Lecture 19
Introduction to Pipelining

Example: We have to build x cars...
...Each car takes 6 steps to build...

Build the frame
(~1 hour)

Build the body
(~1.25 hours)

Install interior
(~1.25 hours)

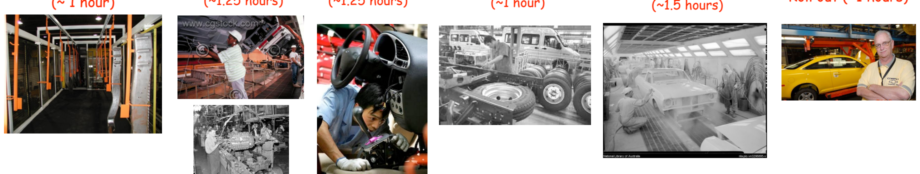
Put on axles, wheels
(~1 hour)

Paint
(~1.5 hours)

Roll out
(~1 hours)

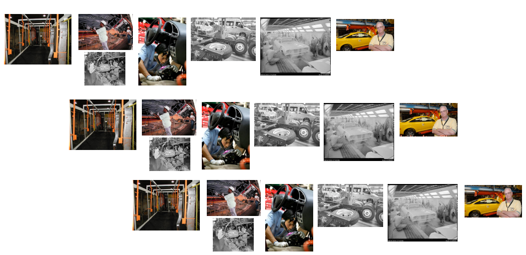
Sequential Car Building... (a lot like multi-cycle)

Build the frame (~ 1 hour) | Build the body (~1.25 hours) | Install interior (~1.25 hours) | Put on axles, wheels (~1 hour) | Paint (~1.5 hours) | Roll out (~1 hours)



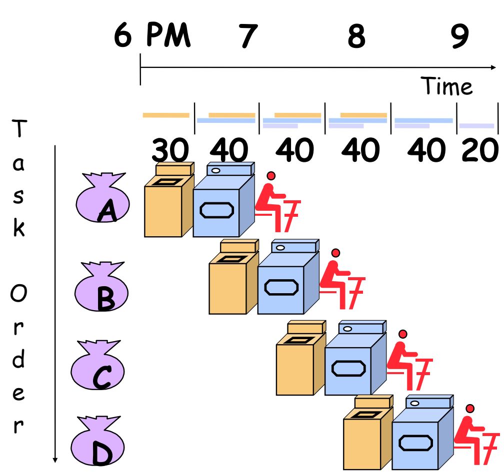
Total time: 7 Hours.
(~1 hour/stage)

Pipelined Car Building...



1 car done ~ every 1.5 hours
(like multi-cycle, limited by time of the longest stage)

Pipelining Lessons (laundry example)



- Multiple tasks operating simultaneously
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Also, need time to "fill" and "drain" the pipeline.

Pipelining: Some terms

- If you're doing laundry or implementing a μP, each stage where something is done called a pipe stage
  - In laundry example, washer, dryer, and folding table are pipe stages; clothes enter at one end, exit other
  - In a μP, instructions enter at one end and have been executed when they leave
- Throughput is how often stuff comes out of a pipeline

On the board…

- The "math" behind pipelining…

More technical detail

- If times for all S stages are equal to T:
  - Time for one initiation to complete still ST
  - Time between 2 initiates = T not ST
  - Initiations per second = 1/T
- Pipelining: Overlap multiple executions of same sequence
  - Improves THROUGHPUT, not the time to perform a single operation

More technical detail

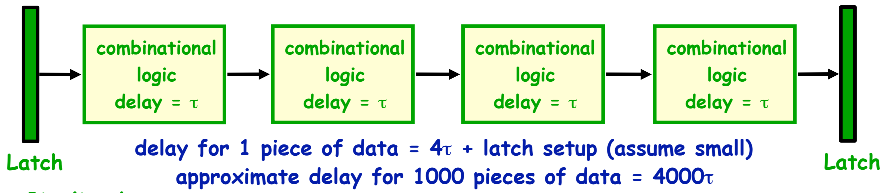
- Book's approach to draw pipeline timing diagrams…
  - Time runs left-to-right, in units of stage time
  - Each "row" below corresponds to distinct initiation
  - Boundary b/t 2 column entries: pipeline register
    - (i.e. hamper)
  - Look at columns to see what stage is doing what

0	1	2	3	4	5	6
Wash 1	Dry 1	Fold 1	Pack 1			
	Wash 2	Dry 2	Fold 2	Pack 2		
		Wash 3	Dry 3	Fold 3	Pack 3	
			Wash 4	Dry 4	Fold 4	Pack 4
				Wash 5	Dry 5	Fold 5
					Wash 6	Dry 6

Time for N initiations to complete: NT + (S-1)T
Throughput: Time per initiation = T + (S-1)T/N → T!

