Cache Operation

Exercise to explore different organizations

Teams Each Take One Organization

- Same trace for all — we have already seen 2-way set associative
- Direct mapped, 4-way set associative, fully associative
- Direct plus vitim buffer, 4-way plus victim buffer
- Note type of hit/miss
  - H = hit, C = compulsory miss, X = conflict miss, V = victim hit
- Reminder that conflict miss happens when a line was in cache previously

Results

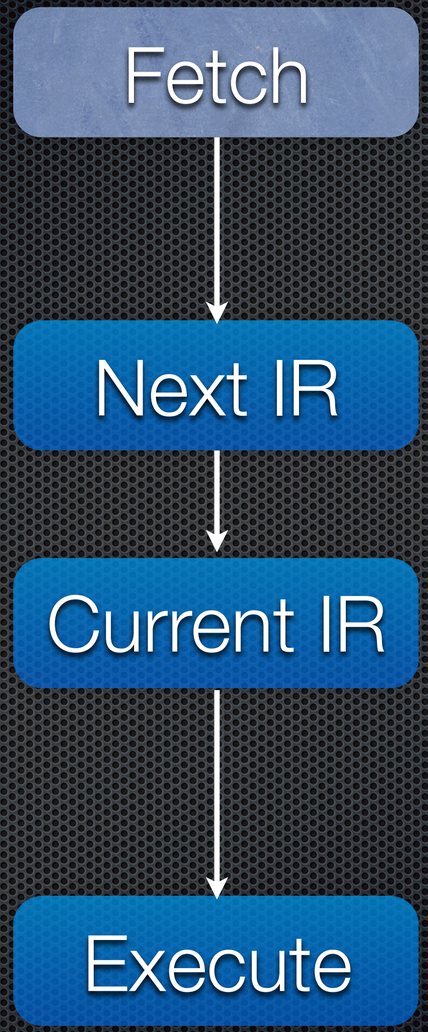
	Hit	Compulsory	Conflict	Victim Hit	Total Hits
Direct	16	22	2	-	16
2-way SA	16	22	2	-	16
4-way SA	16	22	2	-	16
Fully Assoc	18	22	0	-	18
Direct + VB	16	22	1	1	17
4-way + VB	17	22	0	1	18

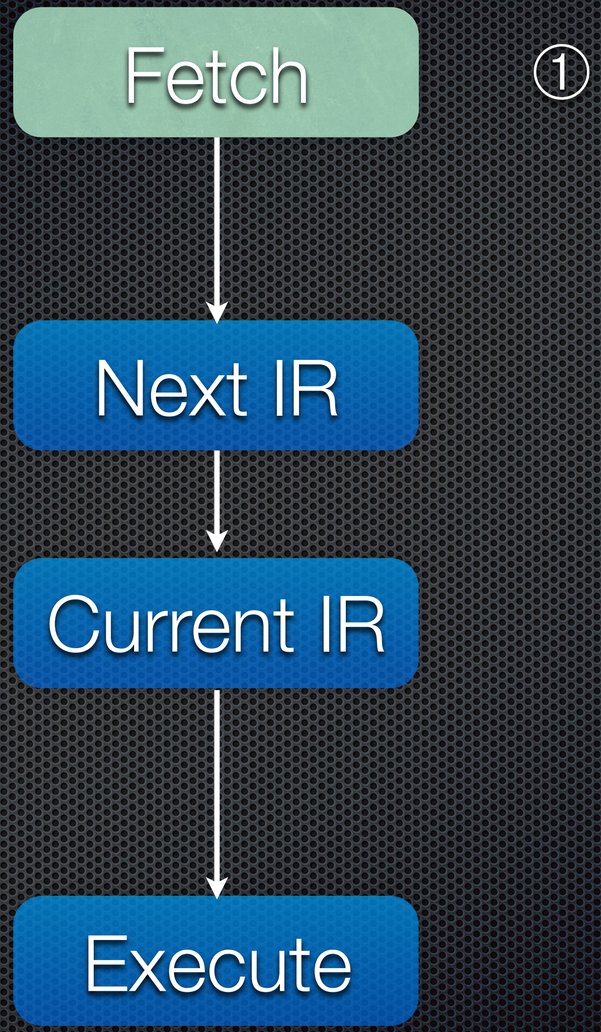
Pipelined Execution

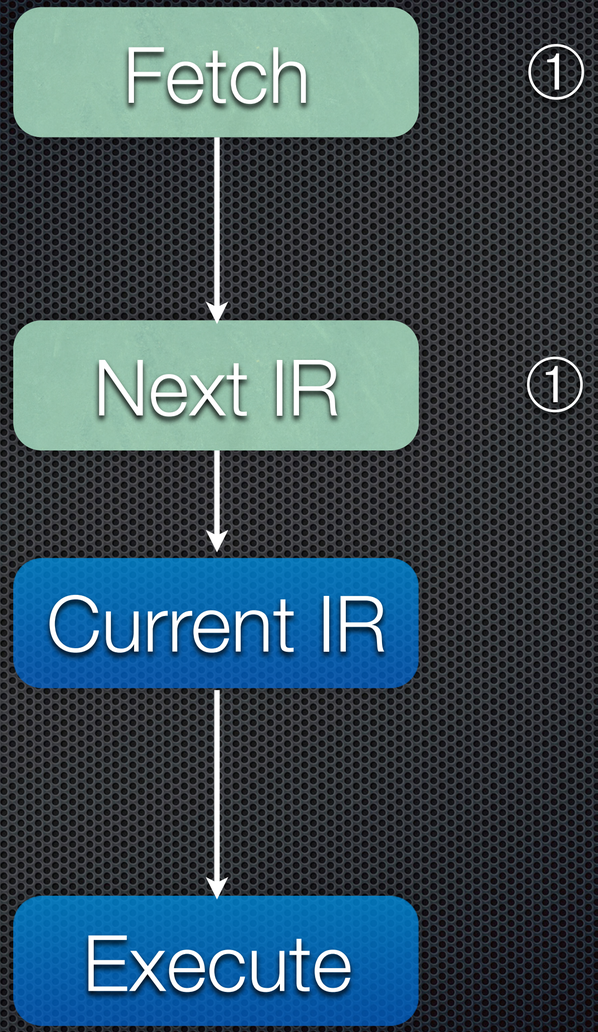
A Basic Form of Instruction Level Parallelism

