CS/ECE 552: Pipelining

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mark Hill, Mikko Lipasti, David Wood, Guri Sohi, John Shen and Jim Smith

Pipelining

- Motivation
- Datapath
- Control

Single Cycle Performance

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) **(CPI)** **(cycle time)**

- Single cycle implementation
  - CPI = 1
  - Cycle = imem + RFrd + ALU + dmem + RFwr + muxes + control
  - E.g. 500+250+500+500+250+0+0 = 2000ps
  - Time/program = IPP x 1 CPI x 2ns = IPP x 2ns

Pipeline Stages

THE UNIVERSITY of WISCONSIN MADISON

Stage	Description	Sample Actions
IF	Fetch	Instr=MEM[PC] NextPC=PC+4
ID	Decode	A=RF(IR[25:21]) B=RF(IR[20:16]) Target={PC+4[31:28], SE(Instr[15:0] << 2)}

EX	Execute	ALUout = A + SE(Instr[15:0]) # lw/sw
		ALUout = A op B # rrr (R-format)
		if (A==B) NextPC = Target # beq

Mem	Memory	MEM[ALUout] = B # sw R_t = MEM[ALUout] #lw

WB	Writeback	Reg(IR[20:16]) = R_t # lw
		Reg(IR[20:16]) = ALUOut # rrr (R-format)

Multicycle Performance

- Multicycle implementation:

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
i	F	D	X	M	W								
i+1						F	D	X	M	W			
i+2											F	D	X
i+3													
i+4													

Multicycle Performance

- Multicycle implementation
  - CPI = 5 // 5 for each instruction
  - Cycle = max(memory, RF, ALU, mux, control)
  - = max(300,200,300) = 300ps
  - Time/prog = IPP x 5 x 500 = IPP x 300ps = IPP x 1.5ns
- Would like:
  - CPI = 1 + overhead from hazards (later)
  - Cycle = 300ps + overhead
  - In practice, ~3x improvement

Latency vs. Throughput

- Instruction latency = 5 cycles
- Instruction throughput = 1/5 instr/cycle
- CPI = 5 cycles per instruction
- Instead
  - Pipelining: process instructions like a lunch buffet
  - ALL microprocessors use it
    - E.g. Intel Core i7, AMD Ryzen, ARM A10

Instruction Throughput

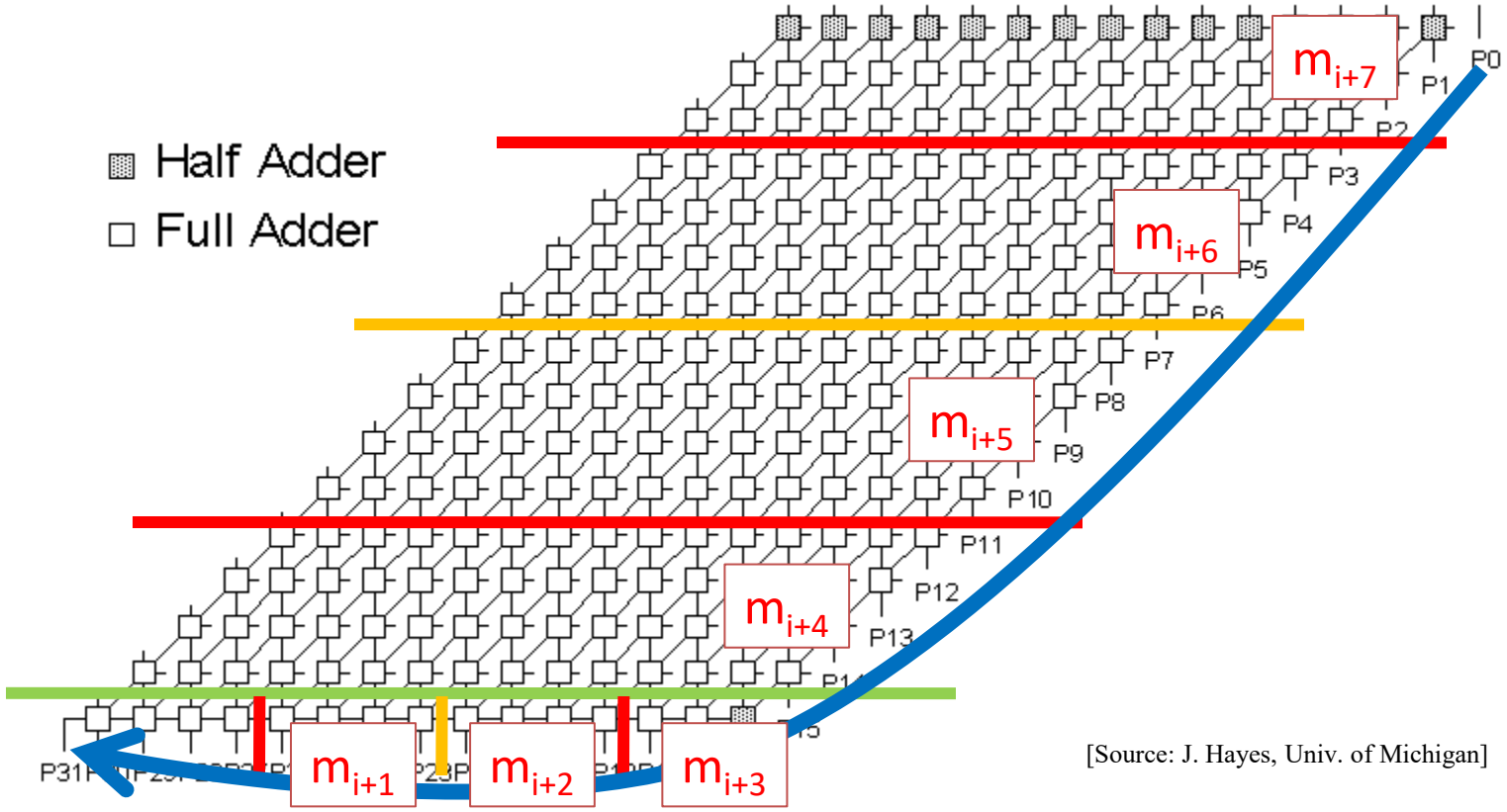
- Instruction Latency = 5 cycles (same)
- Instruction throughput = 1 instr/cycle
- CPI = 1 cycle per instruction!