How much (ideal) speedup?

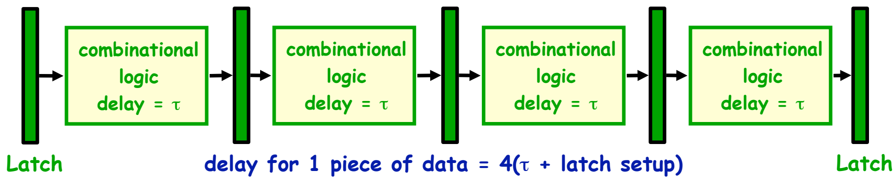
Unpipelined



delay for 1 piece of data = 4τ + latch setup (assume small)

approximate delay for 1000 pieces of data = 4000τ

Pipelined



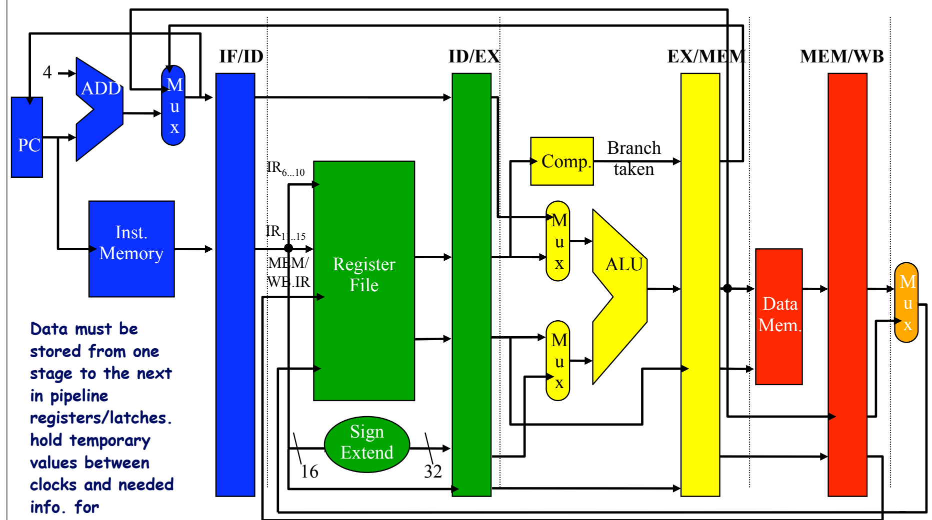
delay for 1 piece of data = 4(τ + latch setup)

approximate delay for 1000 pieces of data = 3τ + 1000τ

speedup for 1000 pieces of data = $\frac{4000}{1003}$ = ~ 4

Ideal speedup = # of pipeline stages

The "new look" dataflow

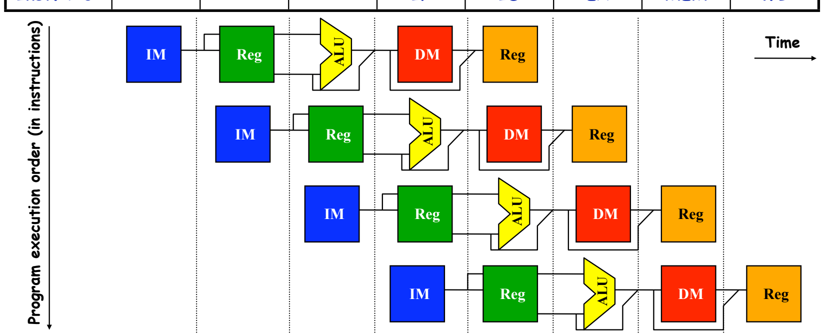


Data must be stored from one stage to the next in pipeline registers/latches. hold temporary values between clocks and needed info. for execution.

Another way to look at it…

Clock Number

Inst. #	1	2	3	4	5	6	7	8
Inst. i	IF	ID	EX	MEM	WB			
Inst. i+1		IF	ID	EX	MEM	WB		
Inst. i+2			IF	ID	EX	MEM	WB	
Inst. i+3				IF	ID	EX	MEM	WB



So, what about the details?