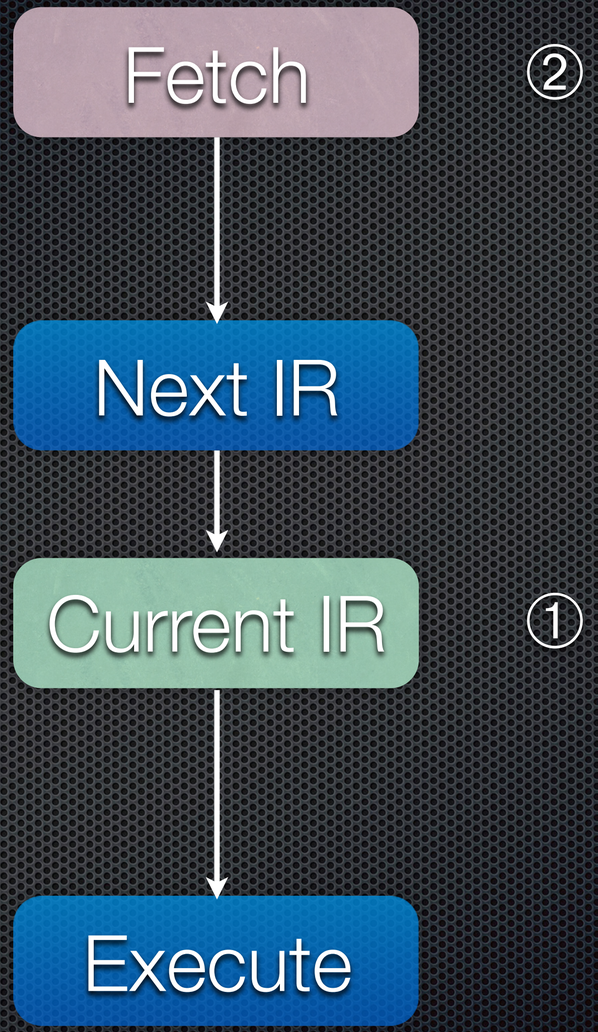
Basic Idea

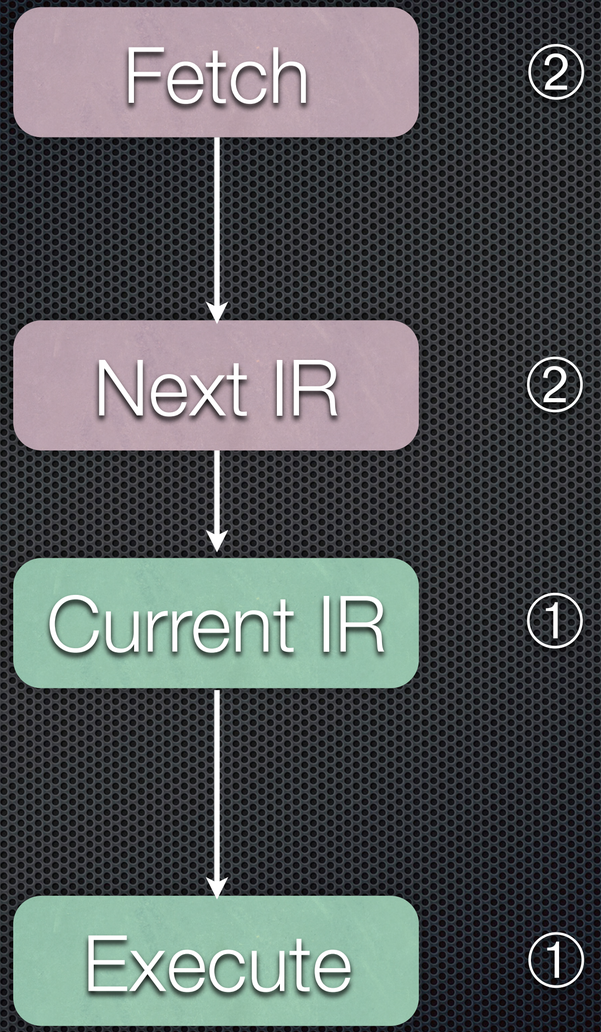
- Every instruction goes through fetch-execute
- We can fetch the next instruction while executing the current one