Ideal Pipelining

Bandwidth increases linearly with pipeline depth

Latency increases by latch delays

Example: Integer Multiplier



[Source: J. Hayes, Univ. of Michigan]

16x16 combinational multiplier

ISCAS-85 C6288 standard benchmark

Tools: Synopsys DC/LSI Logic 110nm gflxp ASIC

Example: Integer Multiplier

Configuration	Delay	MPS (Thrpt)	Area (FF/wiring)	Area Increase
Combinational	3.52ns	284	7535 (--/1759)	
2 Stages	1.87ns	534 (1.9x)	8725 (1078/1870)	16%
4 Stages	1.17ns	855 (3.0x)	11276 (3388/2112)	50%
8 Stages	0.80ns	1250 (4.4x)	17127 (8938/2612)	127%

Pipeline efficiency

- 2-stage: nearly double throughput; marginal area cost
- 4-stage: 75% efficiency; area still reasonable
- 8-stage: 55% efficiency; area more than doubles

Tools: Synopsys DC/LSI Logic 110nm gflxp ASIC

Ideal Pipelining

Cycle: Instr:	1	2	3	4	5	6	7	8	9	10	11	12	13
i	F	D	X	M	W								
i+1		F	D	X	M	W							
i+2			F	D	X	M	W						
i+3				F	D	X	M	W					
i+4					F	D	X	M	W				

Pipelining Idealisms

- Uniform subcomputations
  - Can pipeline into stages with equal delay
- Identical computations
  - Can fill pipeline with identical work
- Independent computations
  - No relationships between work units
  - No *dependences*, hence no *pipeline hazards*
- Can we guarantee these conditions all the time?
  - No, but can get close enough to get significant speedup

