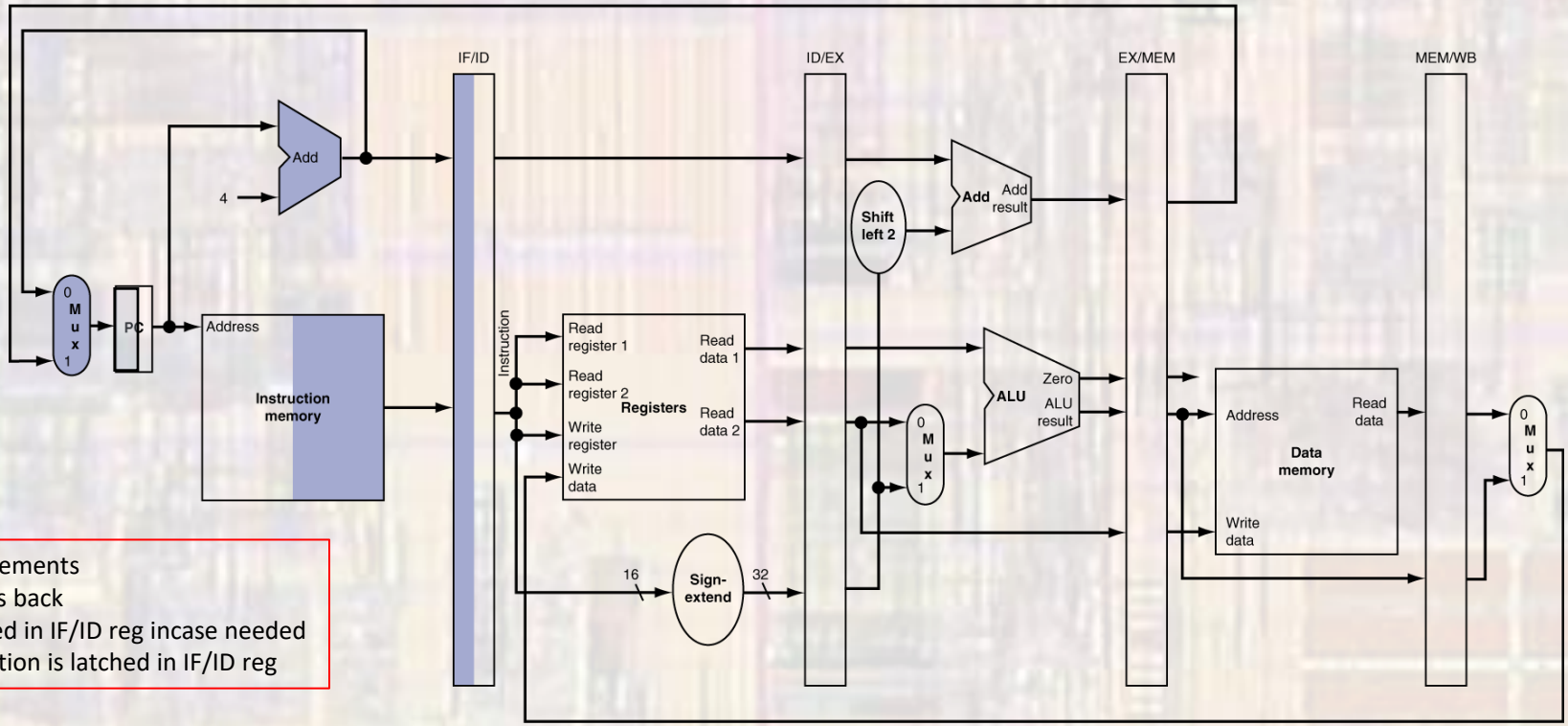
- Mapping the datapath to a pipeline
  - lw instruction – WB