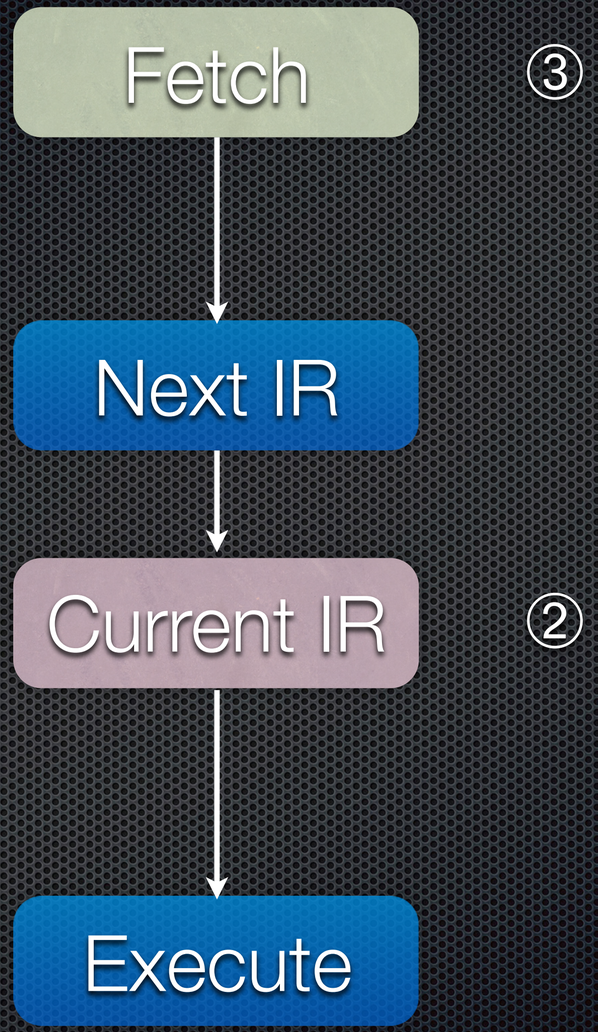

Fetch
Next IR
Current IR
Execute

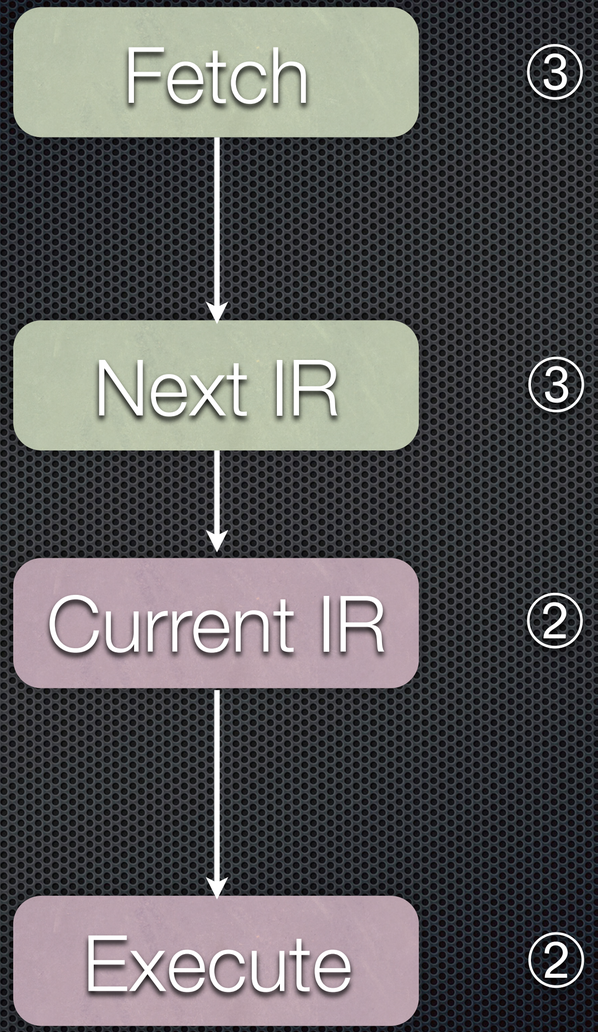

Fetch
①
Next IR
Current IR
Execute

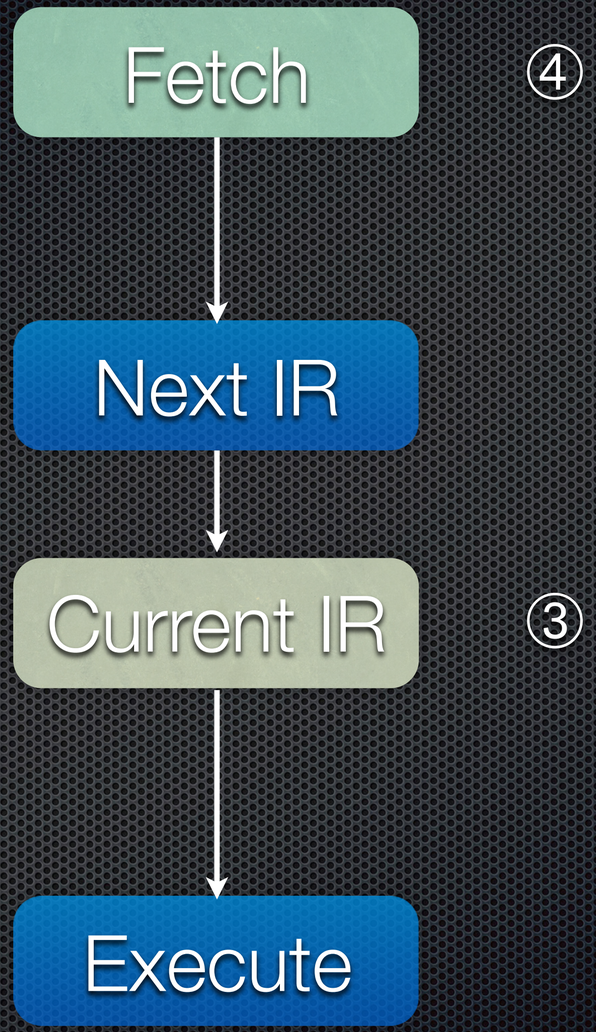

Fetch
①
Next IR
①
Current IR
Execute

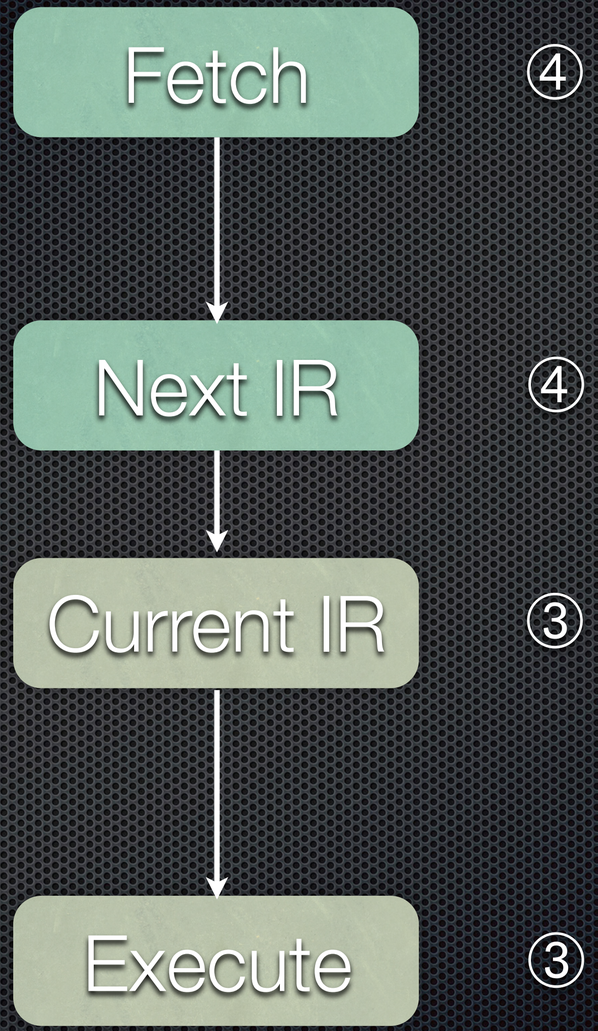

Fetch
②
Next IR
Current IR
①
Execute

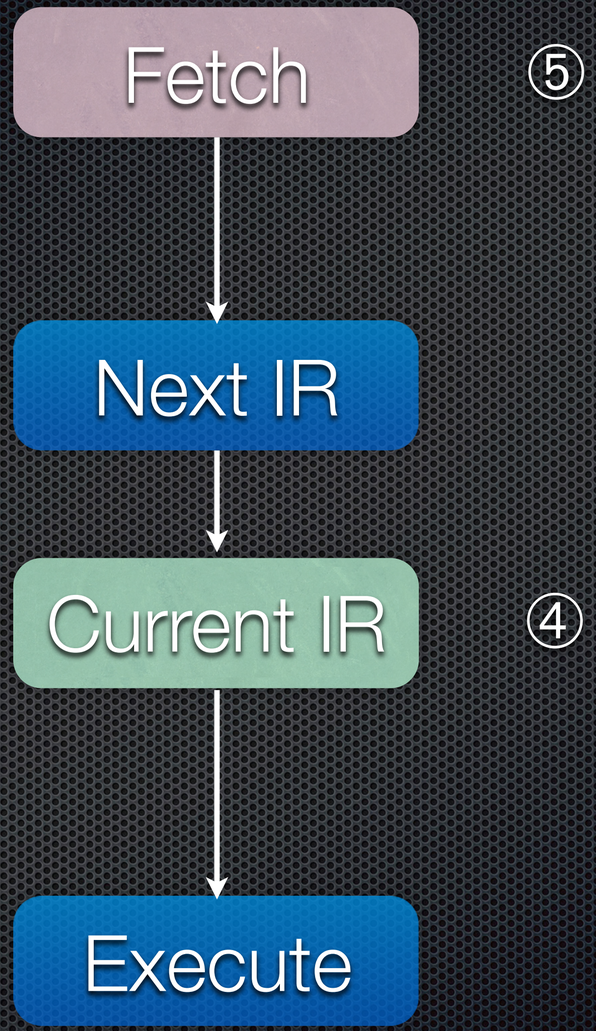

Fetch
②
Next IR
②
Current IR
①
Execute
①

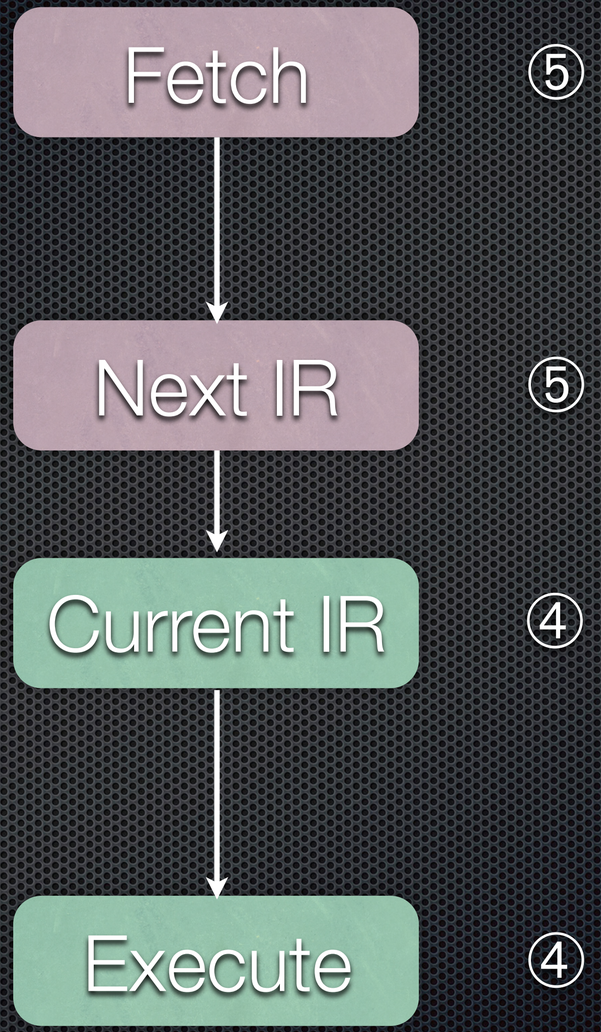

Fetch
③
Next IR
Current IR
②
Execute

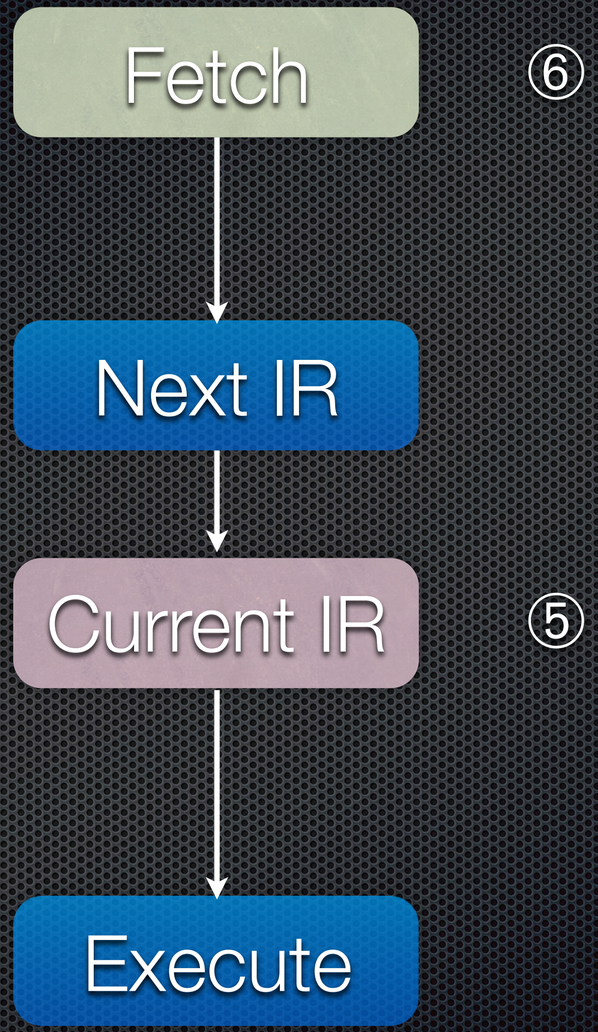

Fetch
③
Next IR
③
Current IR
②
Execute
②

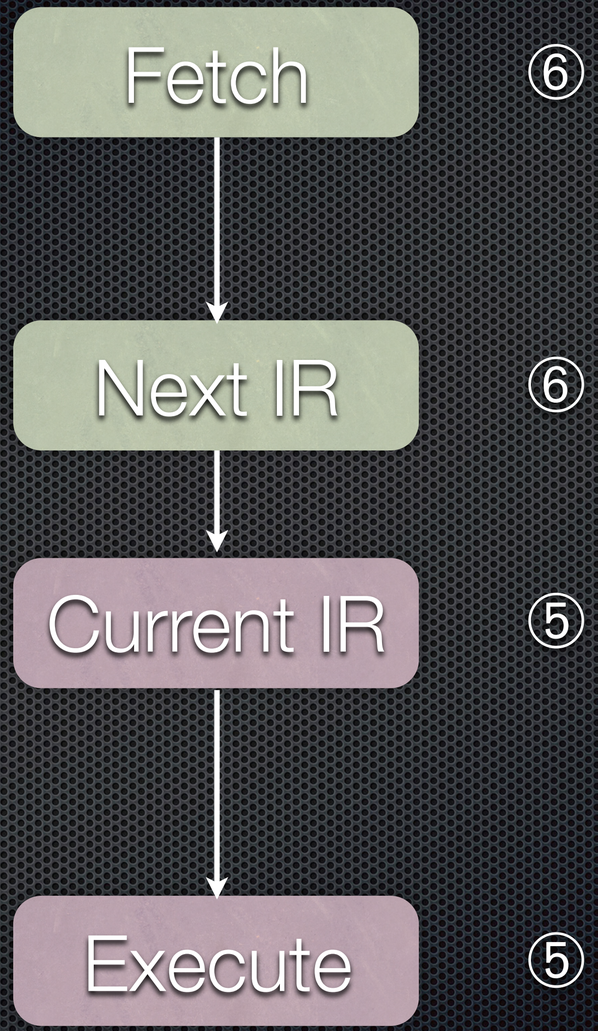

Fetch
④
Next IR
Current IR
③
Execute


Fetch
④
Next IR
④
Current IR
③
Execute
③


Fetch
⑤
Next IR