CS/ECE 552: Pipelining Part 2

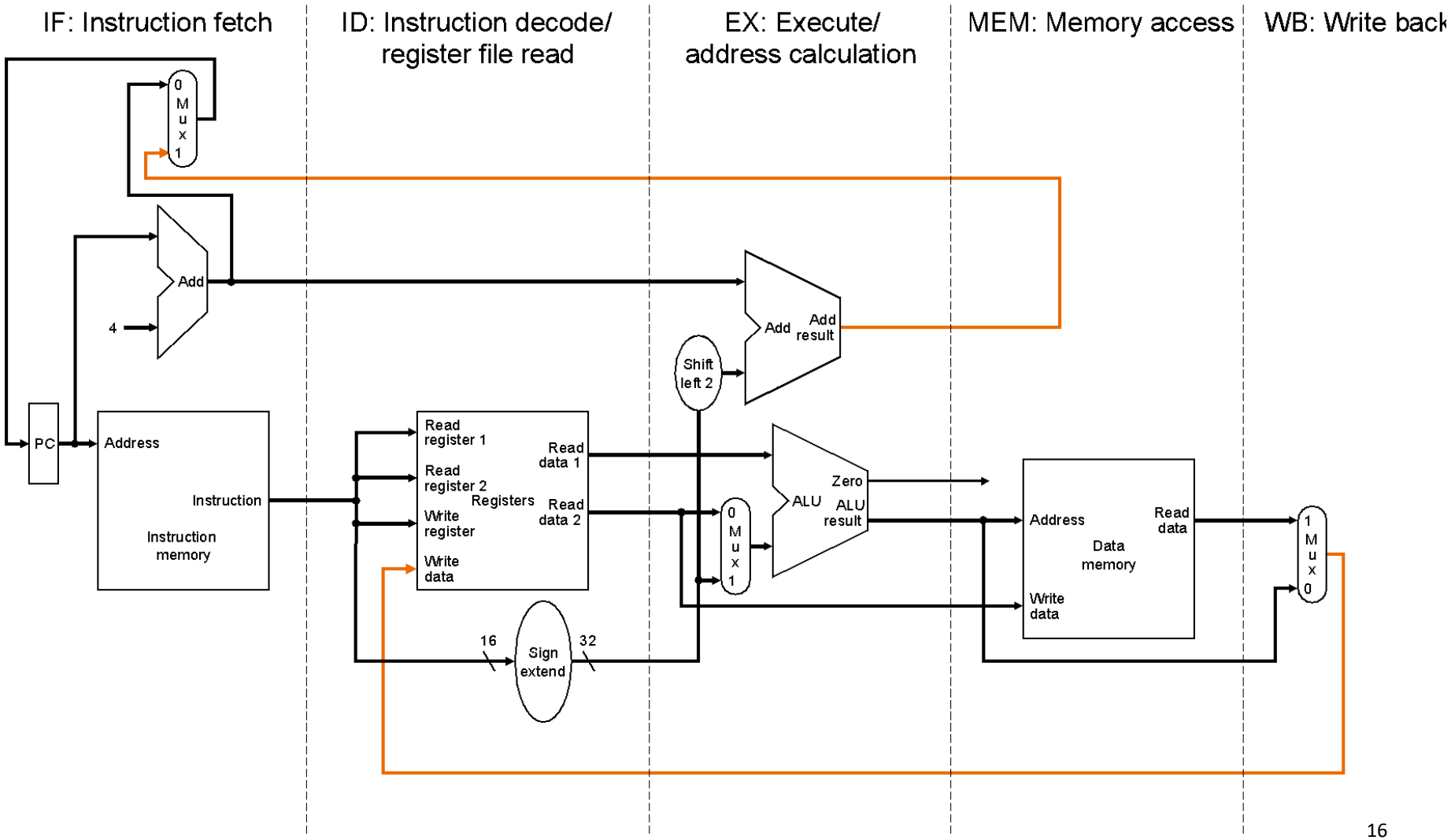
Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mark Hill, Mikko Lipasti, David Wood, Guri Sohi, John Shen and Jim Smith