- In each cycle, new instruction fetched and begins 5 cycle execution
- In perfect world (pipeline) performance improved 5 times over!
- Now, let's talk about overhead…
  - (i.e. what else do we have to worry about?)
    - Must know what's going on in every cycle of machine
    - What if 2 instructions need same resource at same time?
      - (LOTS more on this later)
      - Separate instruction/data memories, multiple register ports, etc. help avoid this

Limits, limits, limits…

- So, now that the ideal stuff is out of the way, let's look at how a pipeline REALLY works…
- Pipelines are slowed b/c of:
  - Pipeline latency
  - Imbalance of pipeline stages
    - (Think: A chain is only as strong as its weakest link)
    - Well, a pipeline is only as fast as its slowest stage
  - Pipeline overhead (from where?)
    - Register delay from pipe stage latches

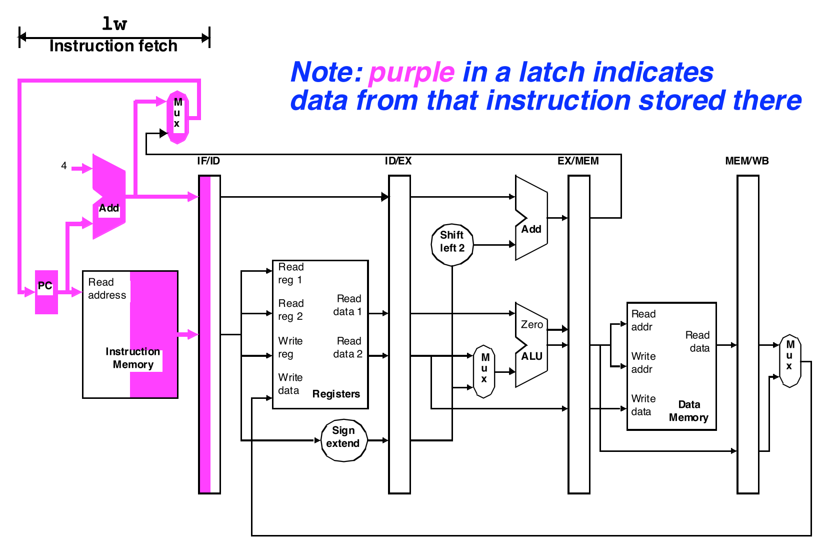
Let's look at some examples:

- Specifically:
  - (1 instruction sequence -- with a problem)
  - (2 instruction sequence)

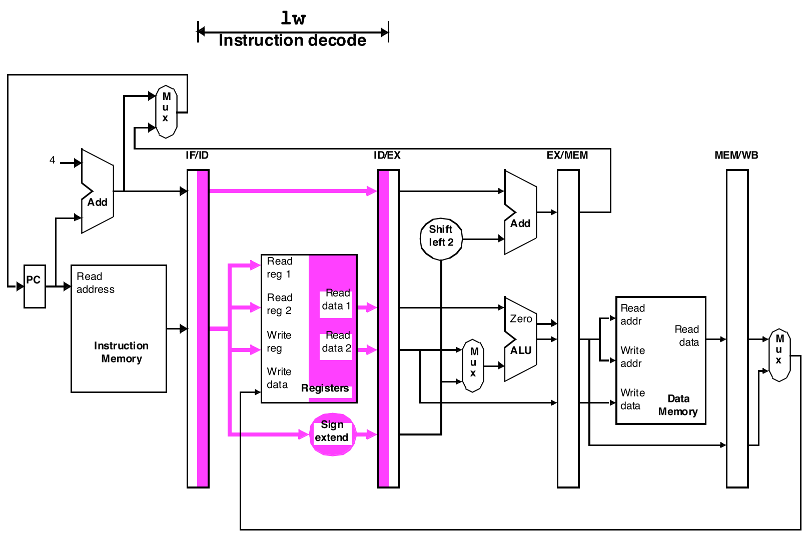
Executing Instructions in Pipelined Datapath

- Following charts describe 3 scenarios:
  - Processing of load word (lw) instruction
    - Bug included in design (make SURE you understand the bug)
  - Processing of lw
    - Bug corrected (make SURE you understand the fix)
  - Processing of lw followed in pipeline by sub
    - (Sets the stage for discussion of HAZARDS and inter-instruction dependencies)