Current IR
④
Execute


Fetch
⑤
Next IR
⑤
Current IR
④
Execute
④


Fetch
⑥
Next IR
Current IR
⑤
Execute


Fetch
⑥
Next IR
⑥
Current IR
⑤
Execute
⑤

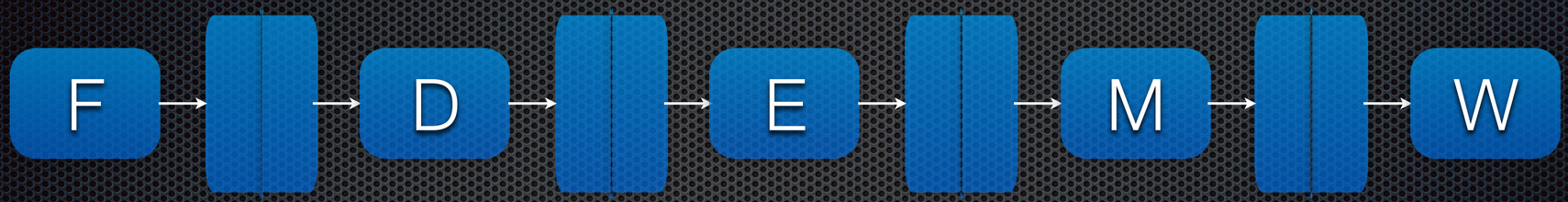
- Notice that after the pipe is full there are always two instructions active at once
- Each stage does half the work of the fetch-execute cycle
- We can clock the logic twice as fast because less work needs to be done in each stage
- A dual register (next/current IR) is needed between the stages to hold the output of the first and the input of the second

- Not shown: Control Unit splits into Fetch Control and Execute Control
- Fetch Control doesn’t depend on IR contents
- Execute Control drives execution based on the instruction in the IR

