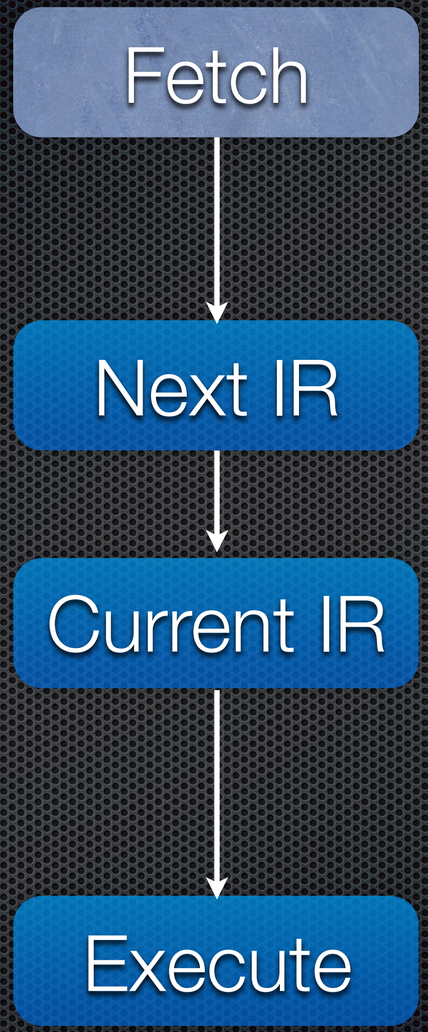
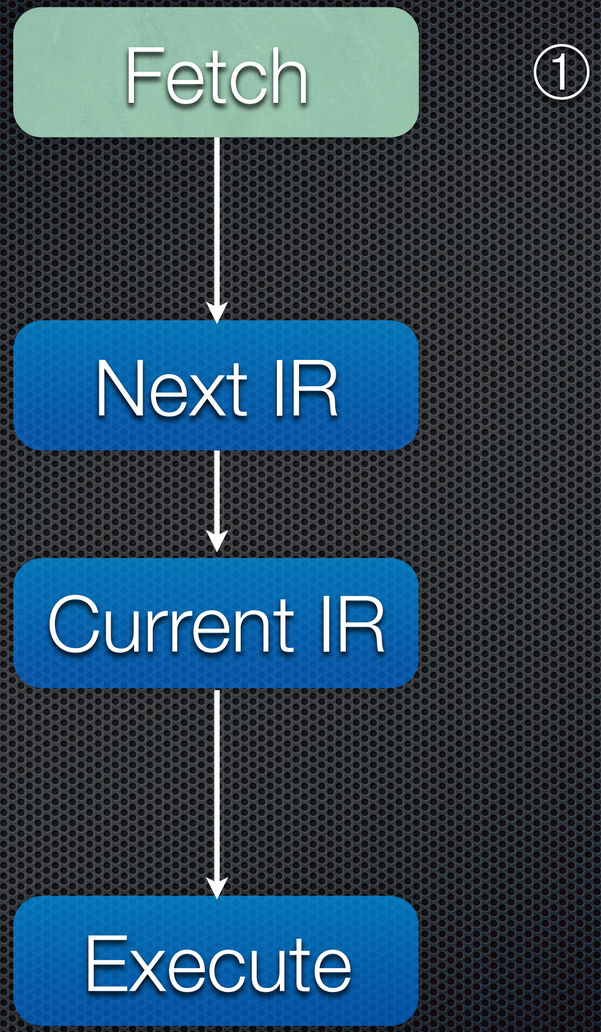
Pipelined Execution

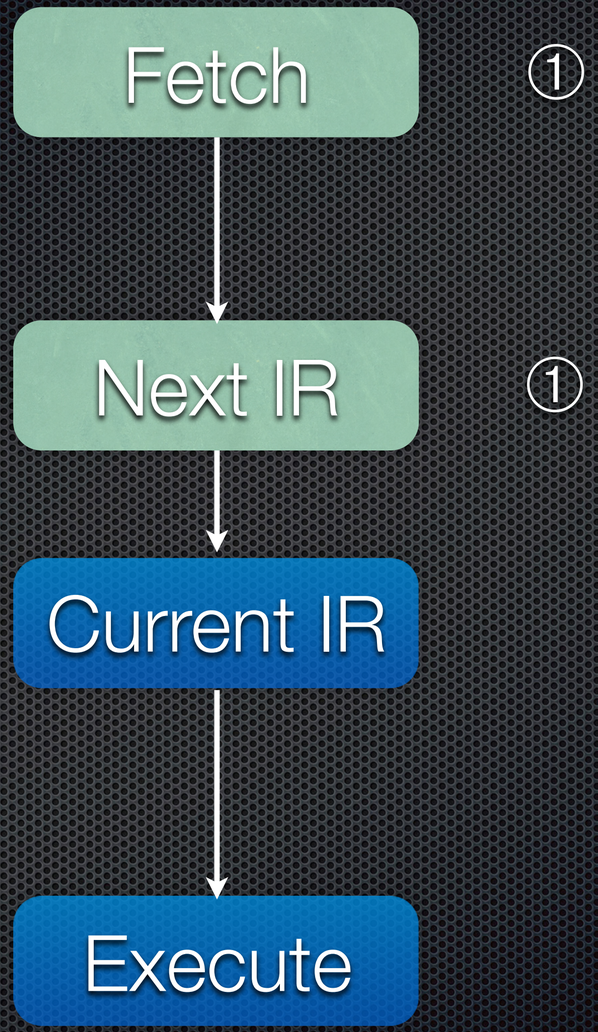
A Basic Form of Instruction Level Parallelism

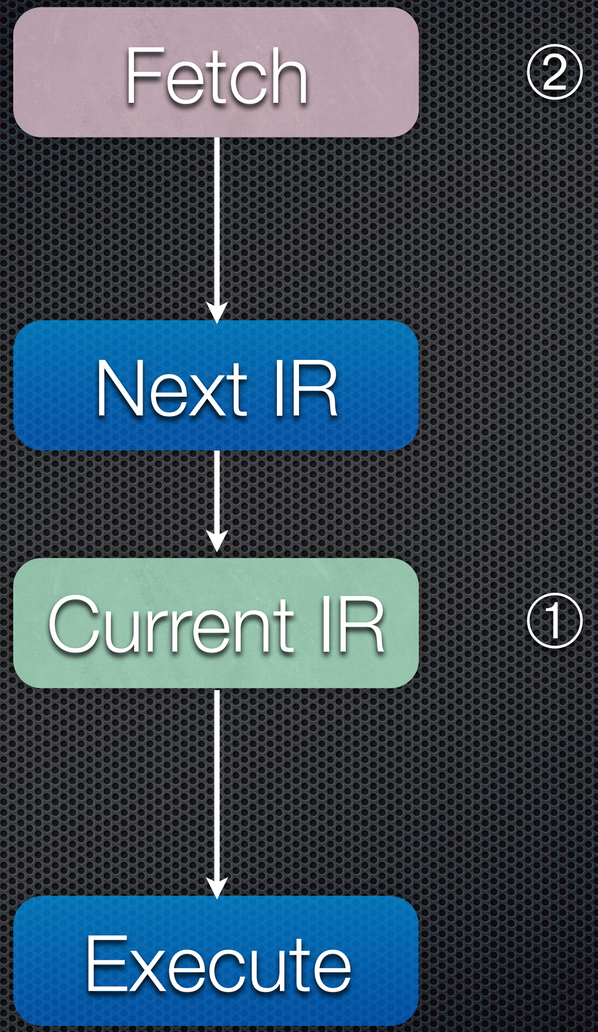
Basic Idea

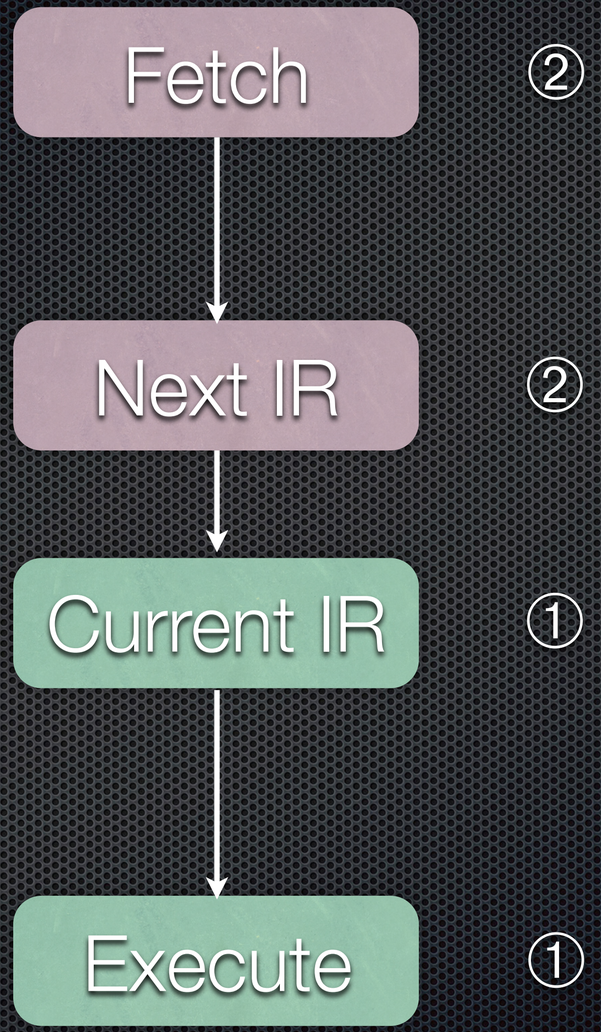
- Every instruction goes through fetch-execute
- We can fetch the next instruction while executing the current one