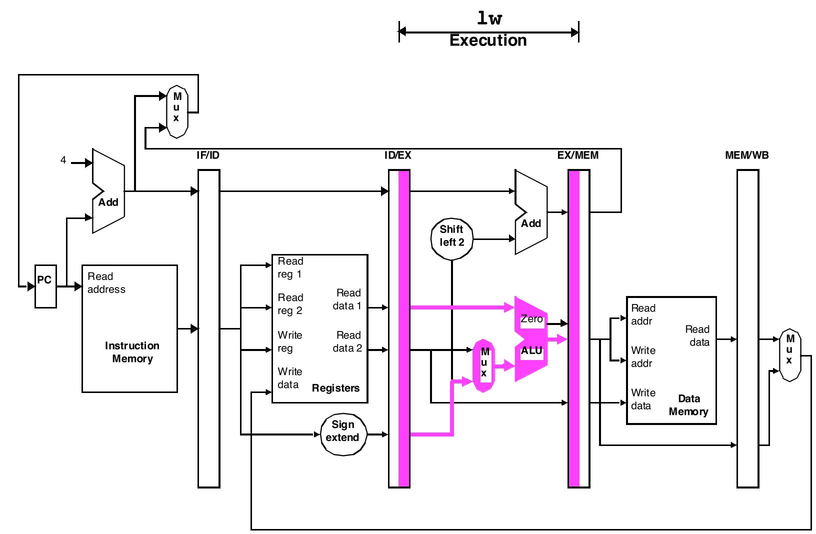
Load word: Cycle 1



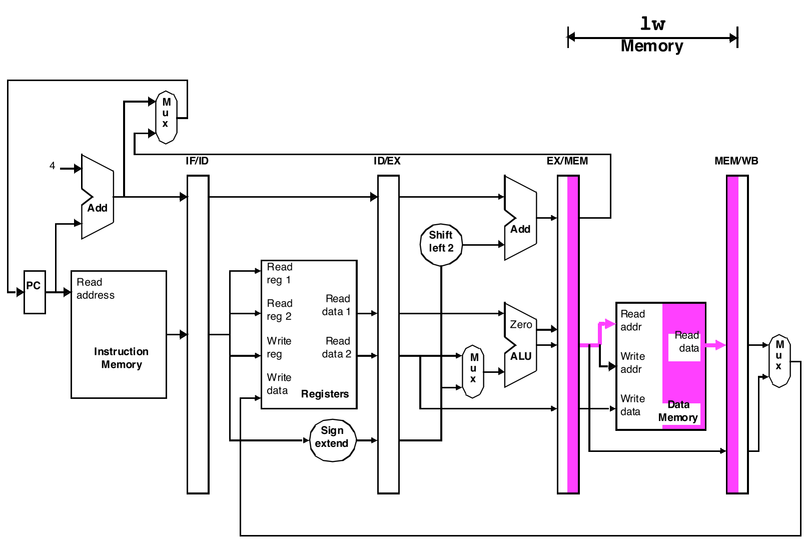
Load Word: Cycle 2



Load Word: Cycle 3



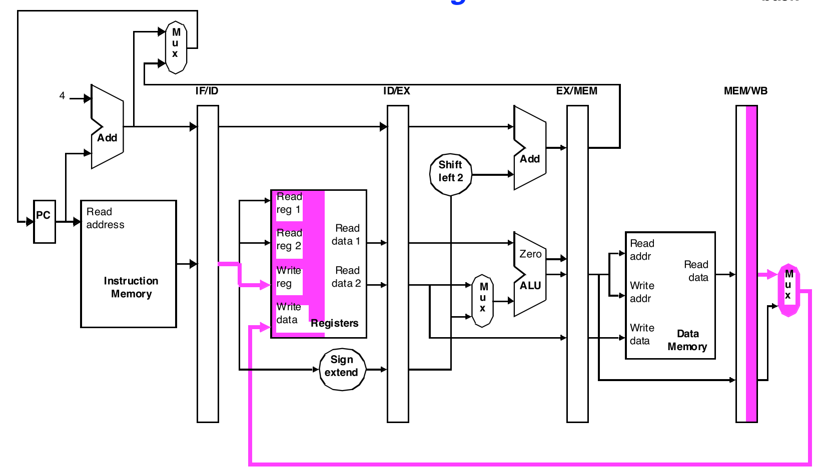
Load Word: Cycle 4



Load Word: Cycle 5

lw
Write back

Where's the bug?