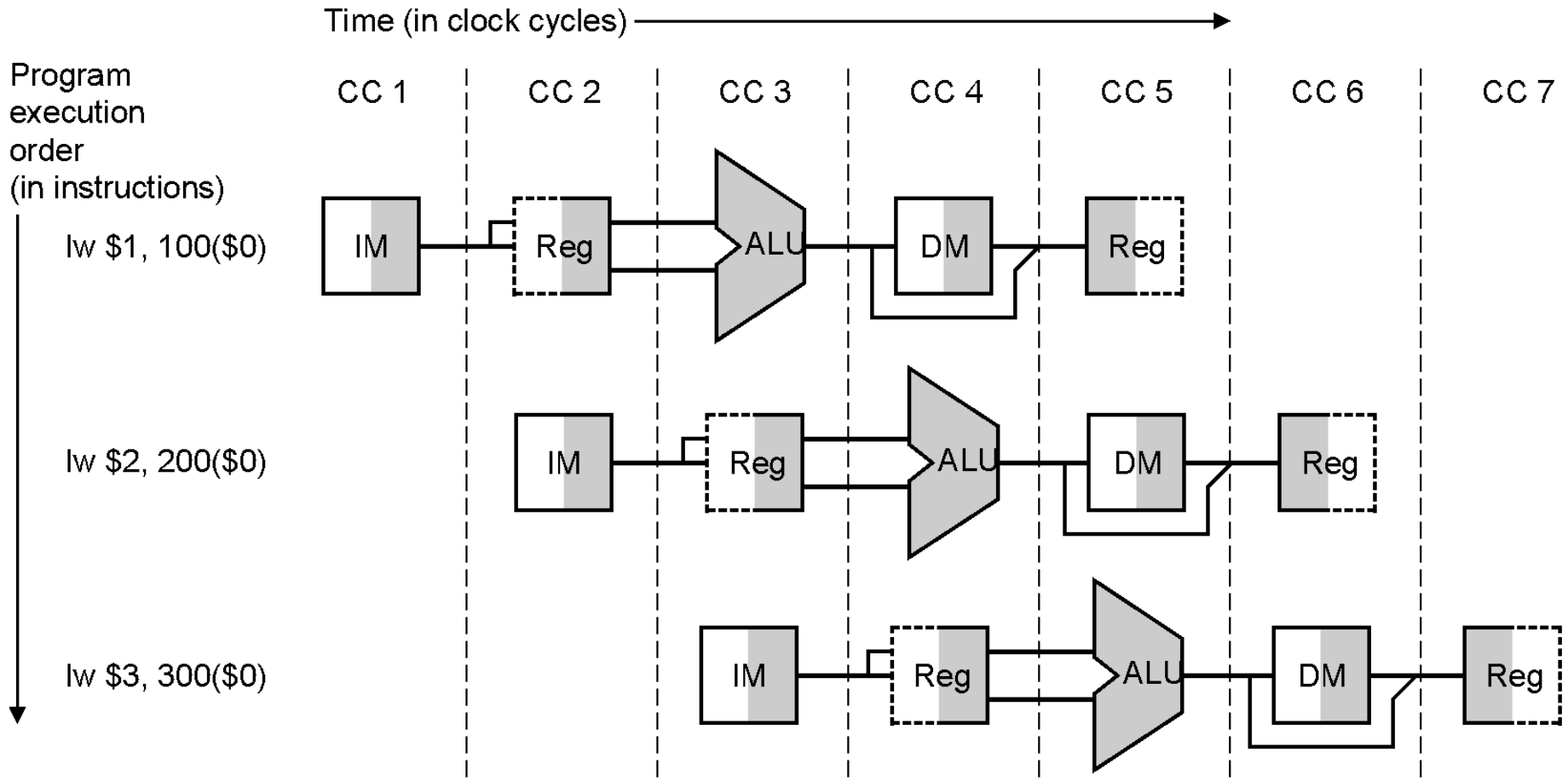
Complications

- Datapath
  - Five (or more) instructions in flight
- Control
  - Must correspond to multiple instructions
- Instructions may have
  - data and control flow *dependences*
  - I.e., units of work are not independent
    - One may have to stall and wait for another

Datapath


IF: Instruction fetch
ID: Instruction decode/ register file read
EX: Execute/ address calculation
MEM: Memory access
WB: Write back
0
M
u
x
1
Add
4
PC
Address
Instruction
Instruction memory
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
0
M
u
x
1
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
1
M
u
x
0
16
Sign extend
32

Datapath


Time (in clock cycles)
Program execution order (in instructions)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
IM
Reg
ALU
DM
Reg

Control

- Control
  - Concurrently set by 5 different instructions
  - Divide and conquer: carry instr. down the pipe
    - Smarter: only carry needed control signals from instr.
    - More on this later