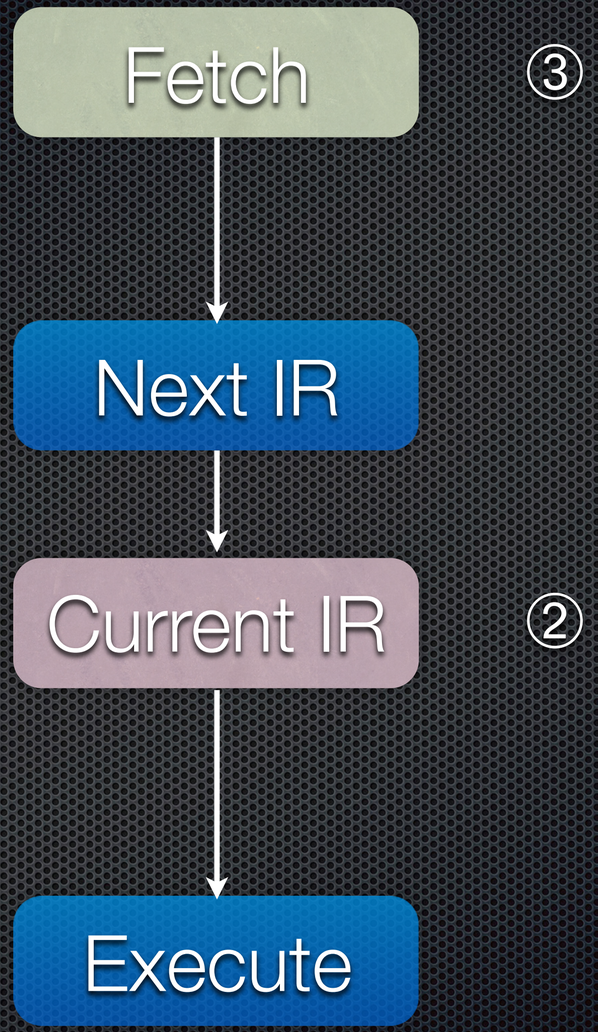

Fetch
Next IR
Current IR
Execute

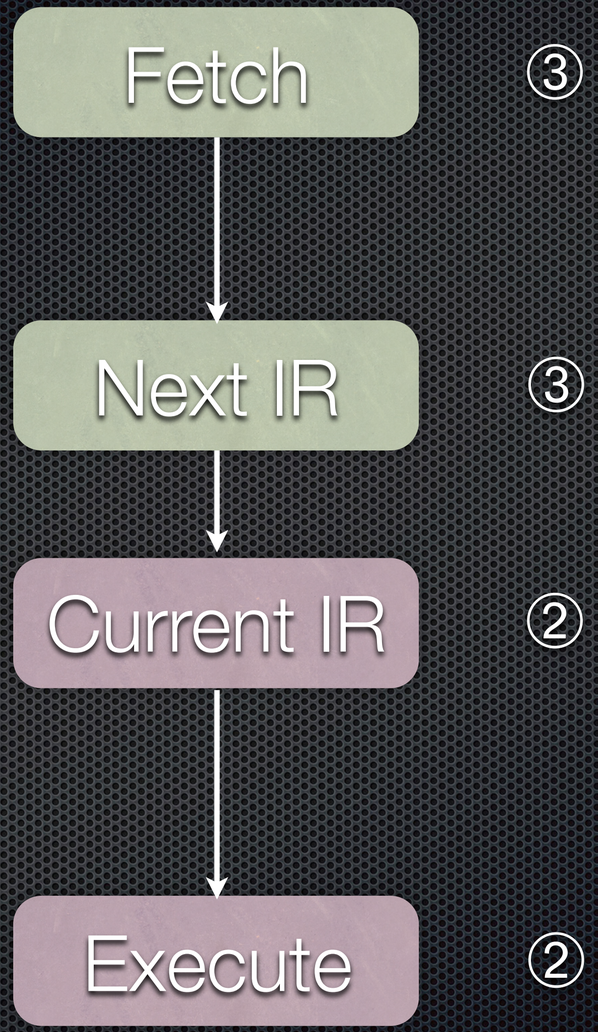

Fetch
①
Next IR
Current IR
Execute

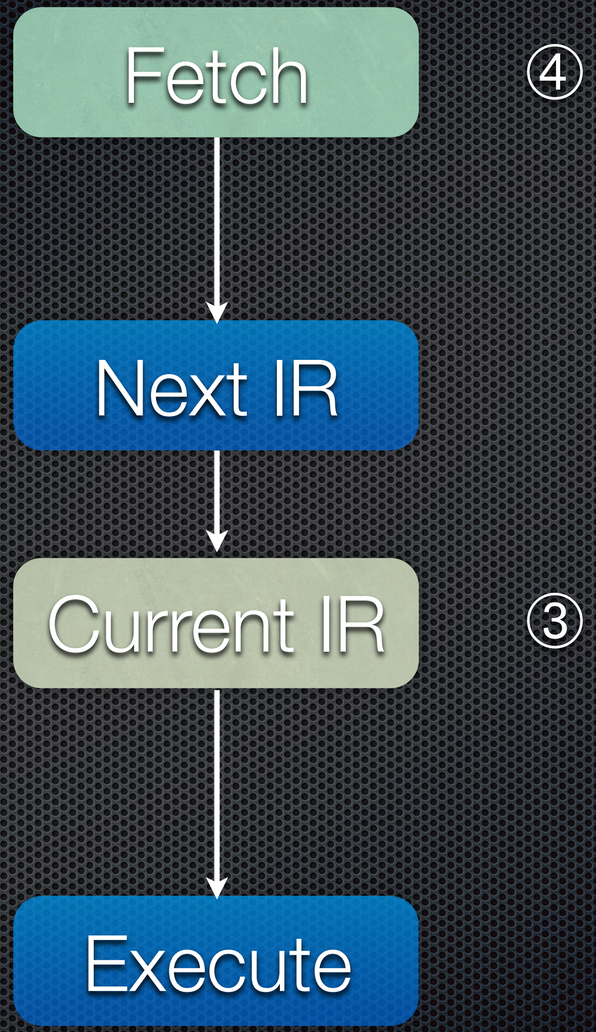

Fetch
①
Next IR
①
Current IR
Execute

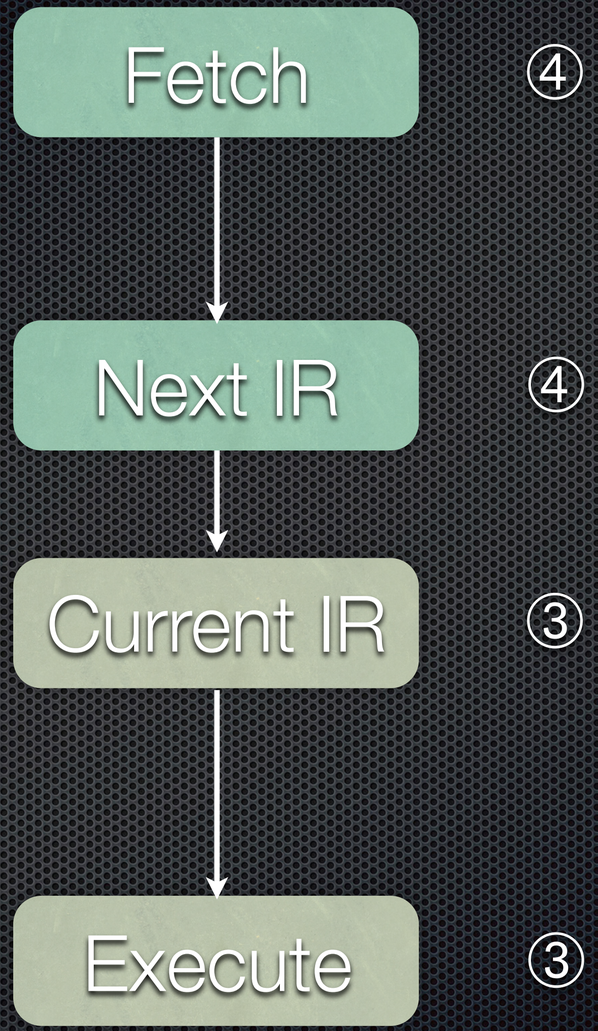

Fetch
②
Next IR
Current IR
①
Execute

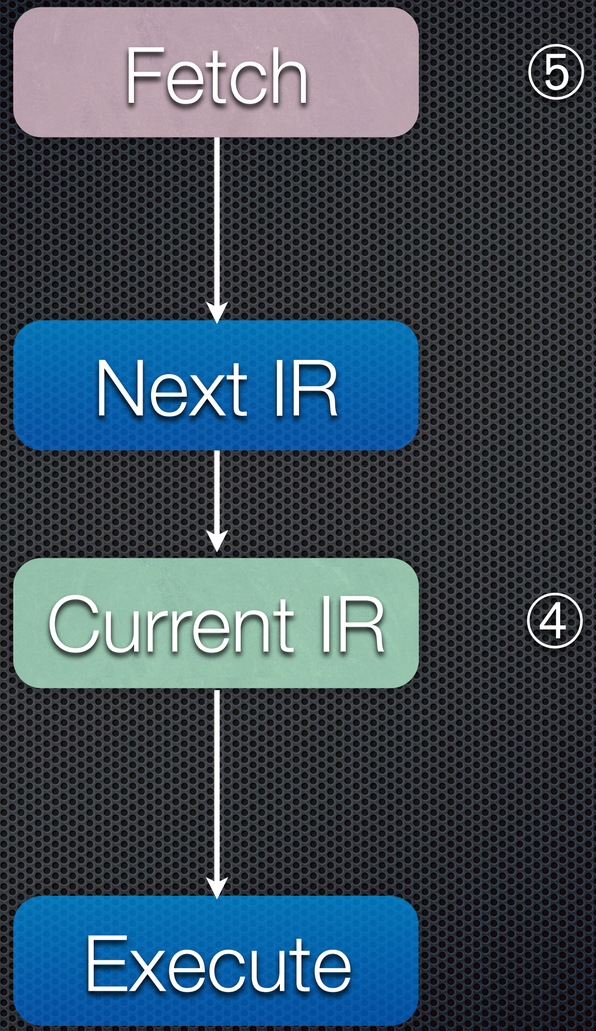

Fetch
②
Next IR
②
Current IR
①
Execute
①

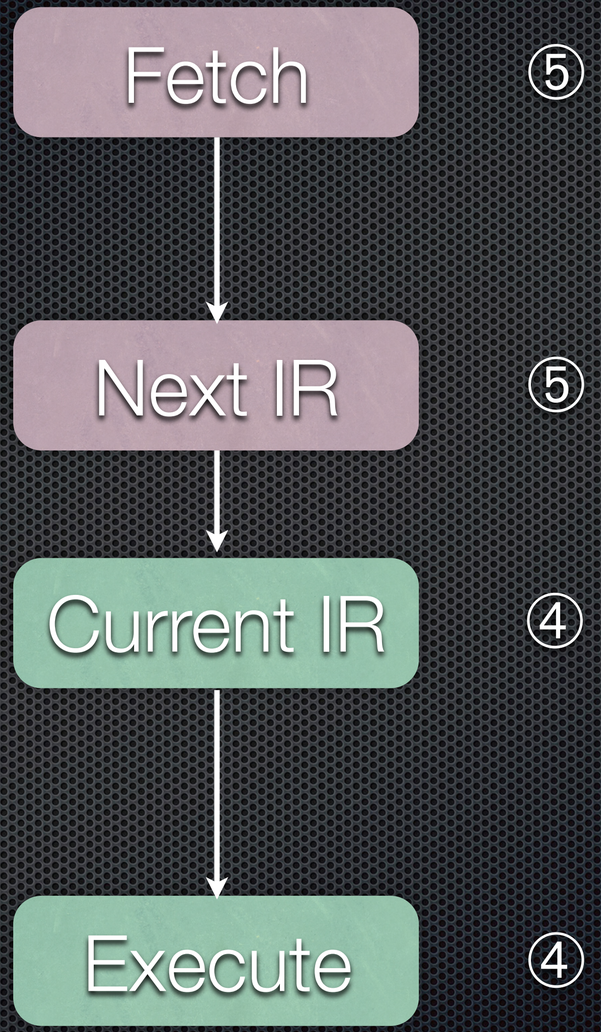

Fetch
③
Next IR
Current IR
②
Execute

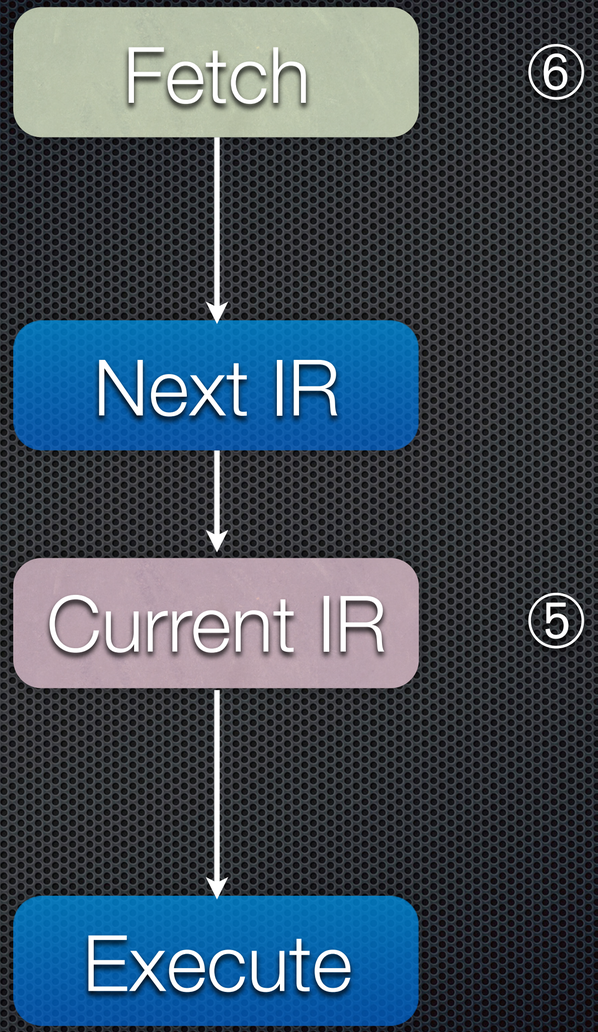

Fetch
③
Next IR
③
Current IR
②
Execute
②

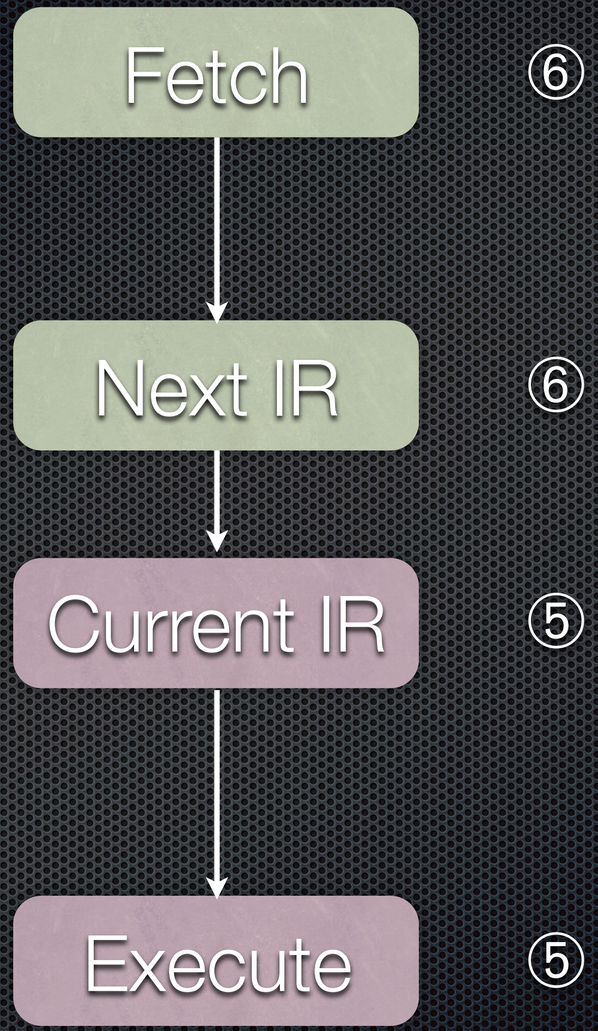

Fetch
④
Next IR
Current IR
③
Execute


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

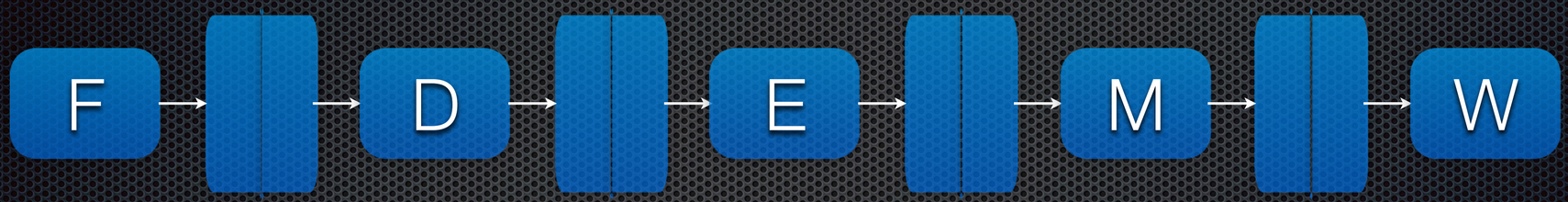
- Notice that after the pipe is full there are always two instructions active at once
- Each stage does half the work of the fetch-execute cycle
- We can clock the logic twice as fast because less work needs to be done in each stage
- A dual register (next/current IR) is needed between the stages to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