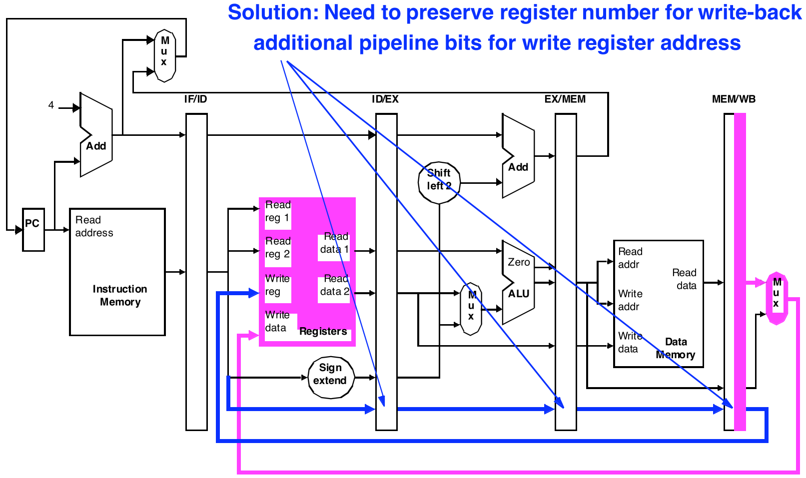


Load Word: Fixed Bug

Bug: source for Write Reg is invalid

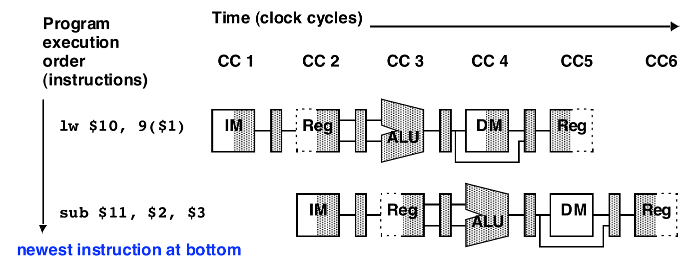
Solution: Need to preserve register number for write-back