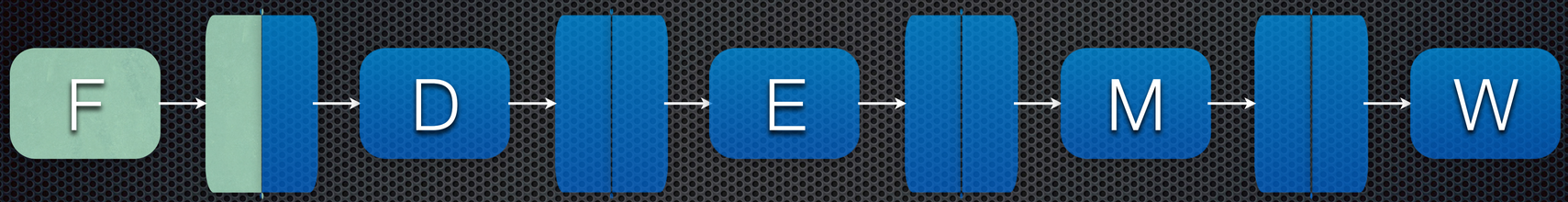
- 2X speedup is good!
- Only had to add a register and rearrange existing logic, so cost is low
- Can we do more?
- Recall that Fetch/Execute had more than just two steps for most instructions. What are they?

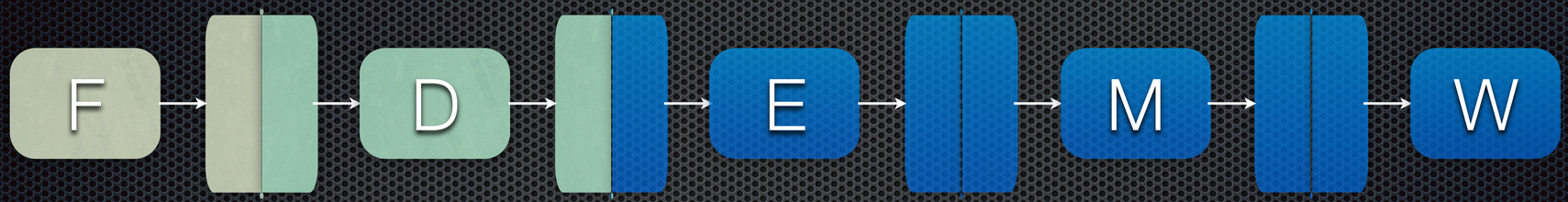
Classic Pipeline

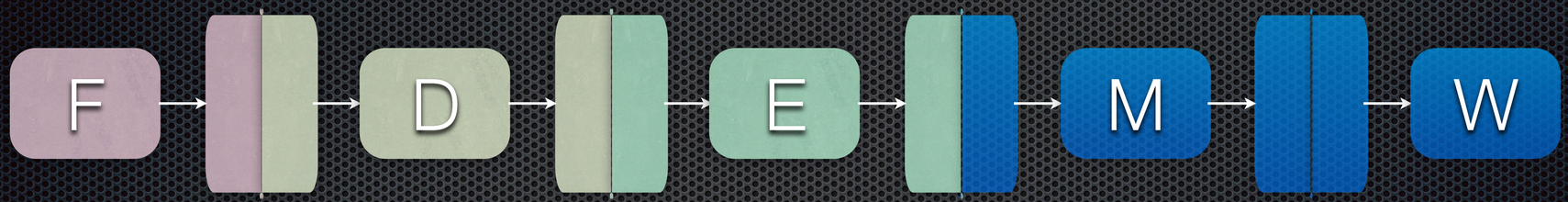
- Fetch
- Decode / Register Operand Fetch
- Execute ALU Operation
- Memory Access
- Write (result) Back to Registers

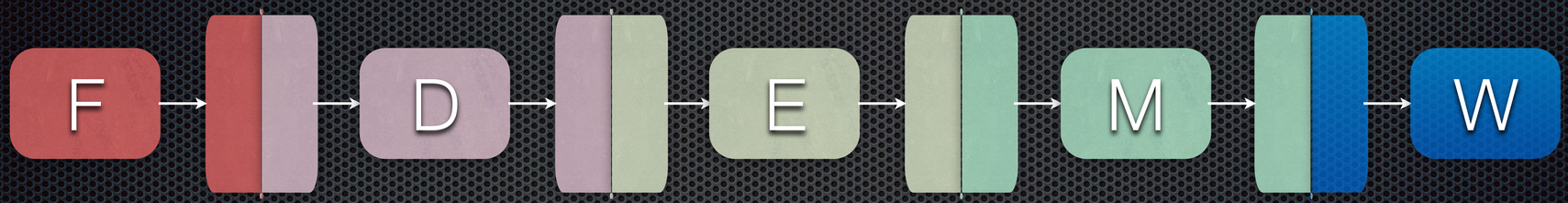

F
D
E
M
W

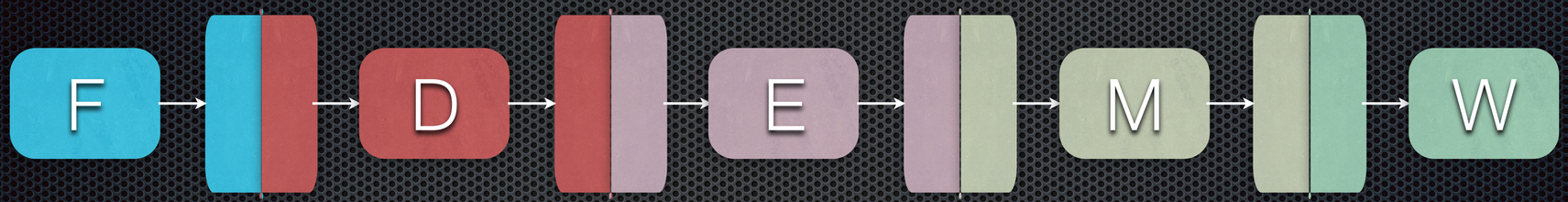
- Let’s run a series of five instructions through this pipeline
- Keep a count of the number of cycles this takes