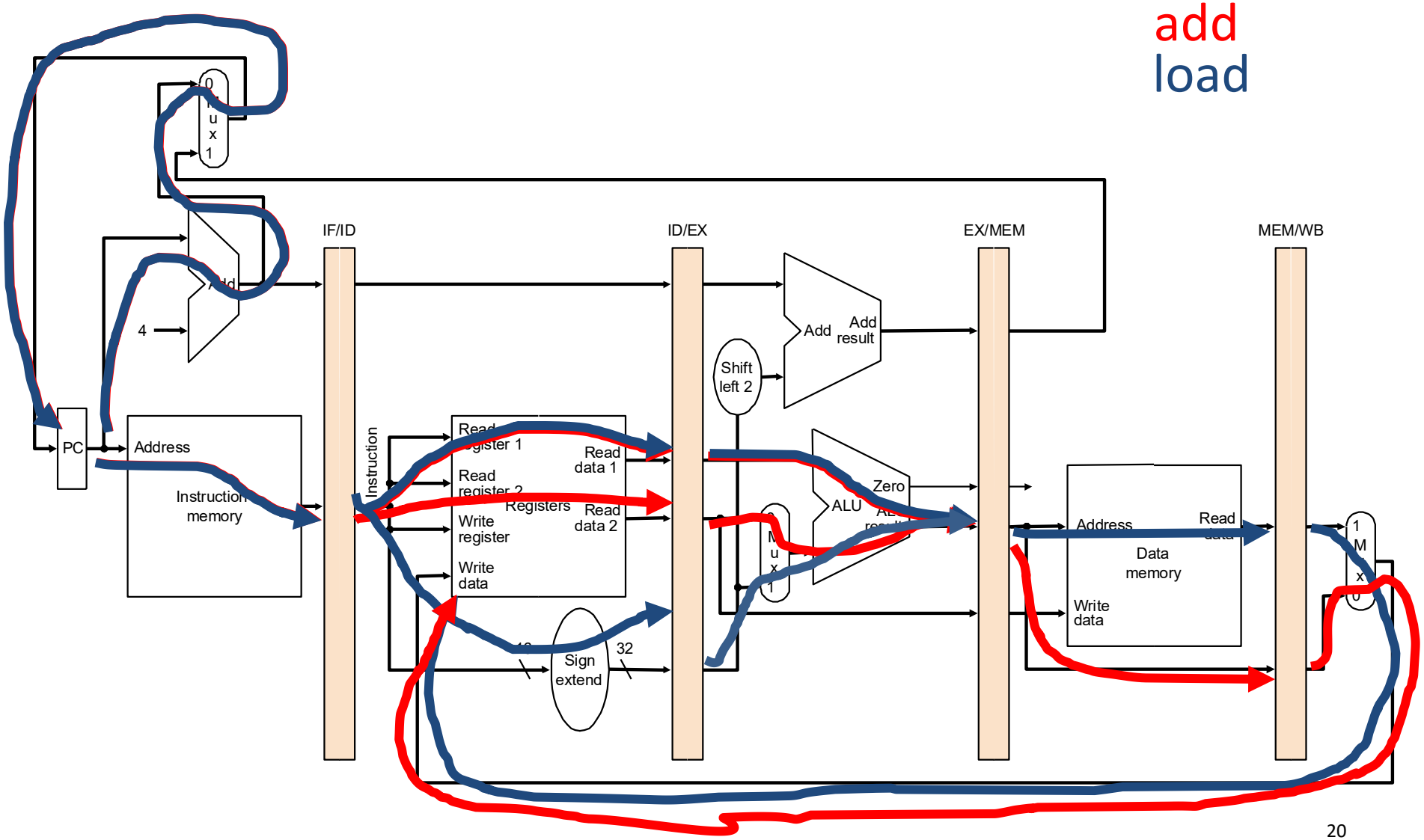
Pipelined Datapath

- Start with single-cycle datapath
- Pipelined execution
  - Assume each instruction has its own datapath
  - But each instruction uses a different part in every cycle
  - Multiplex all on to one datapath
  - Latches/flip-flops separate stages