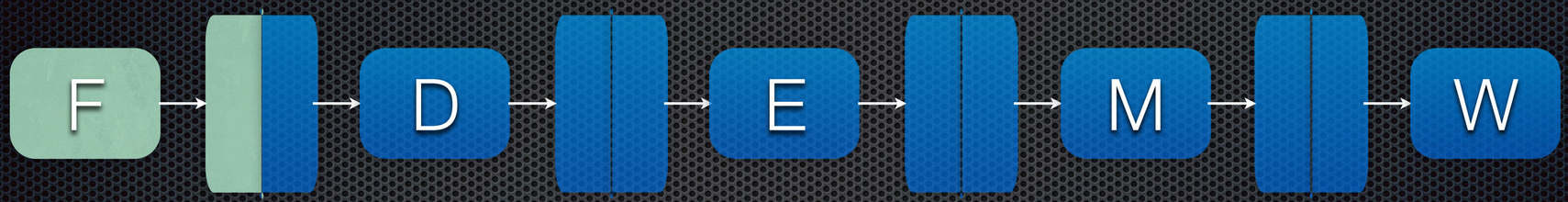
- 2X speedup is good!
- Only had to add a register and rearrange existing logic, so cost is low
- Can we do more?
- Recall that Fetch/Execute had more than just two steps for most instructions. What are they?

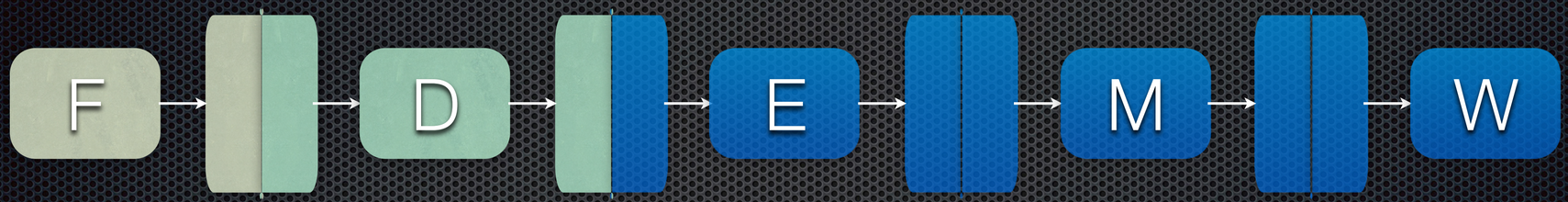
Classic Pipeline

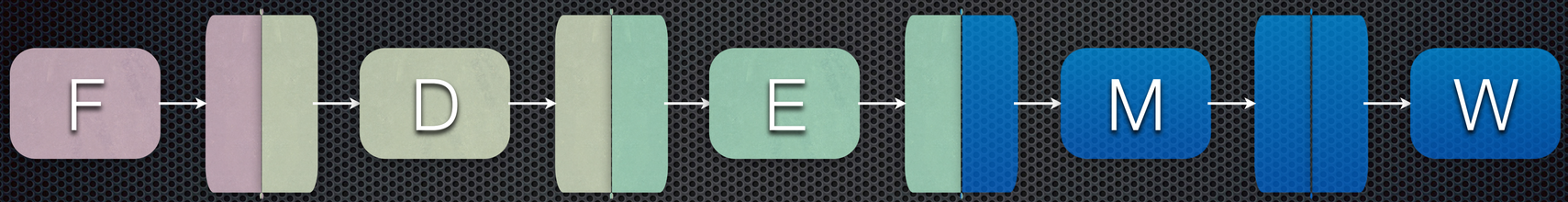
- Fetch
- Decode / Register Operand Fetch
- Execute ALU Operation
- Memory Access
- Write (result) Back to Registers

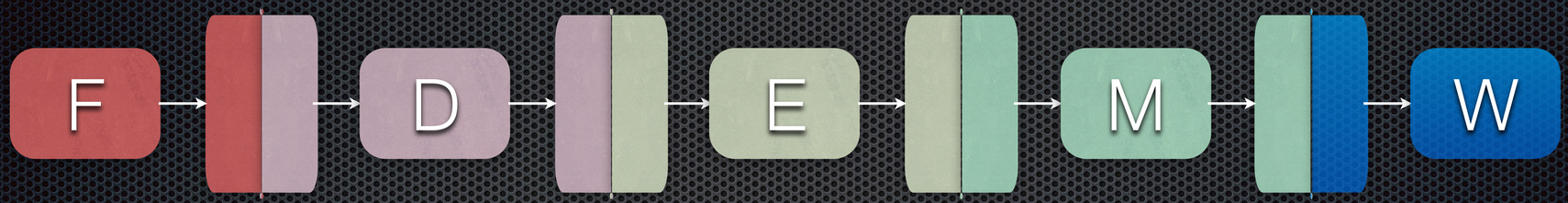

F
D
E
M
W

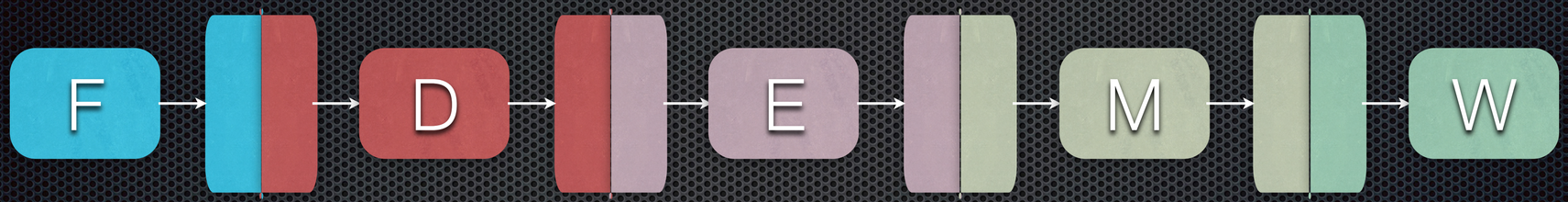
- Let’s run a series of five instructions through this pipeline
- Keep a count of the number of cycles this takes


F
D
E
M
W


F
D
E
M
W

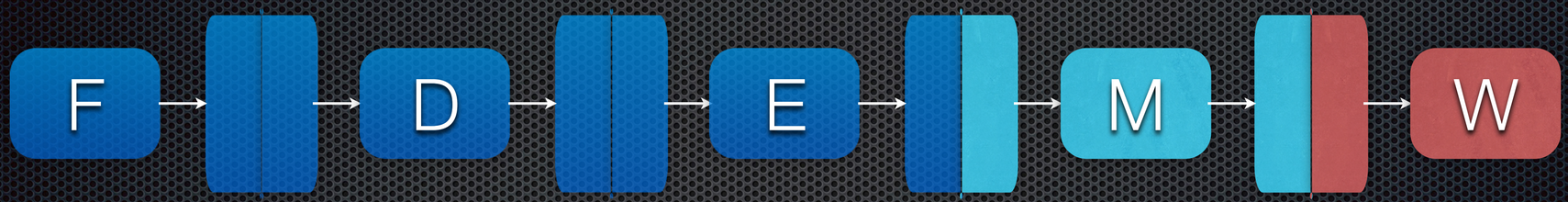

F
D
E
M
W


F
D
E
M
W

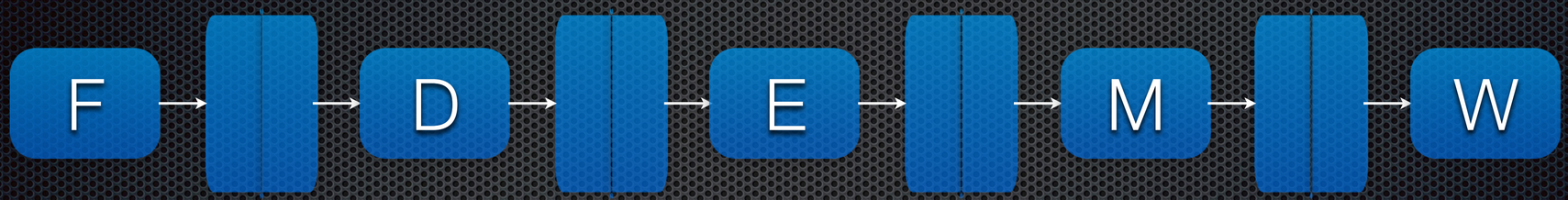

F
D
E
M
W


F
D
E
M
W


F
D
E
M
W


F
D
E
M
W

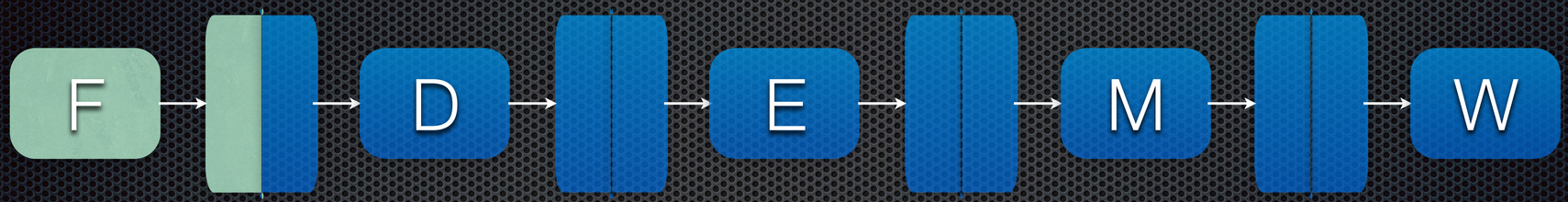

F
D
E
M
W

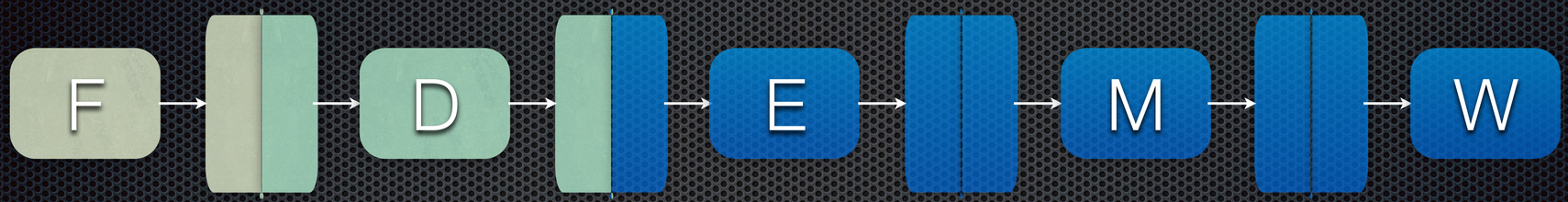

F
D
E
M
W

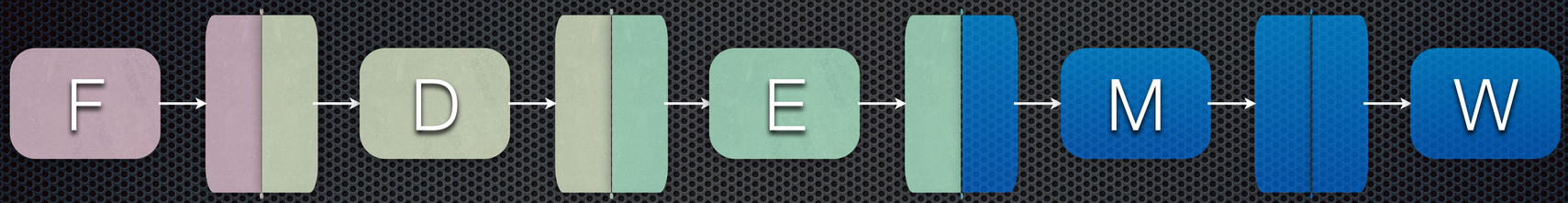
- How many cycles did it take to execute these five instructions?
- Execution is complete in Write Back