F
D
E
M
W


F
D
E
M
W

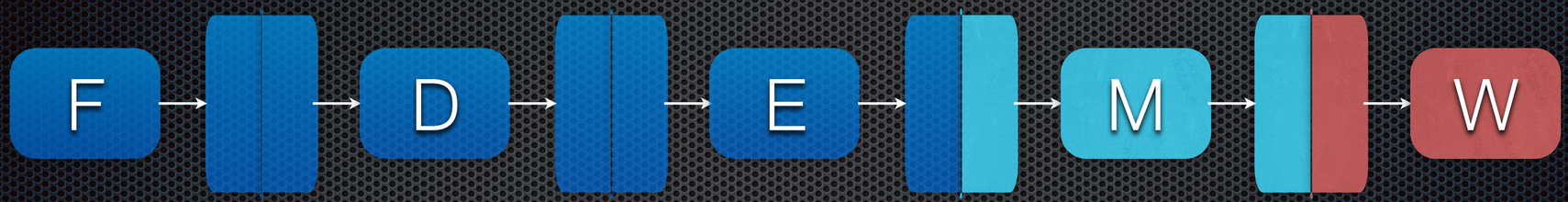

F
D
E
M
W


F
D
E
M
W

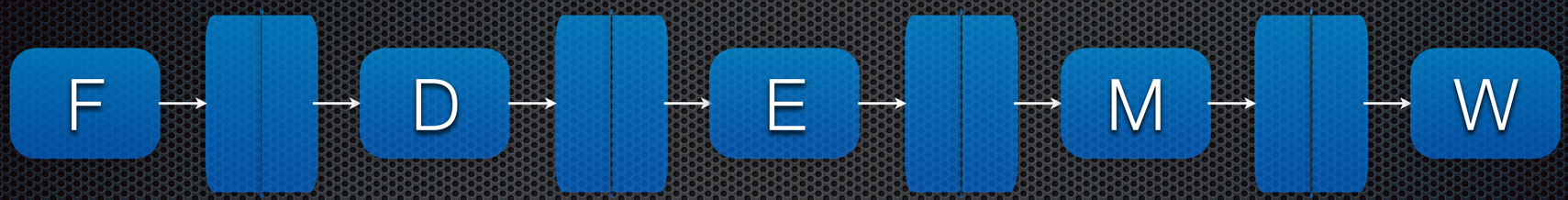

F
D
E
M
W


F
D
E
M
W


F
D
E
M
W


F
D
E
M
W

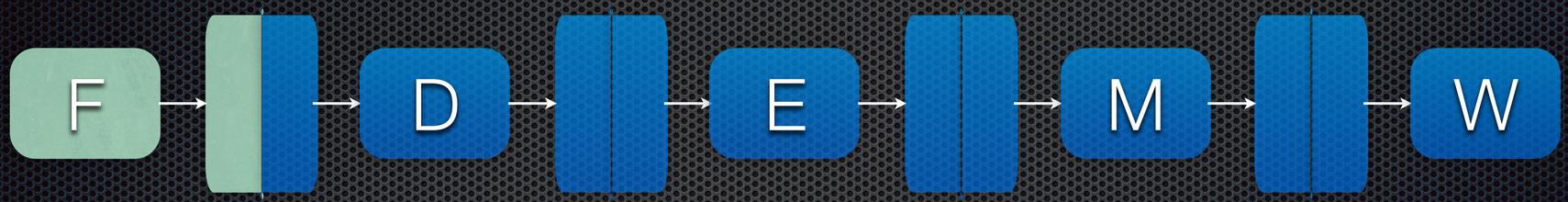

F
D
E
M
W

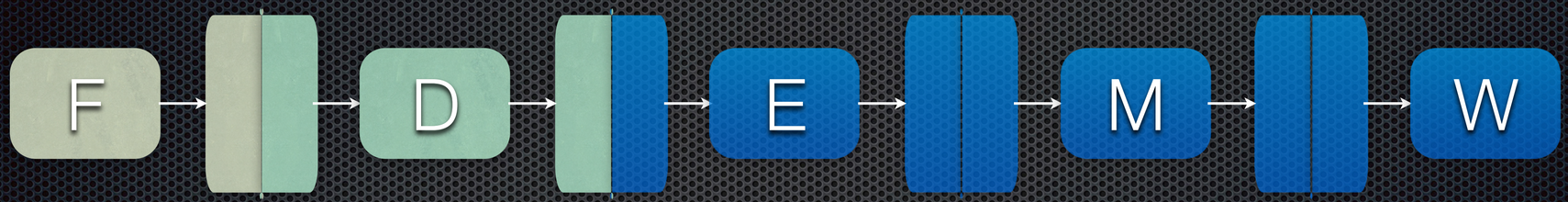

F
D
E
M
W

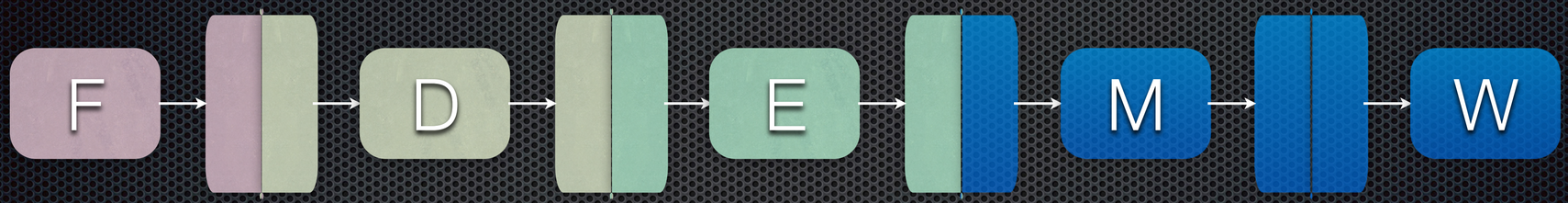
- How many cycles did it take to execute these five instructions?
- Execution is complete in Write Back