Pipelining

- Mapping the datapath to a pipeline
  - sw instruction - IF

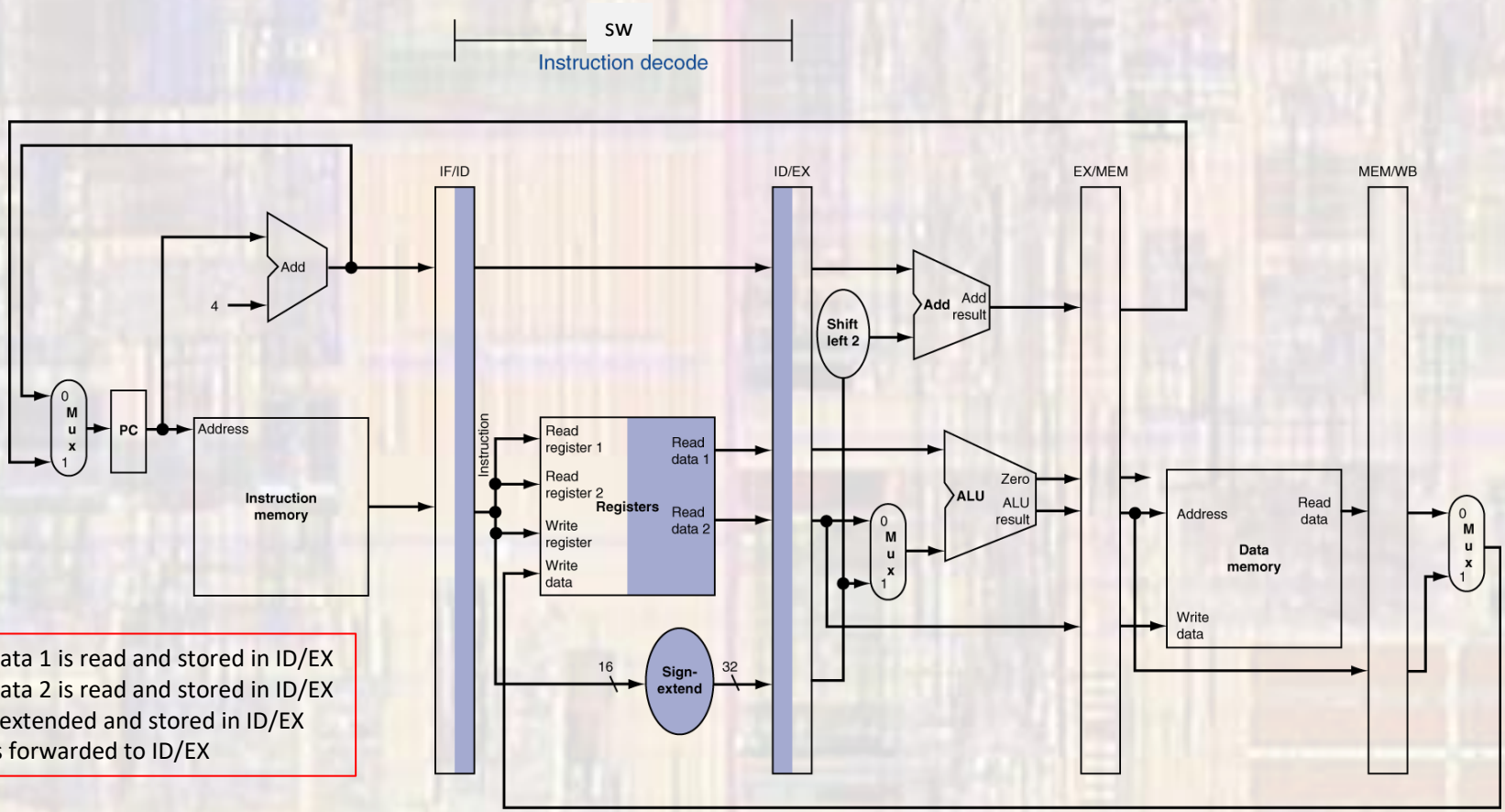




PC increments
feeds back
stored in IF/ID reg incase needed
Instruction is latched in IF/ID reg