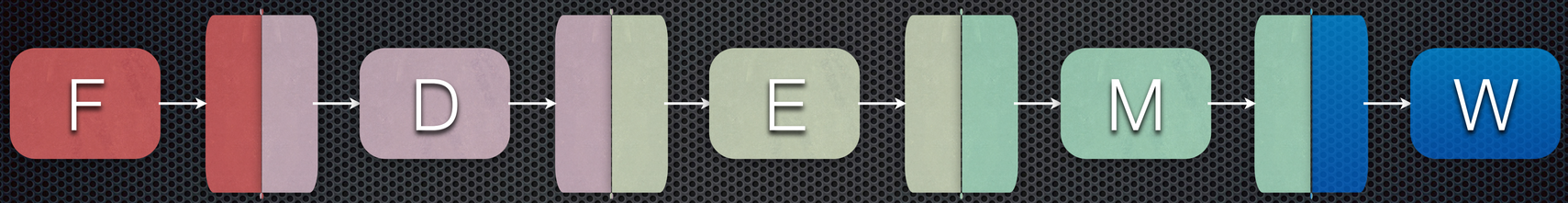
- How many cycles did it take to execute these five instructions?
- Execution is complete in Write Back
- So 9 cycles

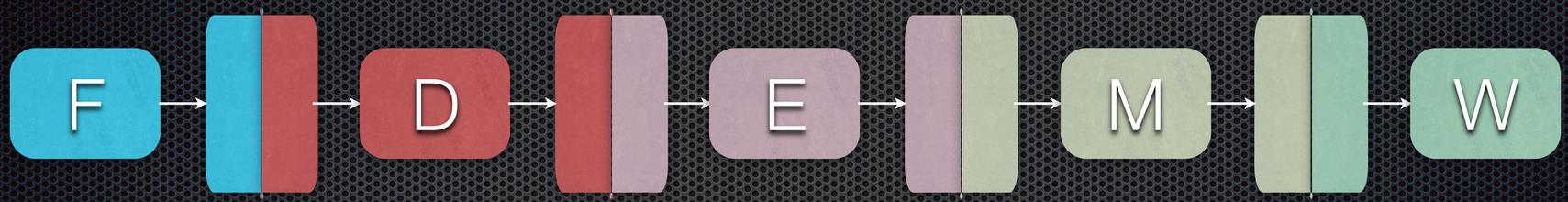
- How many will it take to execute ten?
- Let’s see...