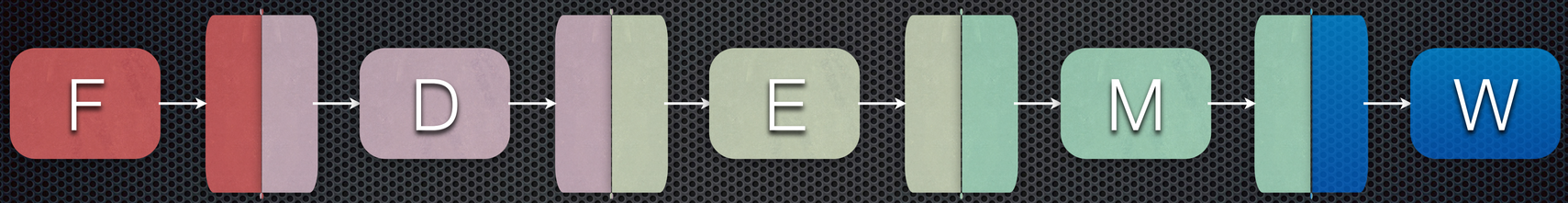
- How many cycles did it take to execute these five instructions?
- Execution is complete in Write Back
- So 9 cycles

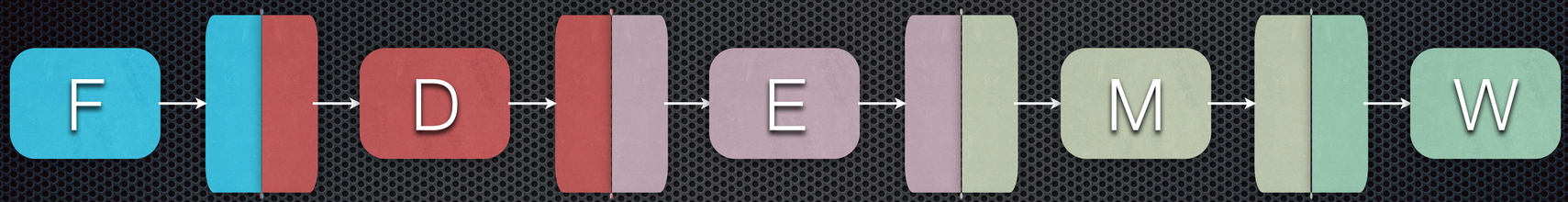
- How many will it take to execute ten?
- Let’s see...