- Ignore dependences and hazards for now
  - Data
  - control

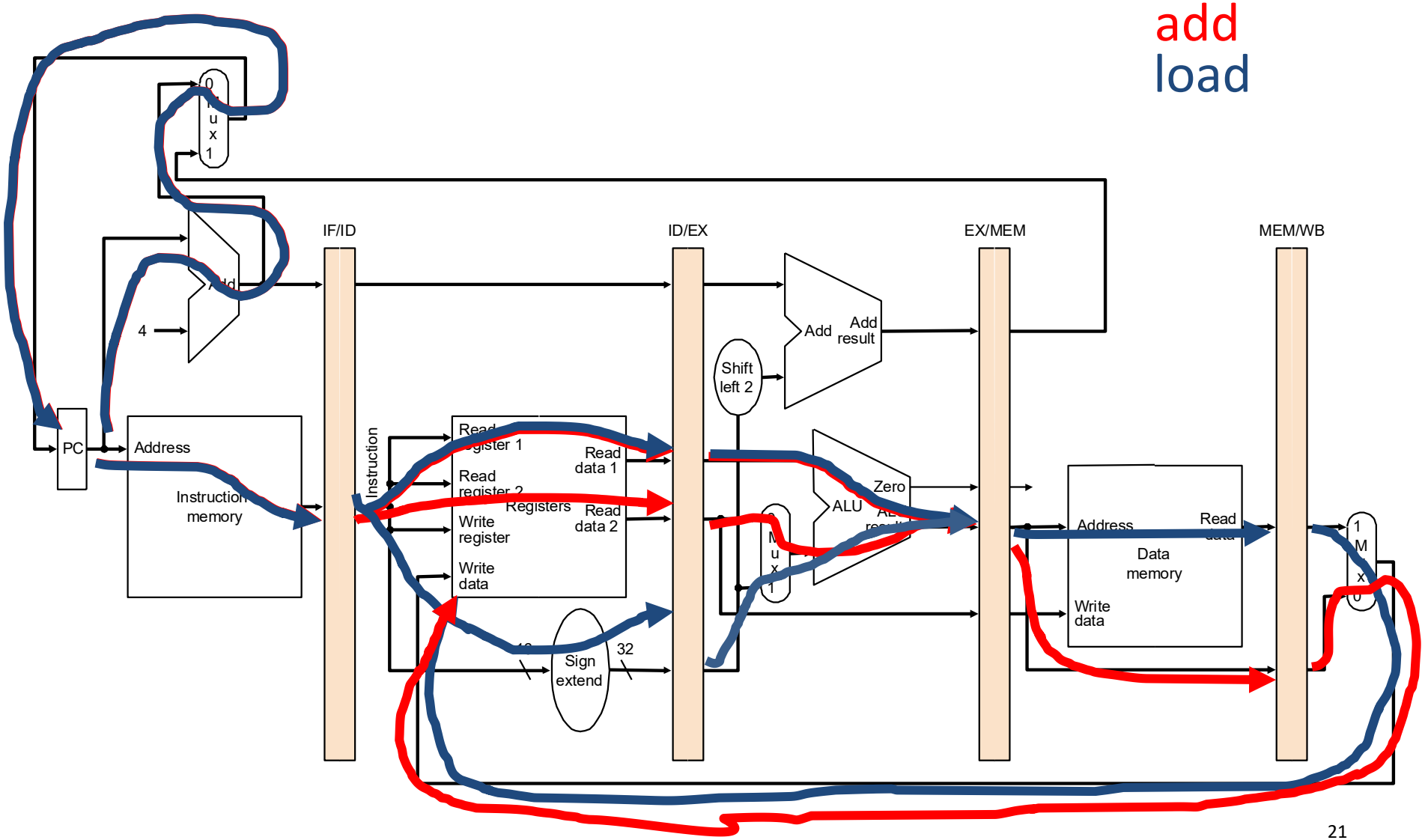
Pipelined Datapath

add
load



Pipelined Datapath

add
load



Pipelined Datapath

- Instruction flow
  - add and load shown, work through beq, st, …
  - 2 instructions in 6 cycles
  - n instructions in n+1 cycles, for large n CPI ~ 1
- Any info needed by a later stage gets passed through the pipeline
  - E.g., store value through EX for a store

CS/ECE 552: Pipelining Part 3

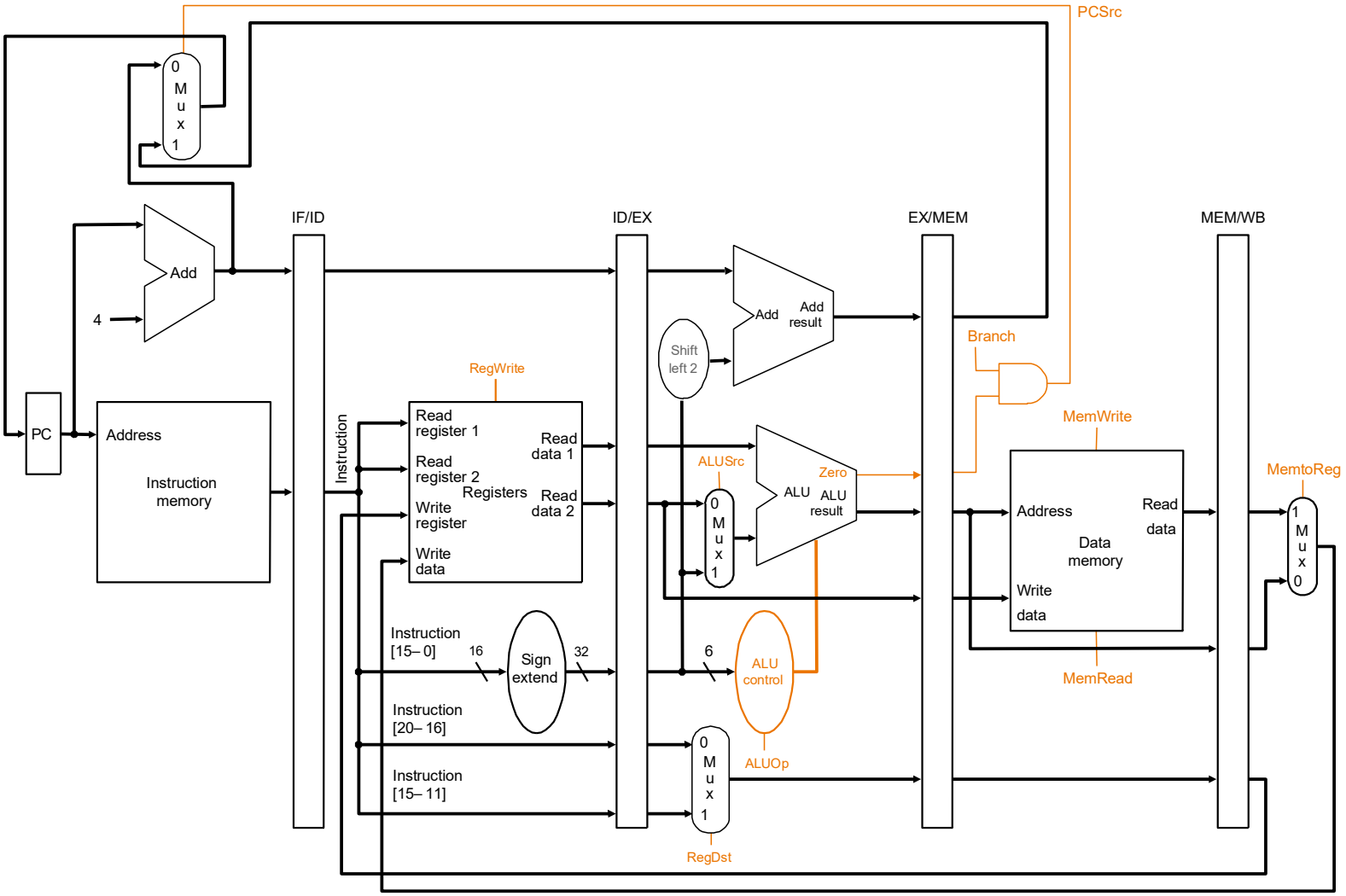
Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mark Hill, Mikko Lipasti, David Wood, Guri Sohi, John Shen and Jim Smith