Pipelining

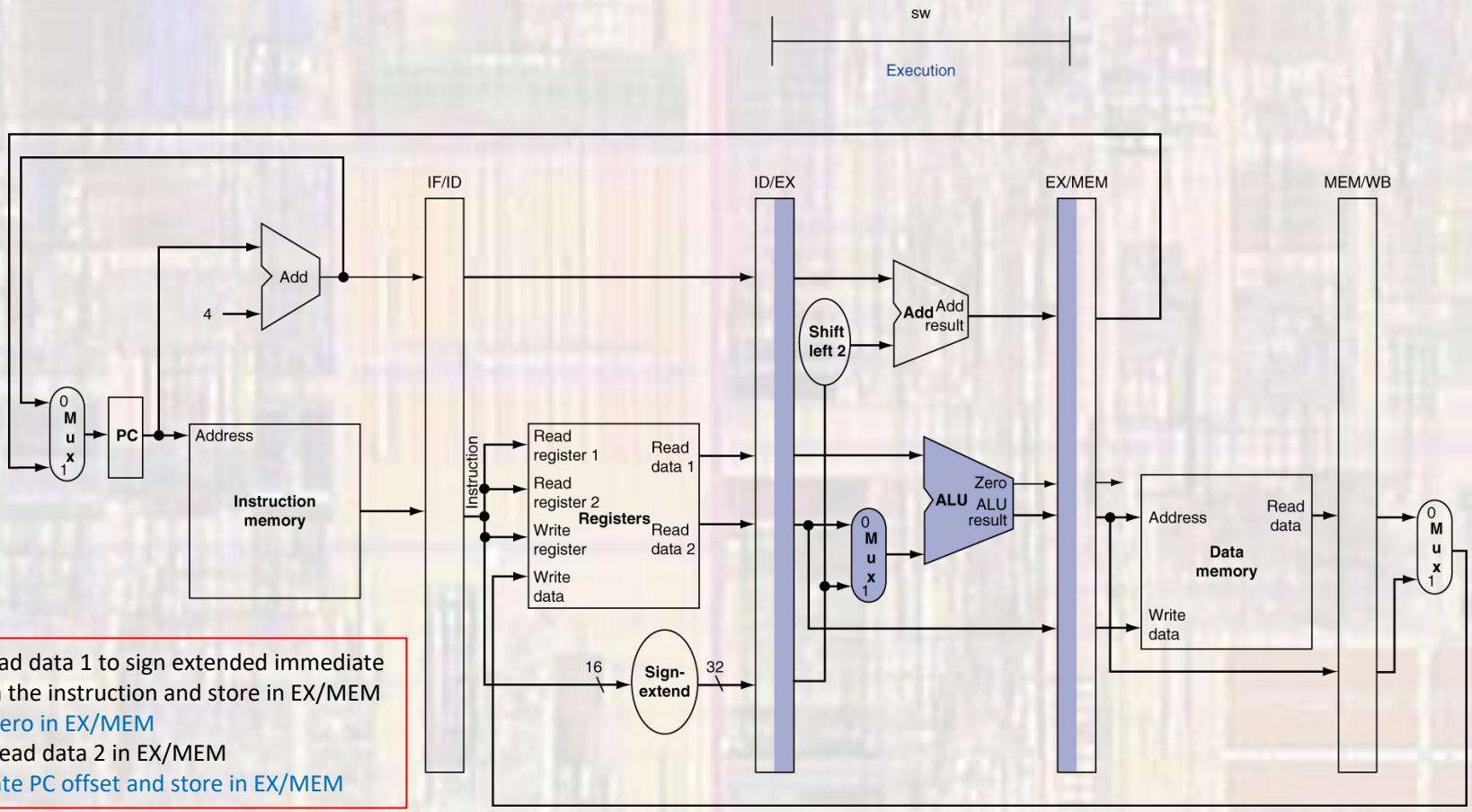
- Mapping the datapath to a pipeline
  - sw instruction – ID (instruction decode and register read)



Read data 1 is read and stored in ID/EX
Read data 2 is read and stored in ID/EX
Sign is extended and stored in ID/EX
PC+4 is forwarded to ID/EX