additional pipeline bits for write register address

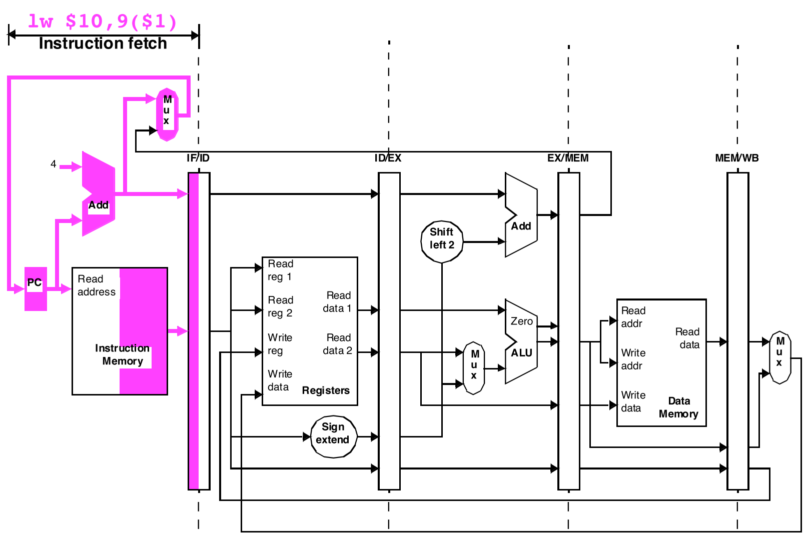


A 2 instruction sequence

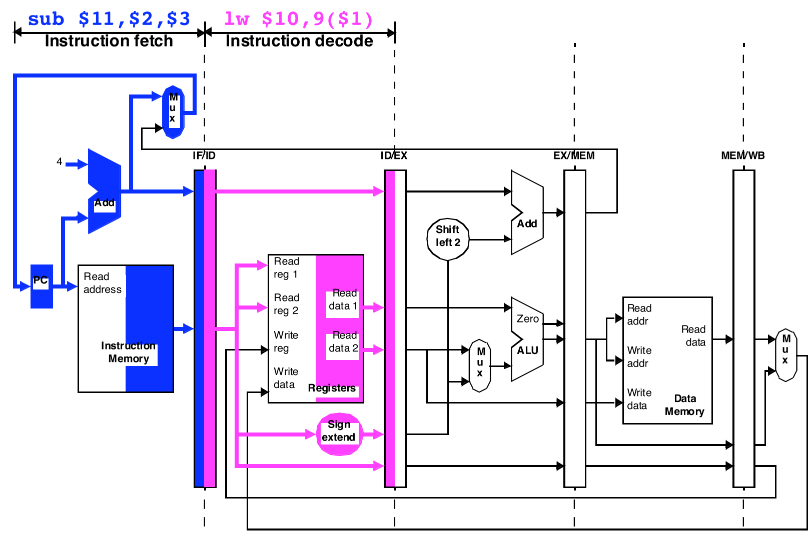
- Examine multiple-cycle & single-cycle diagrams for a sequence of 2 independent instructions
  - (i.e. no common registers b/t them)
    - lw $10, 9($1)
    - sub $11, $2, $3