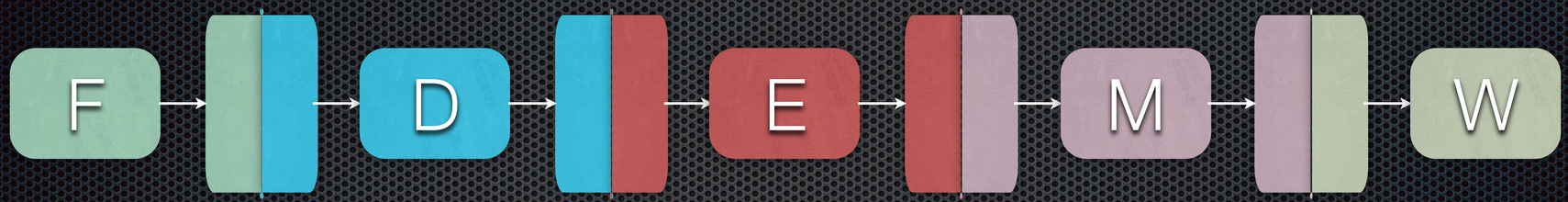

F
D
E
M
W

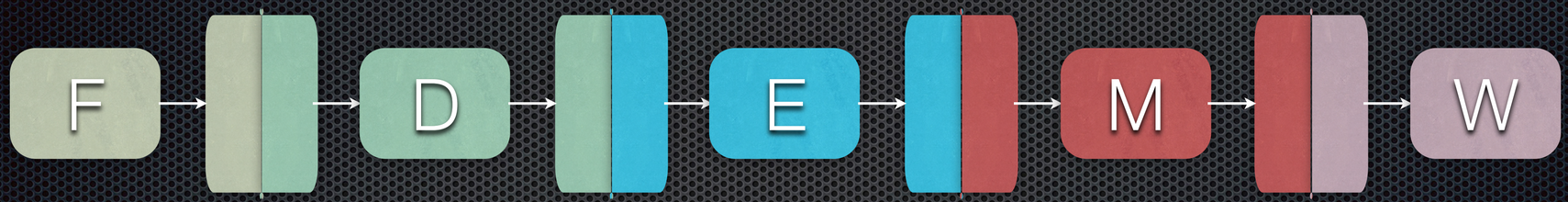

F
D
E
M
W

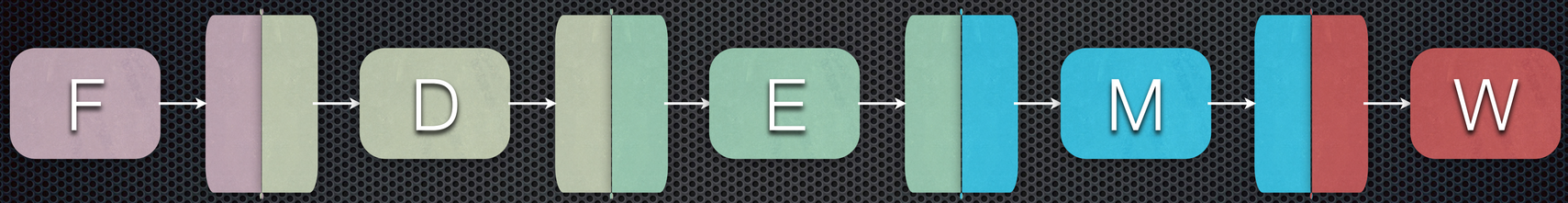

F
D
E
M
W

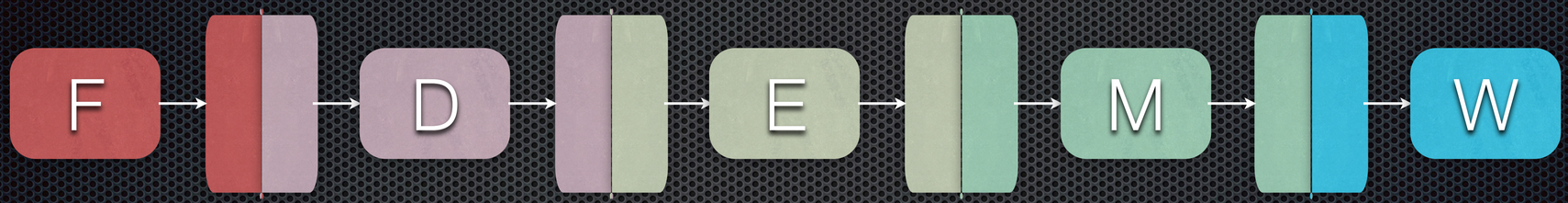

F
D
E
M
W

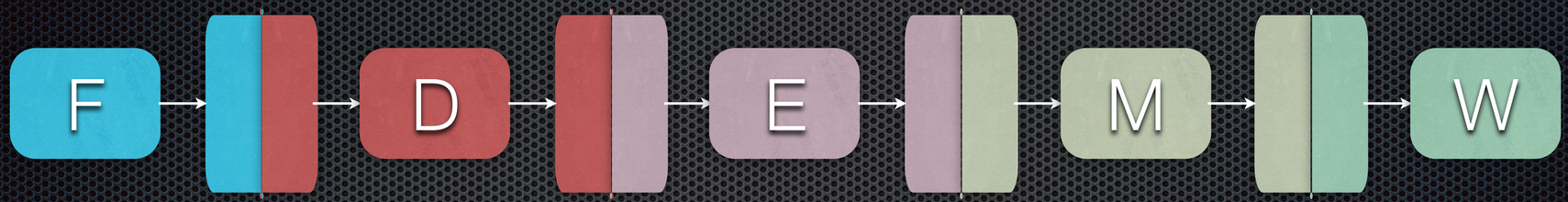

F
D
E
M
W

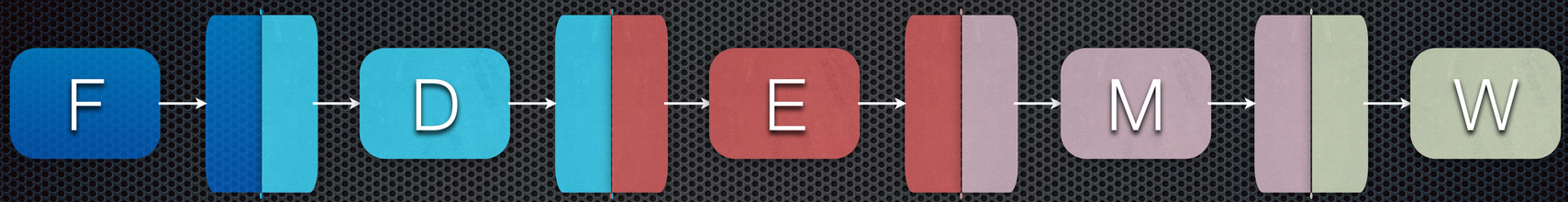

F
D
E
M
W


F
D
E
M
W

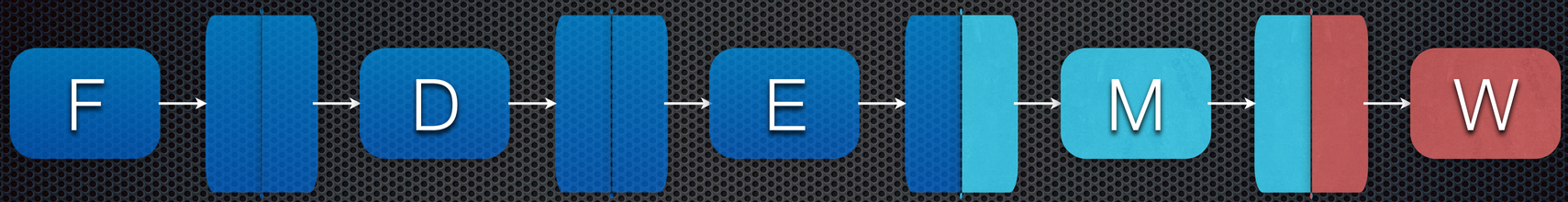

F
D
E
M
W

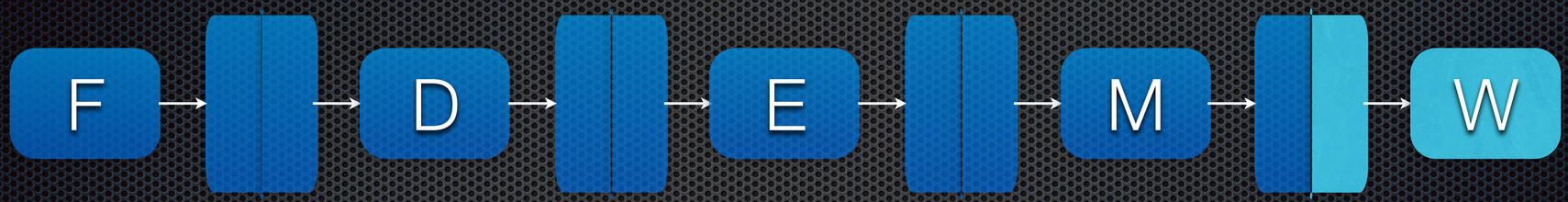

F
D
E
M
W

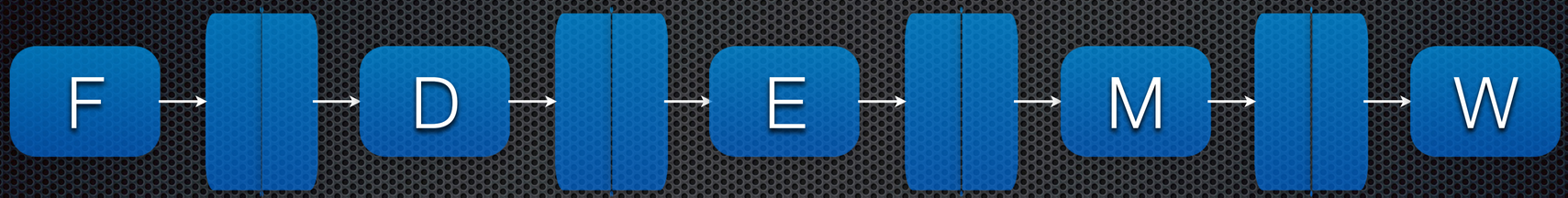

F
D
E
M
W


F
D
E
M
W


F
D
E
M
W


F
D
E
M
W


F
D
E
M
W



So 14 cycles for 10 instructions

- How many cycles will it take to execute N instructions in this five-stage pipe?
- Keep in mind that the cycle time can be roughly five times faster than for a non-pipelined fetch-execute processor
- In the non-pipelined processor, instructions take different numbers of cycles to execute
- With a pipeline, every instruction goes through every stage, but may do nothing in some stages

- Assuming the average non-pipelined instruction takes 1 cycle of 2 ns, what is the speedup for executing 20 instructions in a five-stage pipeline where the clock cycle is 0.4 ns?
- Speedup = Old time / New time

- Old time = 20 instructions * 2 ns = 40 ns
- New time = (5 fill cycles + 19 additional) * 0.4 ns = 9.6 ns
- Speedup = 40 / 9.6 = 4.17
- As the number of instructions increases, the speedup asymptotically approaches the ratio of the clock speeds (2 / 0.4 = 5)
- Note that our simulation only counts cycles, but by only allowing one instruction to pass through the pipe at a time, you get an approximation

- How many cycles will it take to execute N instructions in an M-stage pipe?
- Why don’t we build pipes that are arbitrarily long?