Pipelining

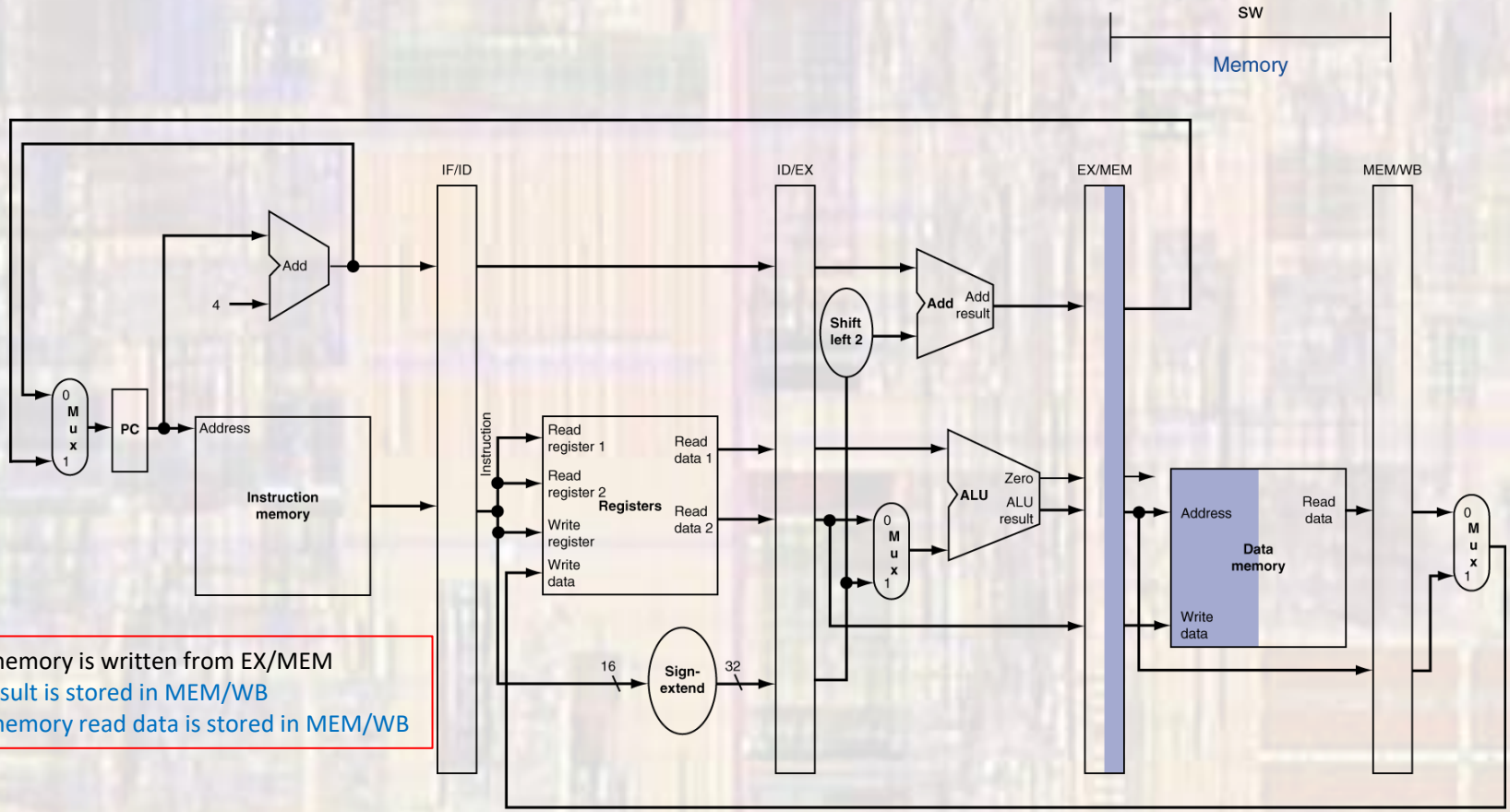
- Mapping the datapath to a pipeline
  - sw instruction – EX



Add read data 1 to sign extended immediate from the instruction and store in EX/MEM
Store zero in EX/MEM
Store read data 2 in EX/MEM
Calculate PC offset and store in EX/MEM

Pipelining

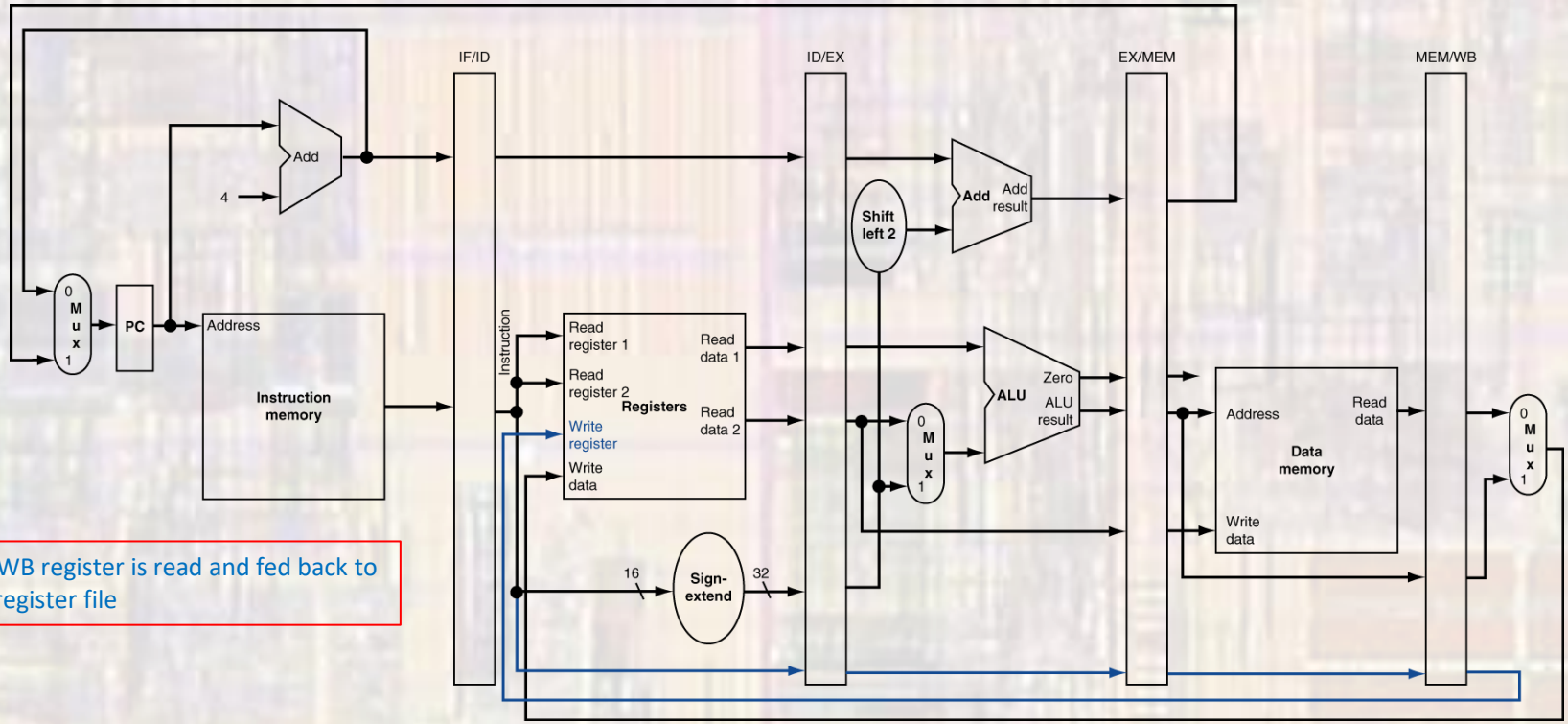
- Mapping the datapath to a pipeline
  - sw instruction – MEM