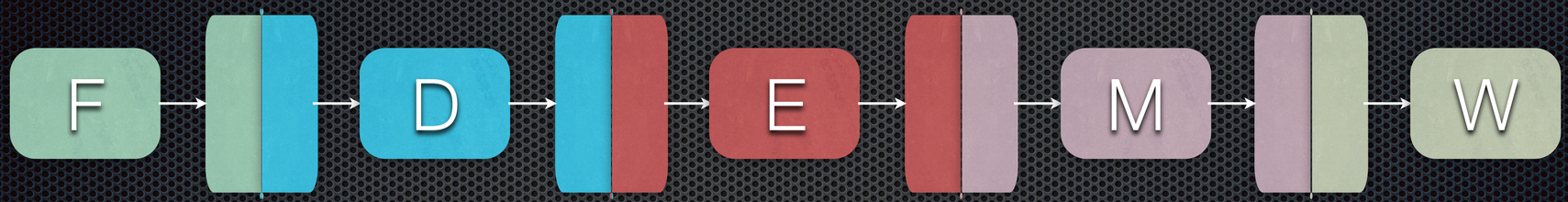

F
D
E
M
W

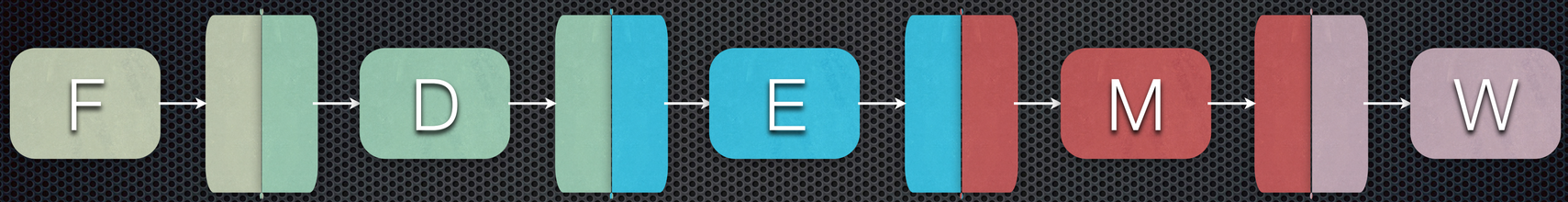

F
D
E
M
W

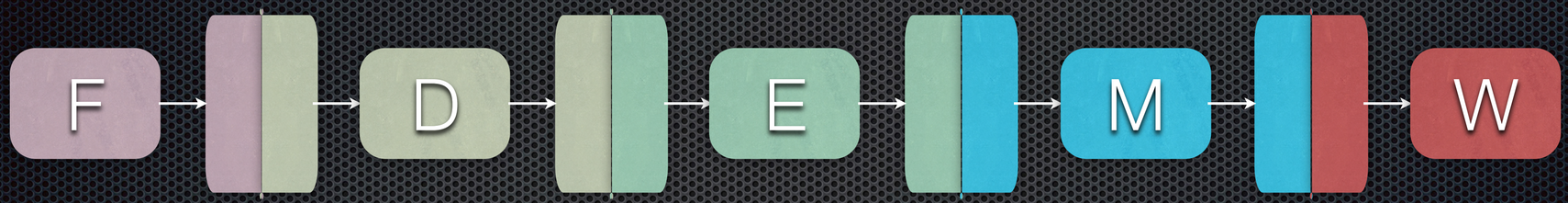

F
D
E
M
W

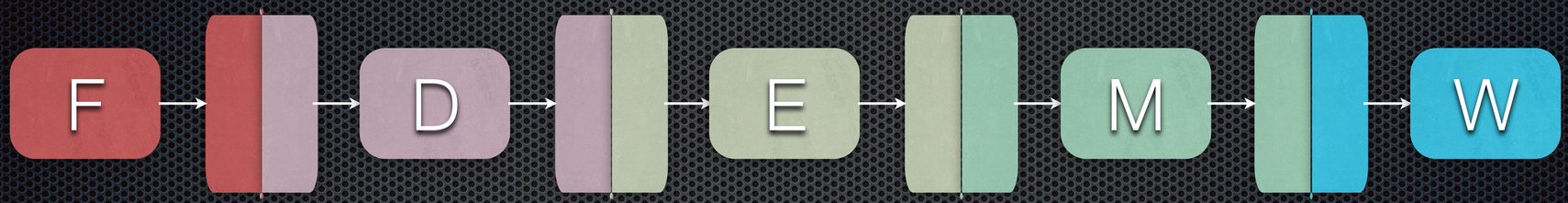

F
D
E
M
W

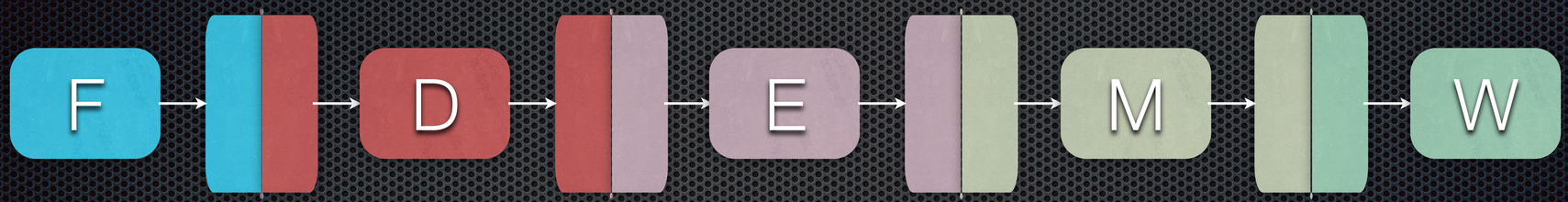

F
D
E
M
W

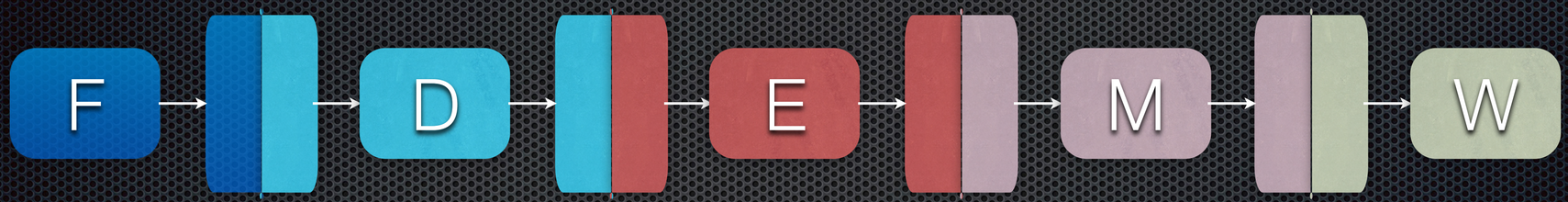

F
D
E
M
W


F
D
E
M
W

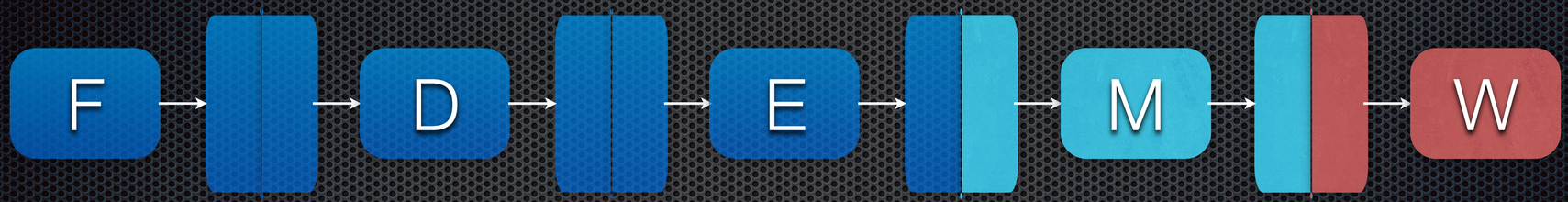

F
D
E
M
W

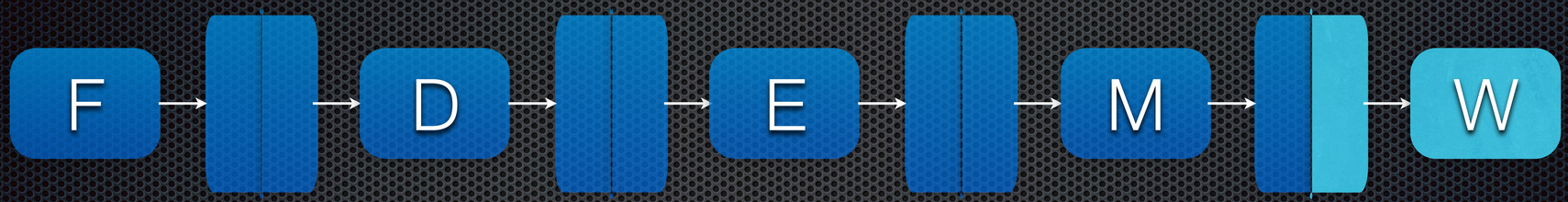

F
D
E
M
W

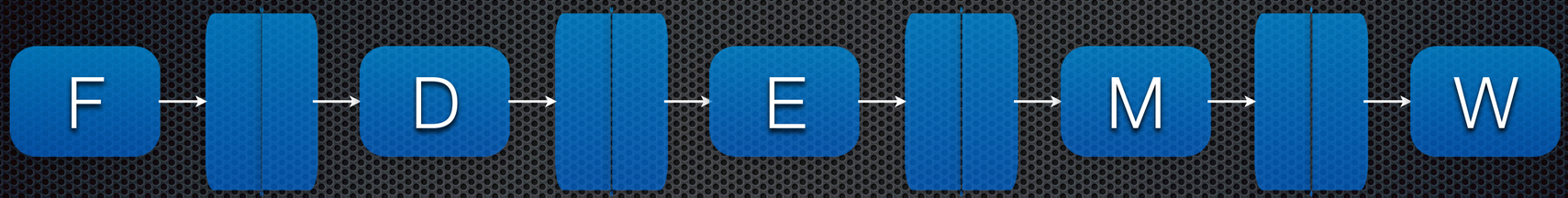

F
D
E
M
W


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

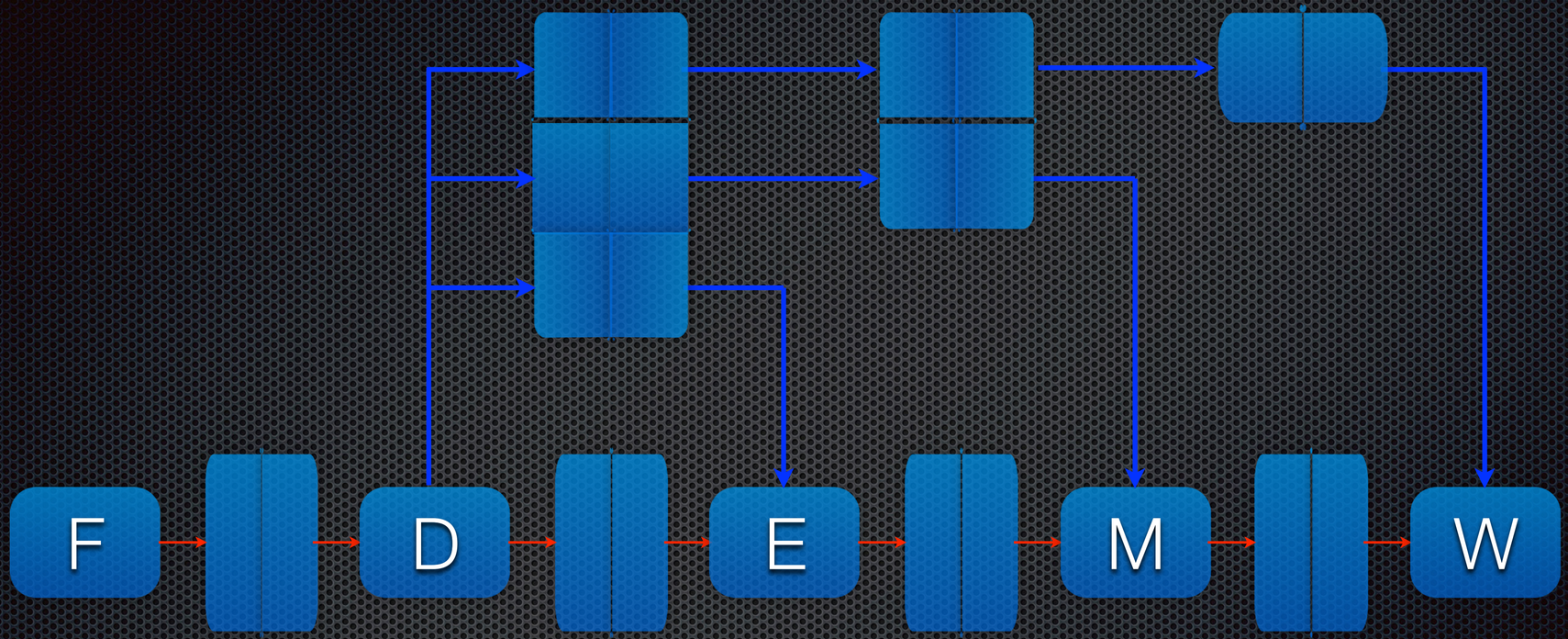
- How many cycles will it take to execute N instructions in this five-stage pipe?
- Keep in mind that the cycle time can be roughly five times faster than for a non-pipelined fetch-execute processor
- In the non-pipelined processor, instructions take different numbers of cycles to execute
- With a pipeline,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

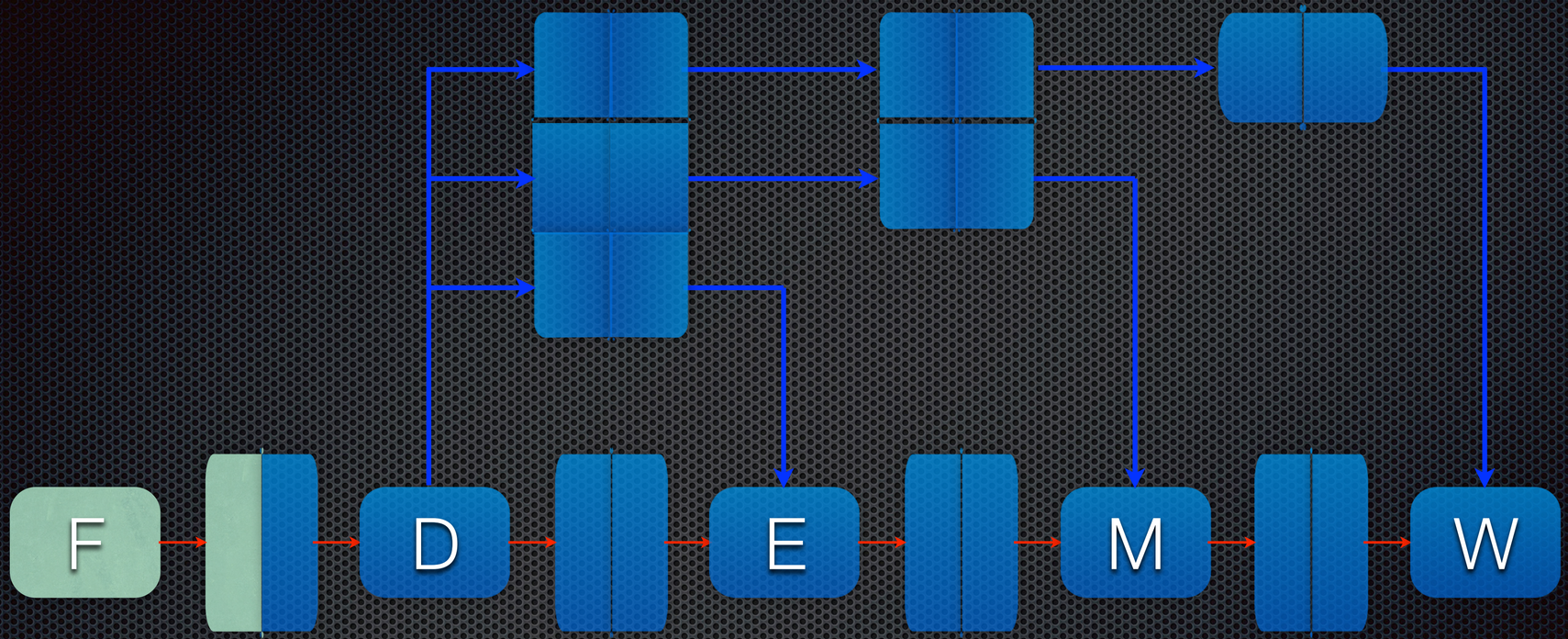
- Assuming the average non-pipelined instruction takes 3 cycles of 2 ns each, what is the speedup for executing 20 instructions in a five-stage pipeline where the clock cycle is 0.4 ns?
- Speedup = Old time / New time

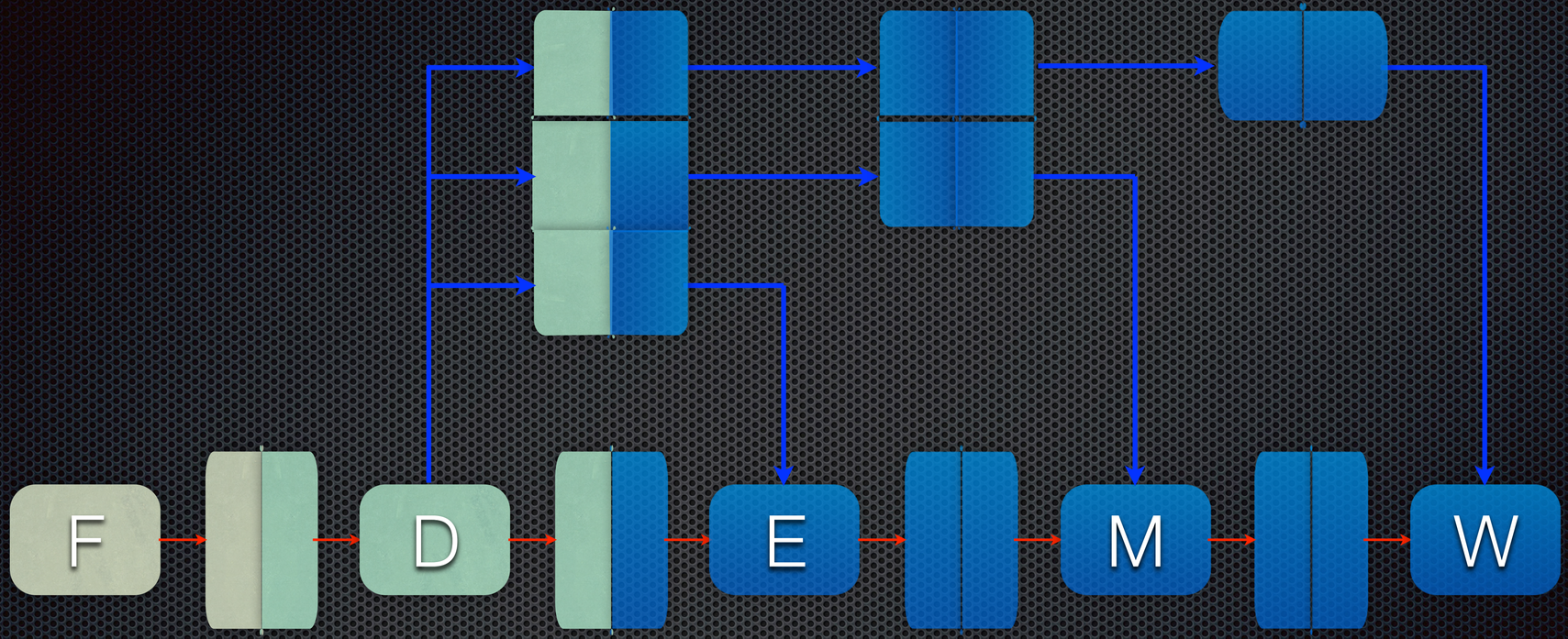
- How many cycles will it take to execute N instructions in an M-stage pipe?
- Why don’t we build pipes that are arbitrarily long?