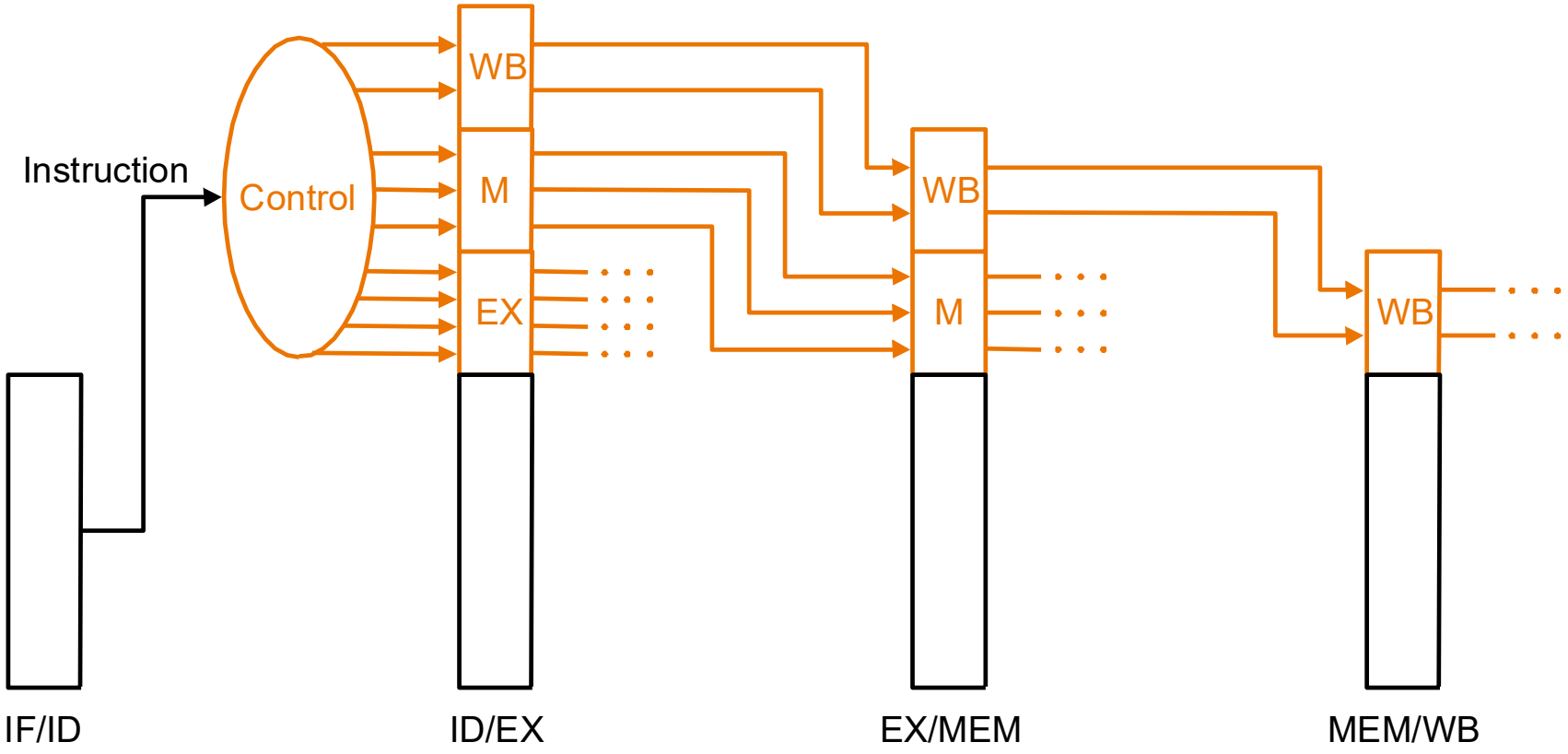
Pipelined Control

- IF and ID
  - None in IF
  - Implicit in ID
    - Just decoded instruction from IF/ID pipeline register
- EX
  - ALUop, ALUsrc, RegDst
- MEM
  - Branch, MemRead, MemWrite
- WB
  - MemtoReg, RegWrite