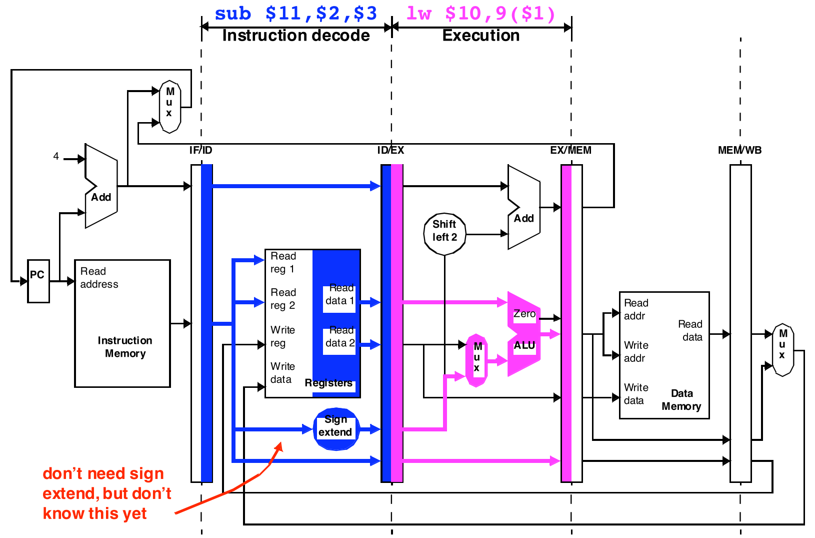
Single-cycle diagrams: cycle 1



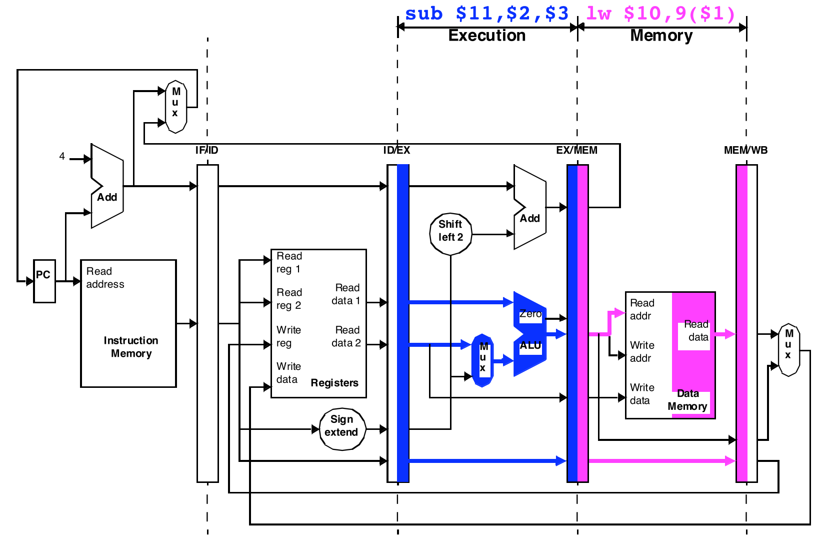
Single-cycle diagrams: cycle 2



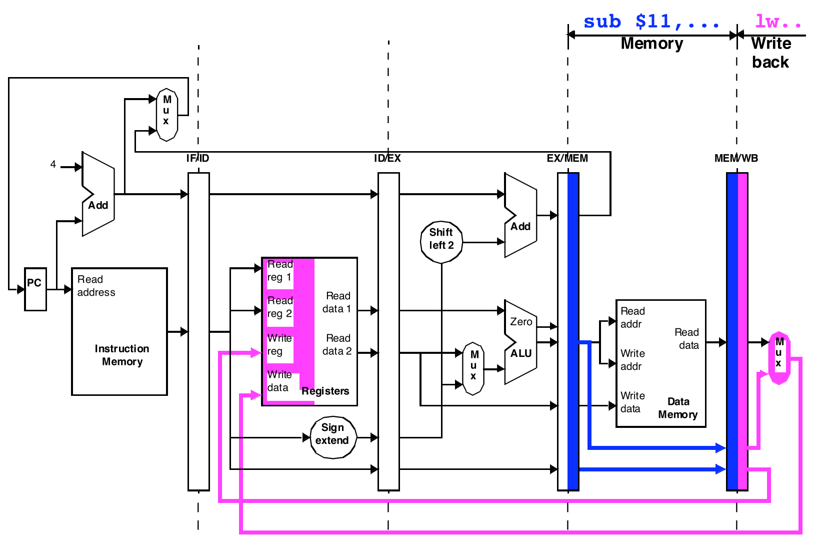
Single-cycle diagrams: cycle 3



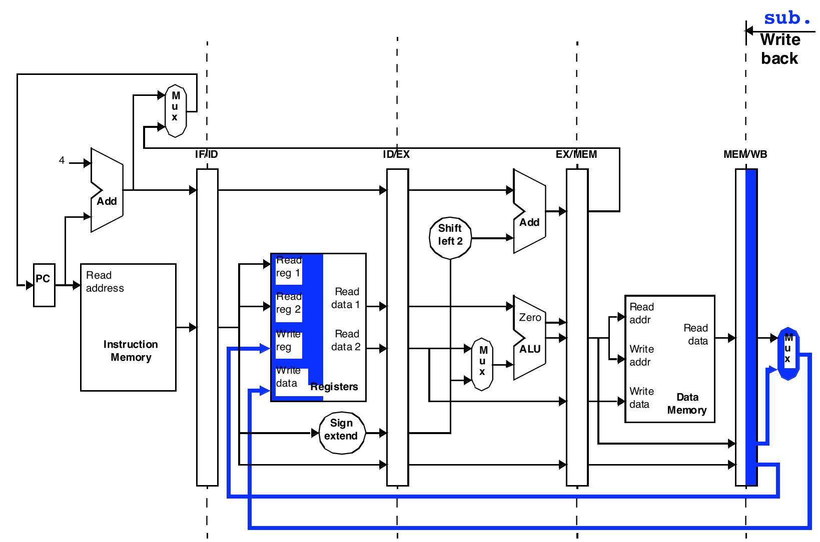
Single-cycle diagrams: cycle 4



Single-cycle diagrams: cycle 5



Single-cycle diagrams: cycle 6



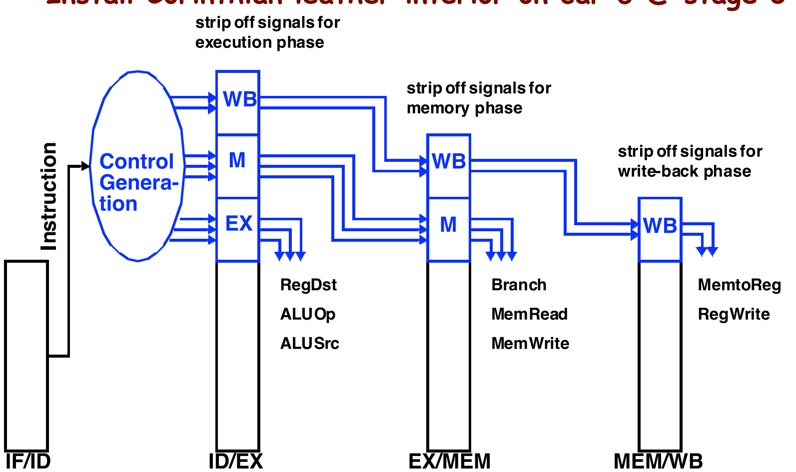
What about control signals?

Questions about control signals

- Following discussion relevant to a single instruction
- Q: Are all control signals active at the same time?
- A: ?
- Q: Can we generate all these signals at the same time?
- A: ?

Passing control w/pipe registers

- Analogy: send instruction with car on assembly line
  - "Install Corinthian leather interior on car 6 @ stage 3"



Pipelined datapath w/control signals