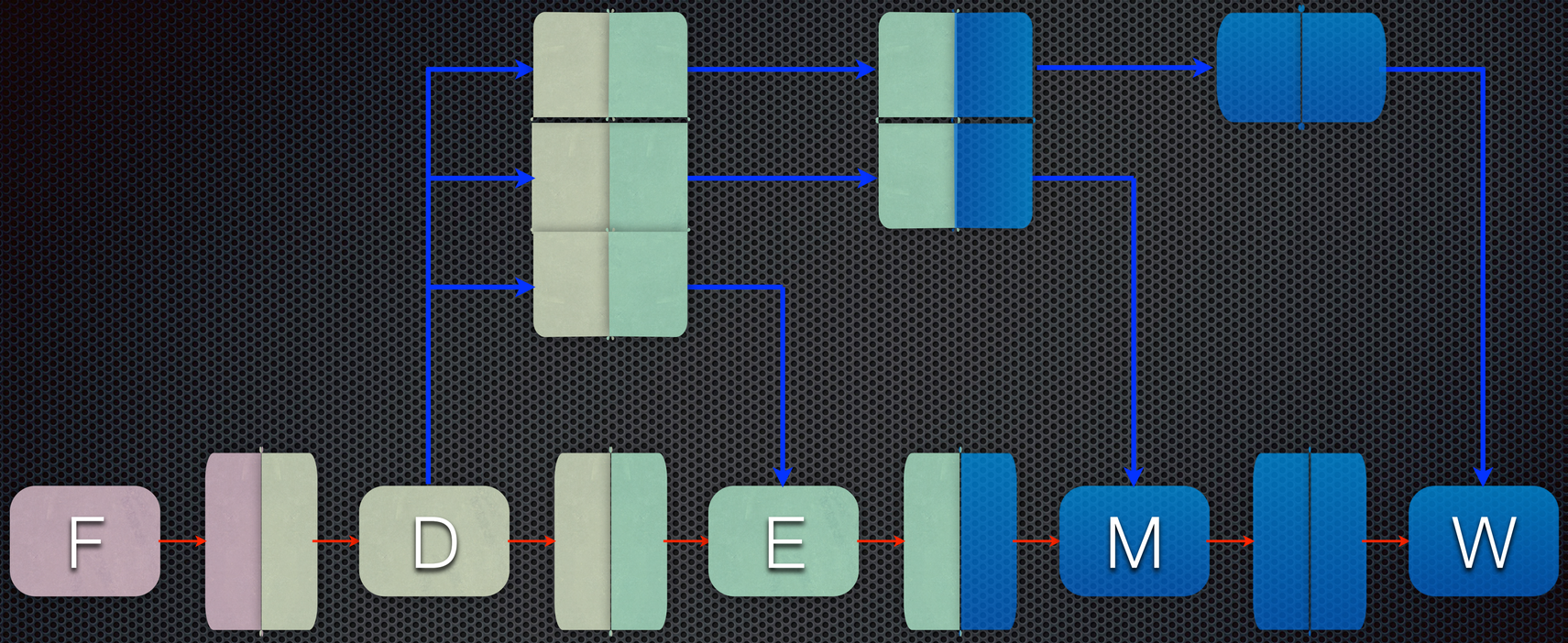
Pipeline Control

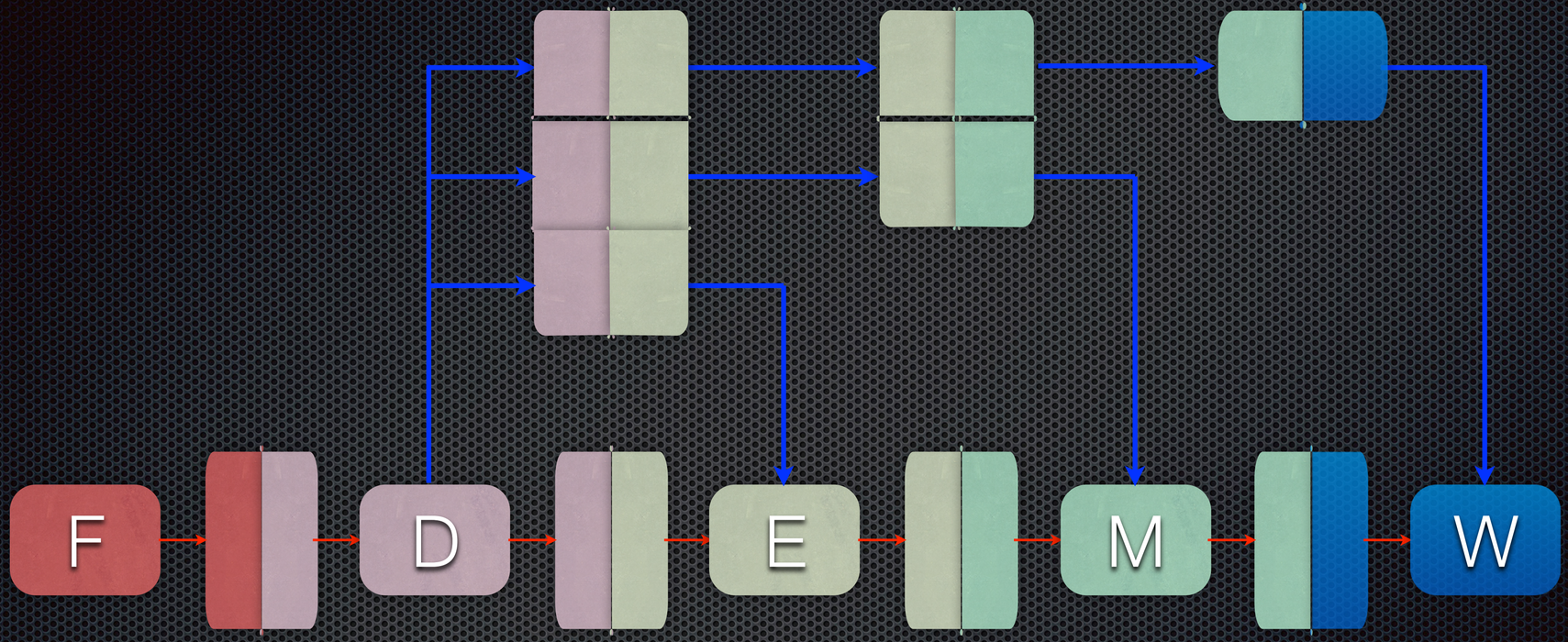
- How does the CU keep track of what each stage should be doing for each instruction in the pipe?
- At decode, it generates all of the control signals for the current instruction, for every stage, and stores them in a set of control registers
- The control registers are also pipelined
- At each stage, one register is “consumed” and the others are passed on to the next stage