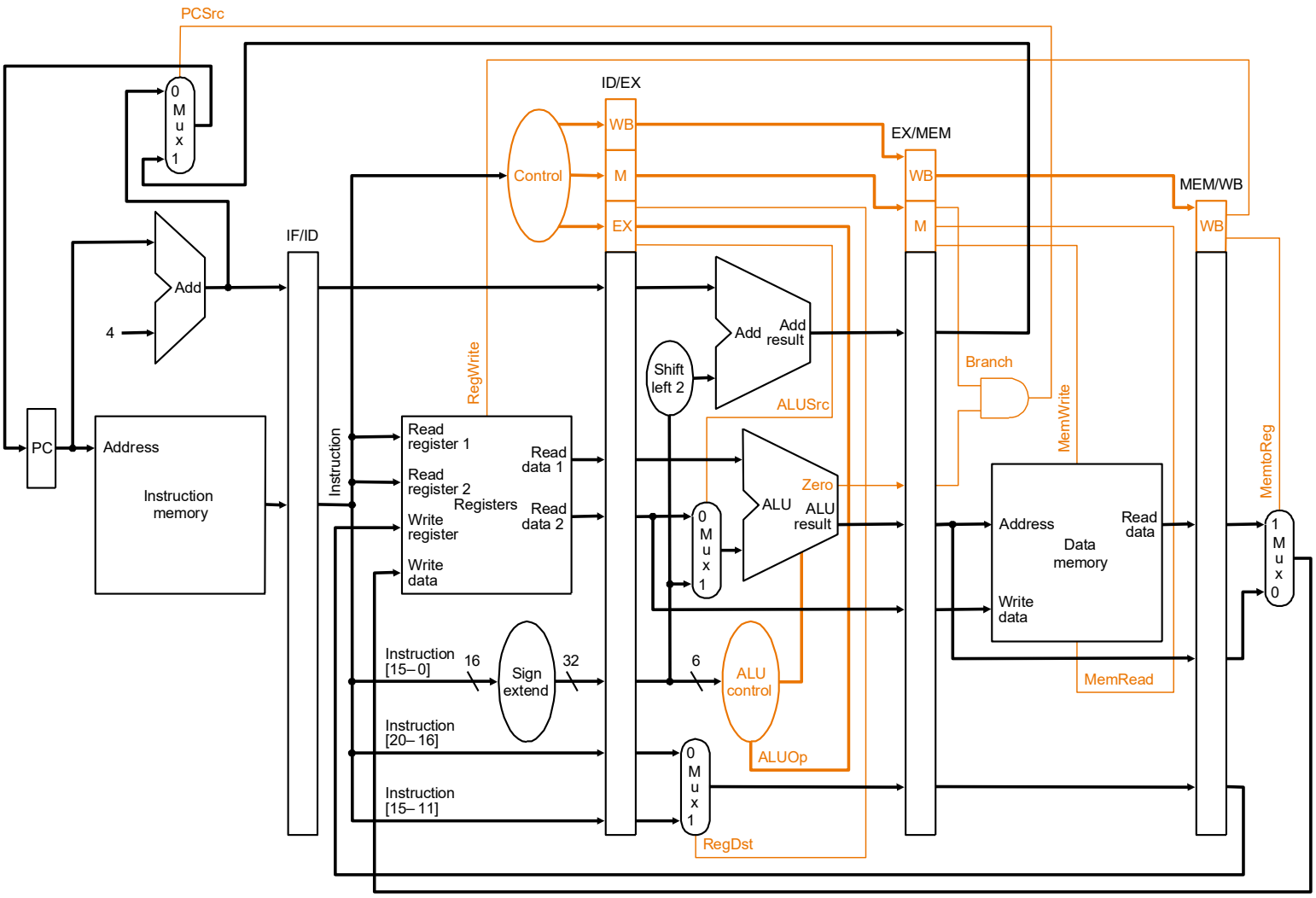
Datapath Control Signals



Pipelined Control



All Together



Pipelined Control

- Each stage controlled by a different instruction
- Decode instruction in ID, pass its control signals through pipeline
- Control sequencing embedded in pipeline
  - No explicit FSM
  - Instead, distributed FSM (harder to verify)

Summary

- Motivation
- Datapath
- Control
- Next
  - Program dependences
  - Pipeline hazards