Data memory is written from EX/MEM
ALU result is stored in MEM/WB
Data memory read data is stored in MEM/WB

Pipelining

- Mapping the datapath to a pipeline
  - sw instruction – WB



MEM/WB register is read and fed back to the register file

Pipelining

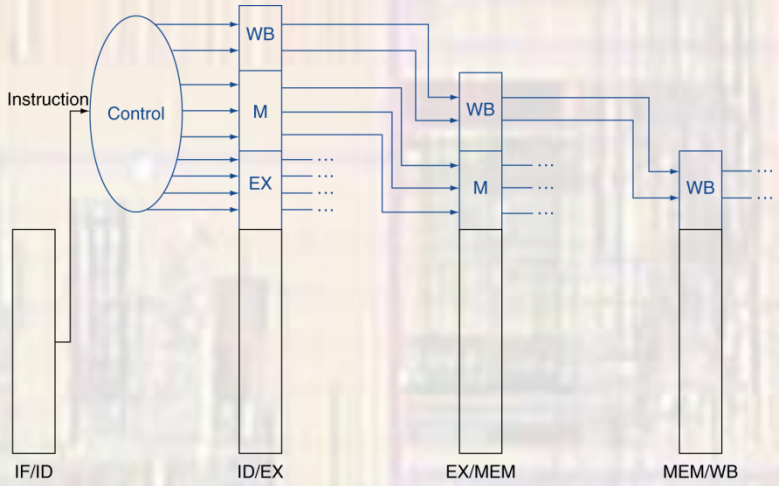
- Pipeline Control
  - Many more control signals than we show
  - IF – all control lines operate the same way for all instructions
    - PC is read
    - Program Memory is read
    - PC is updated
  - ID - all control lines operate the same way for all instructions
    - Instruction is decoded
    - Registers are read

Pipelining

- Pipeline Control
  - EX – executes or calculates an address
    - *RegDst* – choose between 2nd or 3rd register field for WB
    - *ALUOp* – L/S, Branch, or R-type
    - *ALUSrc* – selects Read Data 2 or sign extended immediate
    - These are generated in the ID stage but used in the EX stage
      - Must pass them forward through the ID/EX register
  - MEM – R/W to memory and selects the offset branch value
    - *MemRead* , *MemWrite* – memory read / write
    - *Branch* – combined with "zero" selects the offset branch to feed back to the PC
    - These are generated in the ID stage but used in the MEM stage
      - Must pass them forward through the ID/EX register and the EX/MEM register

Pipelining

- Pipeline Control
  - WB – chooses what to write back
    - *RegWrite* – enables a write to the register file
    - *MemtoReg* – choose between ALU output or memory output to feed back to the register file
  - These are generated in the ID stage but used in the MEM stage
    - Must pass them forward through the ID/EX, EX/MEM and MEM/WB registers



Pipelining

- Pipeline Control