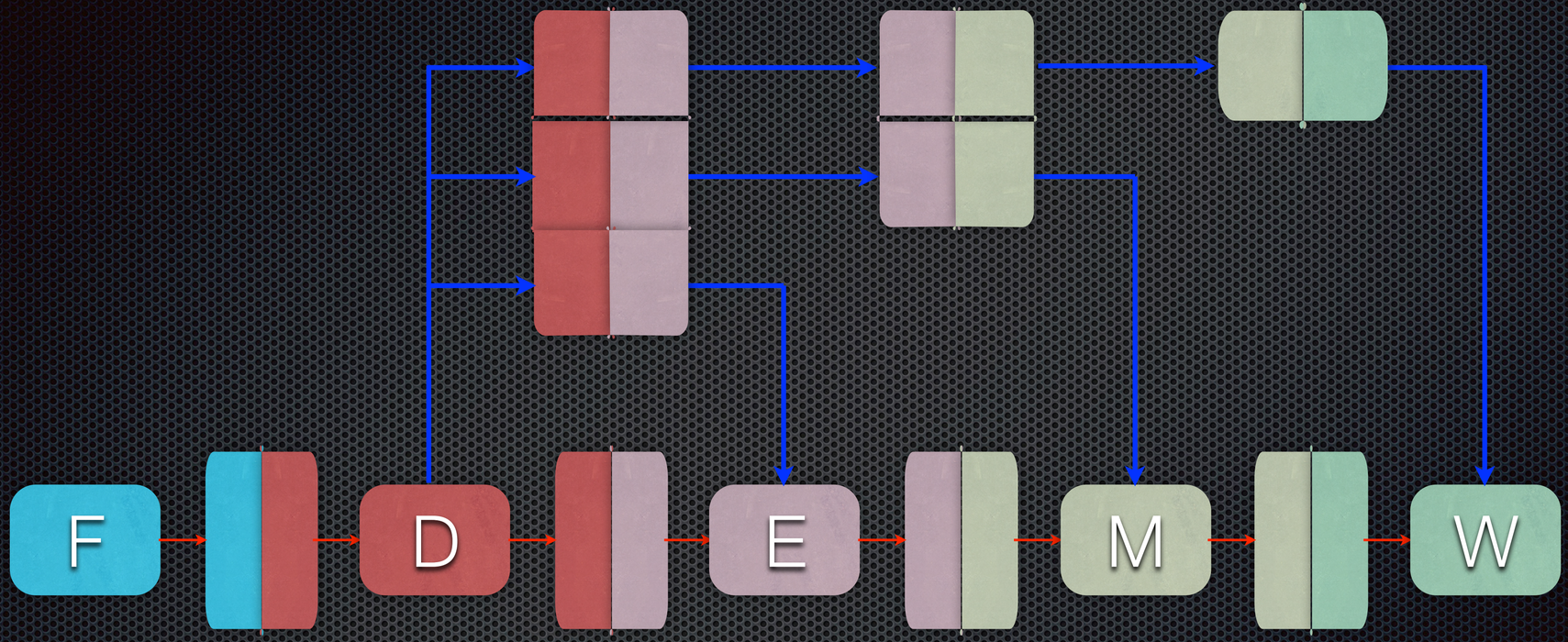

F
D
E
M
W

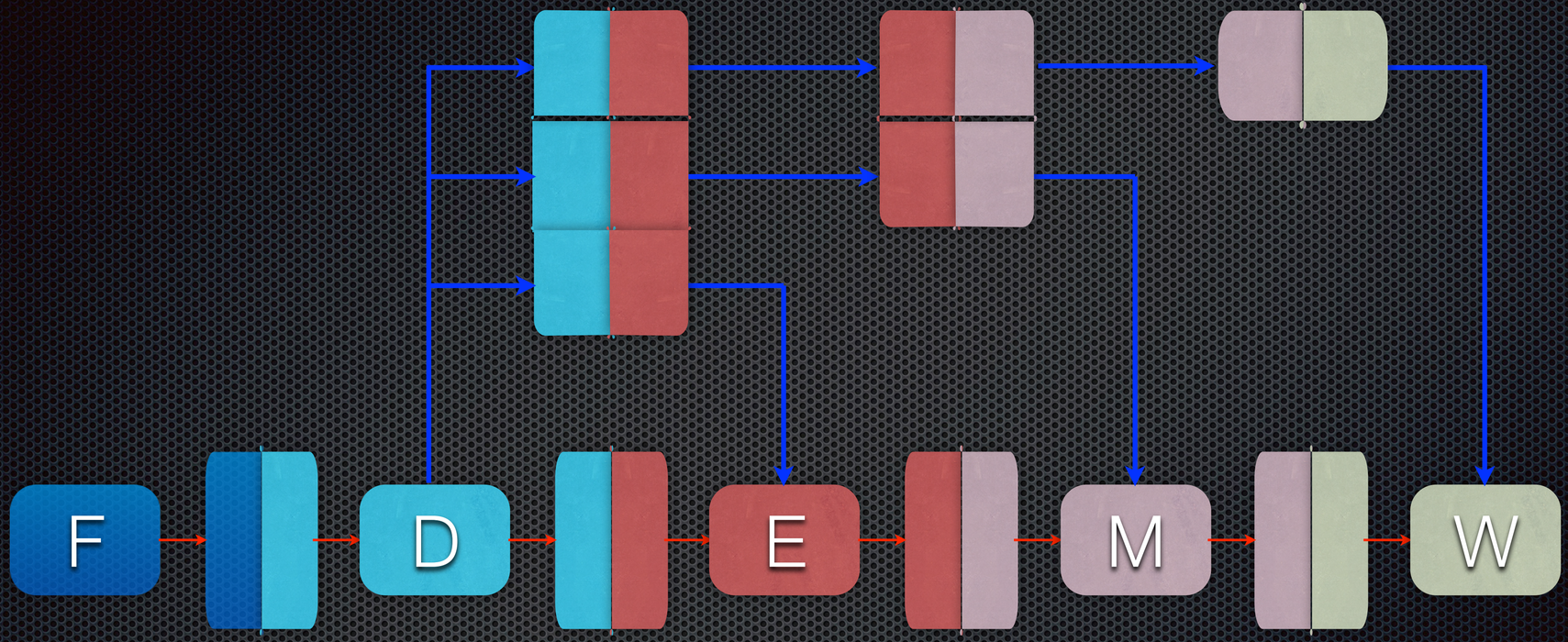

F
D
E
M
W

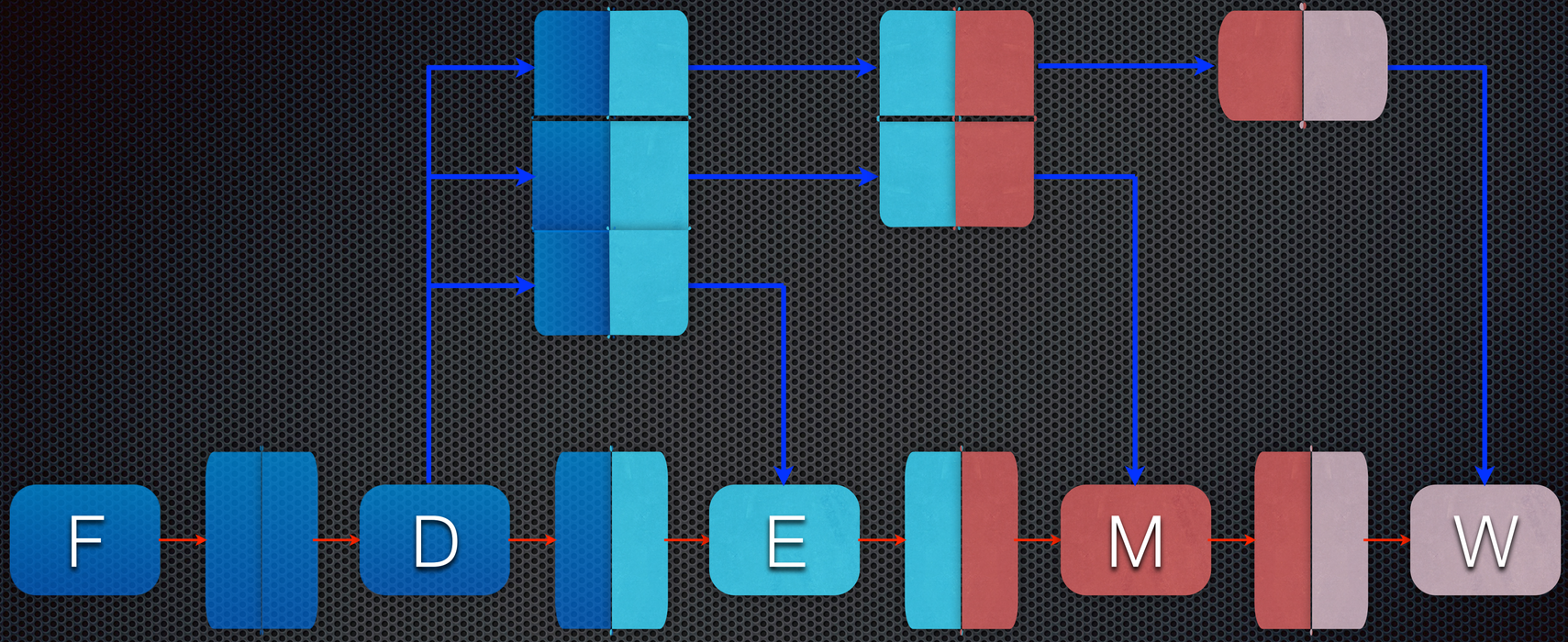

F
D
E
M
W

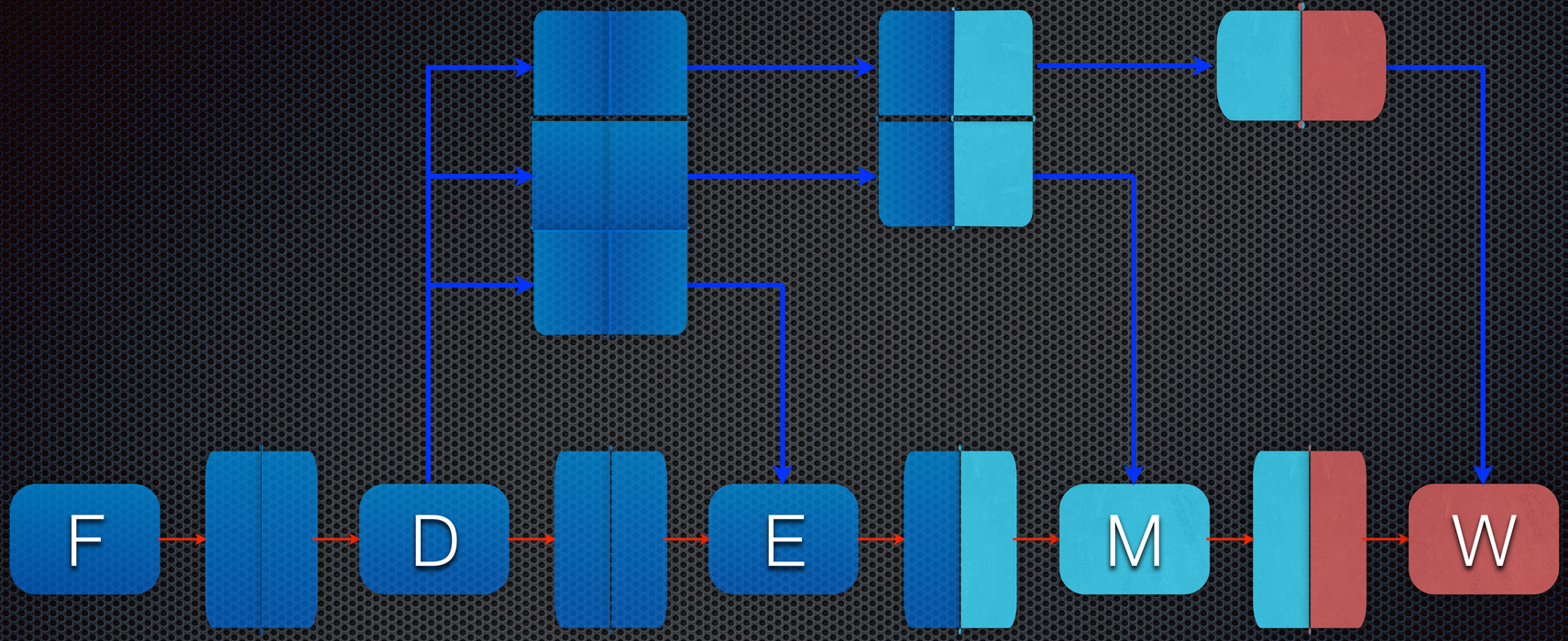

F
D
E
M
W

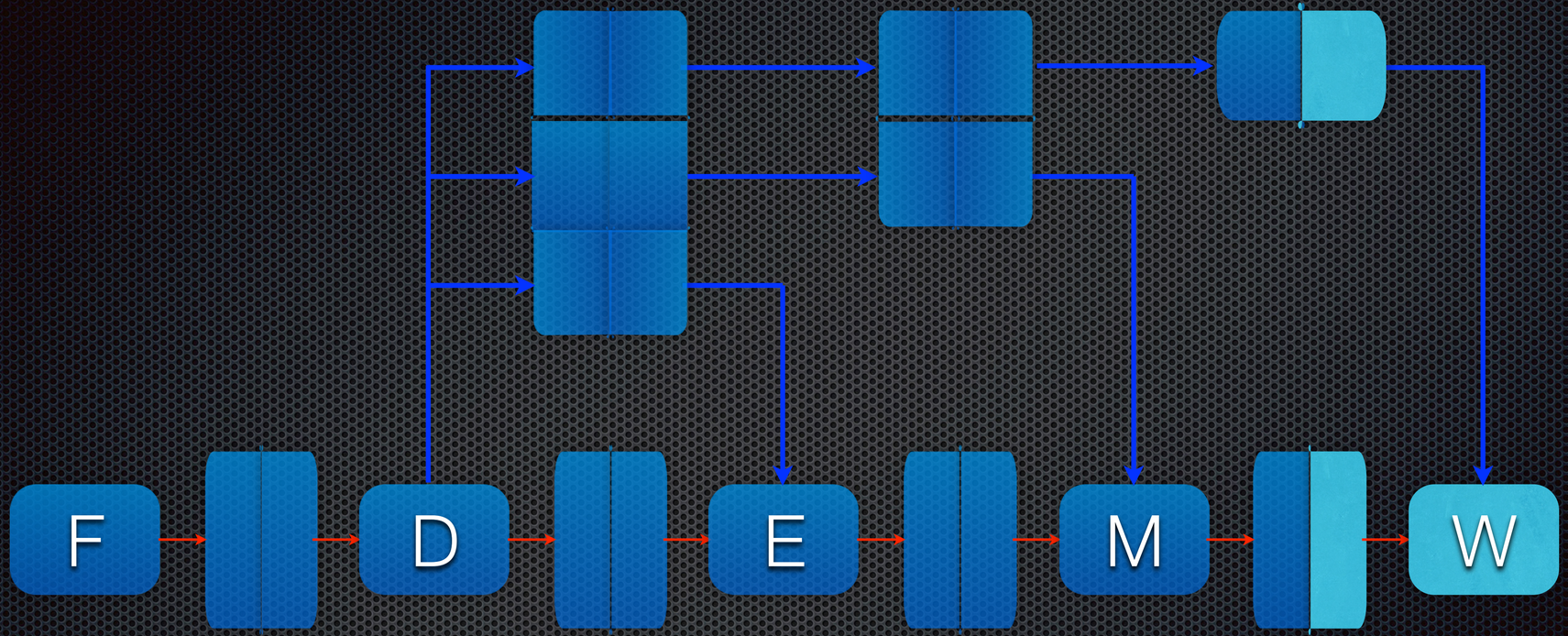

F
D
E
M
W

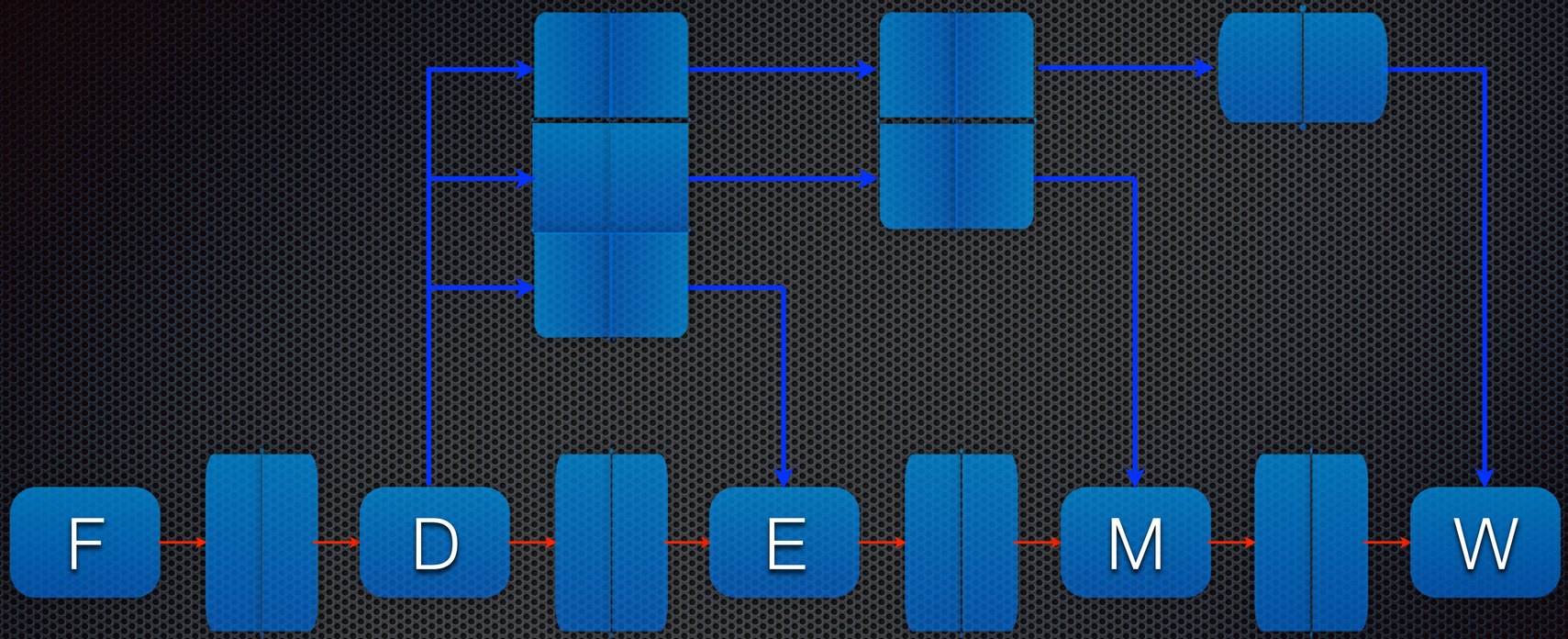

F
D
E
M
W


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

Simulation

- On each cycle, evaluation proceeds back to front
- When a stage advances, its opens a space for the one behind it to advance (like a queue)
- Stages that take multiple cycles don’t advance
- A stage is blocked from advancing if the next stage is still full
- A multi-cycle operation may be able to do work while blocked from advancing (e.g., FP DIV in Execute while a cache miss is serviced in Memory)

Simulation

- When Fetch stage is opened, an instruction is fetched
- Decode examines a fetched instruction and creates an object that has all of the control information broken into fields by stage, plus state fields (operands, result)
- At each stage, the corresponding control field in the instruction object is examined
- Using the state of the instruction (e.g., operand values), the stage carries out its task (e.g., computes a result), and updates the state (e.g., puts result in result field)

Simulation

- Non-piped simulation can still use the pipeline
- Just block fetch until the current instruction writes back
- Only one instruction will be in the pipe at a given time
  - Will take at least 5 cycles to pass through
  - Remember that pipelines are clocked roughly N times as fast

Pipelined Instruction Flow and Hazards

Rescheduling to Improve Efficiency

MIPS R4400 Pipe

- IF: Instruction First (I-cache fetch part 1)
- IS: Instruction Second (I-cache & decode)
- RF: Register File (Get operands)
- EX: EXecute (Perform operation)
- DF: Data First (D-cache fetch part 1)
- DS: Data Second (D-cache fetch part 2)
- TC: Tag Check (Check for miss)
- WB: Write Back (Save result)

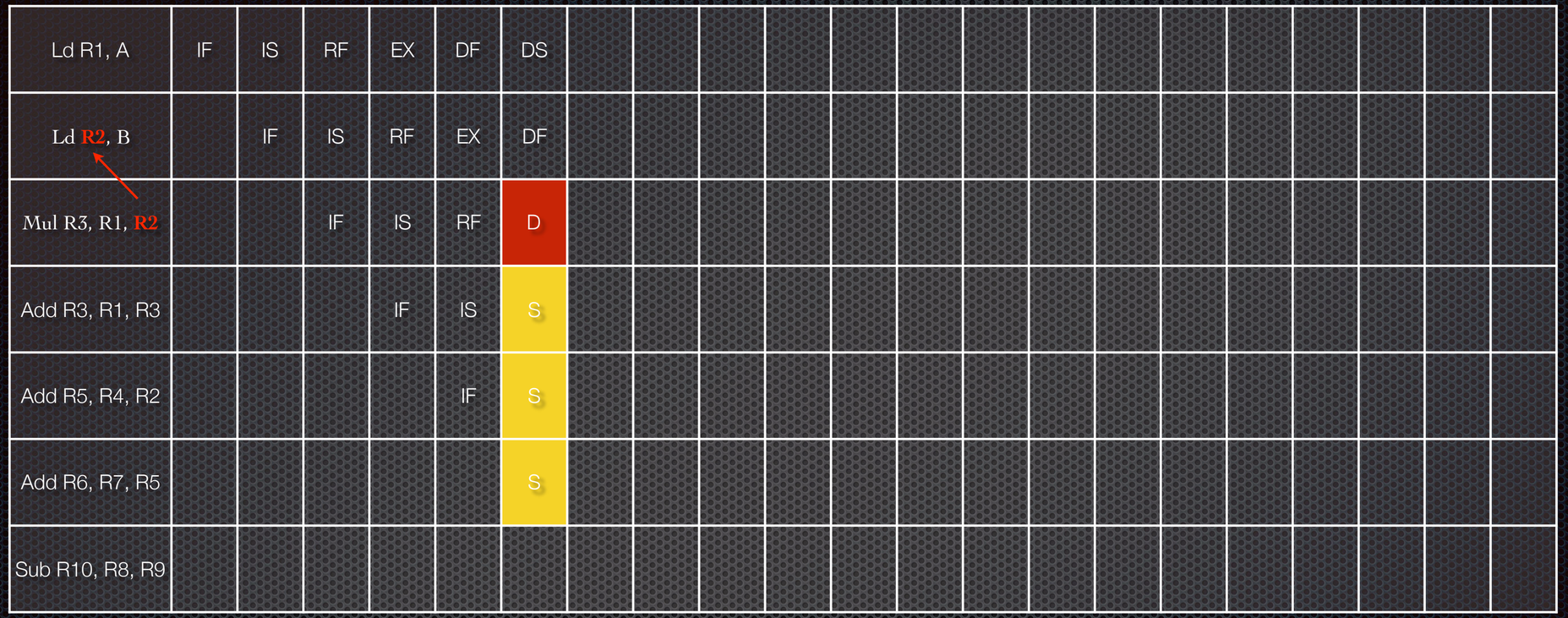
Ld R1, A	IF																		
Ld R2, B																			
Mul R3, R1, R2																			
Add R3, R1, R3																			
Add R5, R4, R2																			
Add R6, R7, R5																			
Sub R10, R8, R9																			

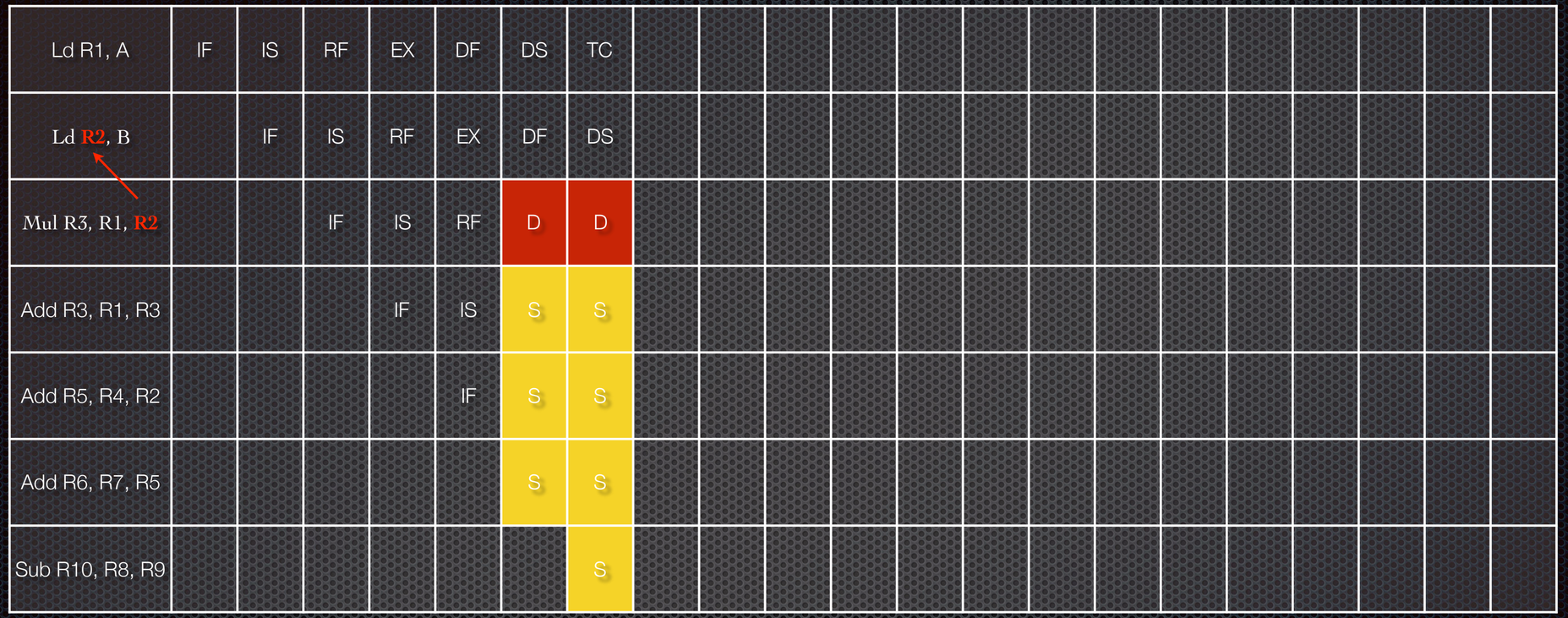
Ld R1, A	IF	IS																		
Ld R2, B		IF																		
Mul R3, R1, R2																				
Add R3, R1, R3																				
Add R5, R4, R2																				
Add R6, R7, R5																				
Sub R10, R8, R9																				

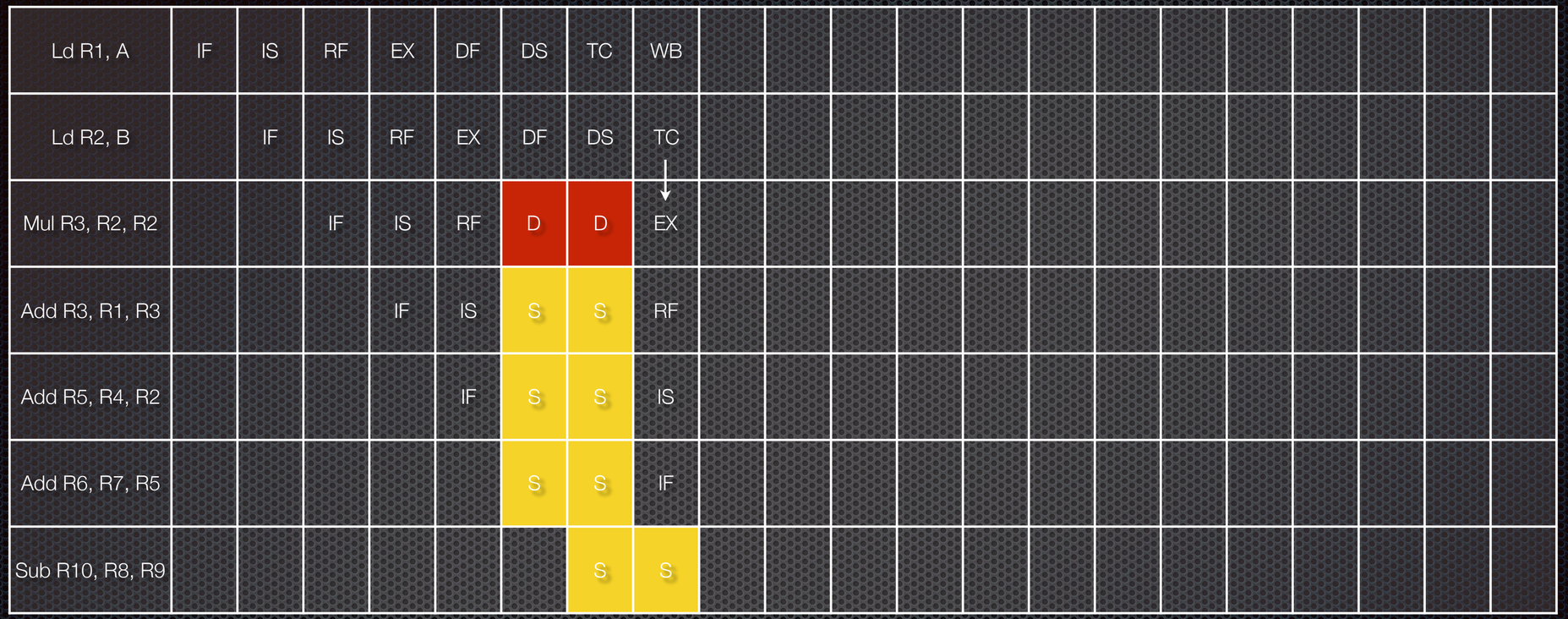
Ld R1, A	IF	IS	RF																
Ld R2, B		IF	IS																
Mul R3, R1, R2			IF																
Add R3, R1, R3																			
Add R5, R4, R2																			
Add R6, R7, R5																			
Sub R10, R8, R9																			

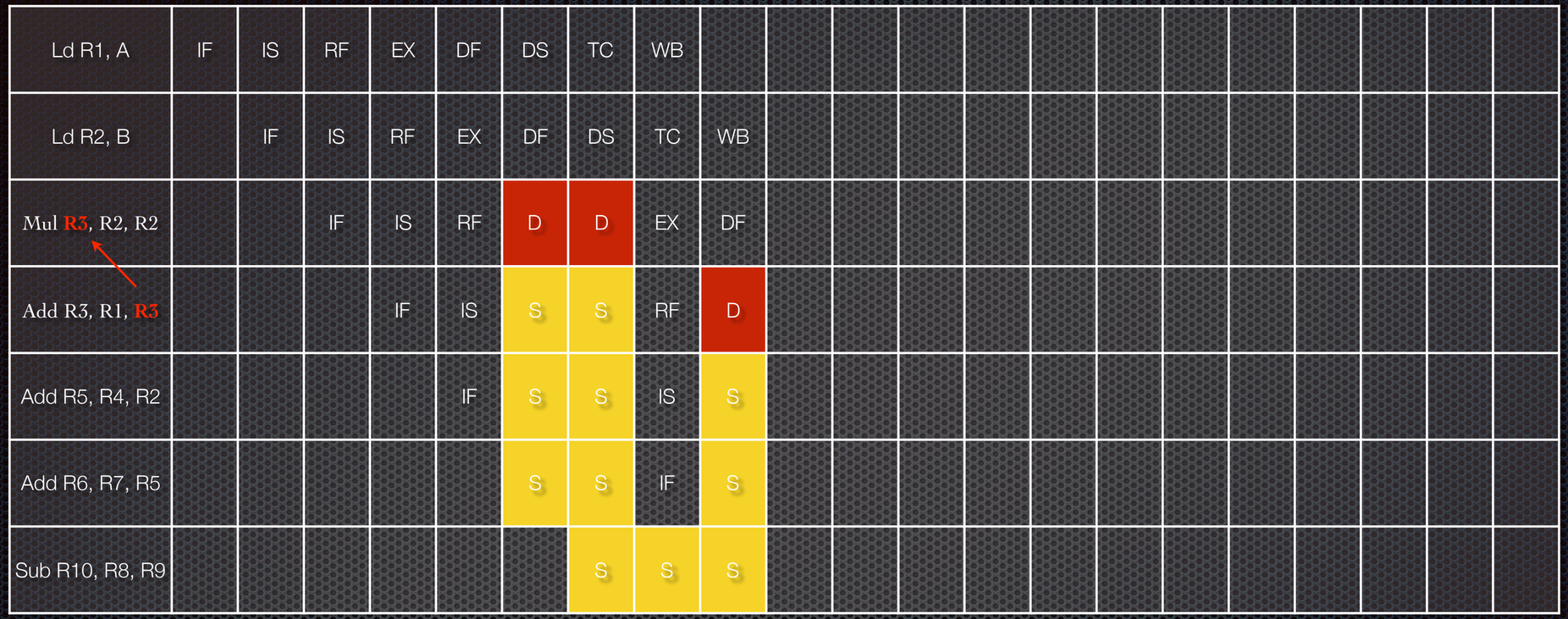
Ld R1, A	IF	IS	RF	EX															
Ld R2, B		IF	IS	RF															
Mul R3, R1, R2			IF	IS															
Add R3, R1, R3				IF															
Add R5, R4, R2																			
Add R6, R7, R5																			
Sub R10, R8, R9																			

Ld R1, A	IF	IS	RF	EX	DF															
Ld R2, B		IF	IS	RF	EX															
Mul R3, R1, R2			IF	IS	RF															
Add R3, R1, R3				IF	IS															
Add R5, R4, R2					IF															
Add R6, R7, R5																				
Sub R10, R8, R9																				


Ld R1, A
IF IS RF EX DF DS
Ld R2, B
IF IS RS RF EX DF
Mul R3, R1, R2
IF IS RF D
Add R3, R1, R3
IF IS S
Add R5, R4, R2
IF S
Add R6, R7, R5
S
Sub R10, R8, R9


Ld R1, A
IF IS RF EX DF DS TC
Ld R2, B
IF IS RF EX DF DS
Mul R3, R1, R2
IF IS RF D D
Add R3, R1, R3
IF IS S S
Add R5, R4, R2
IF S S
Add R6, R7, R5
S S
Sub R10, R8, R9
S


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC
Mul R3, R2, R2
IF IS RF D D EX
Add R3, R1, R3
IF IS S S RF
Add R5, R4, R2
IF S S IS
Add R6, R7, R5
S S IF
Sub R10, R8, R9
S S


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC WB
Mul R3, R2, R2
IF IS RF EX DF D D EX DF
Add R3, R1, R3
IF IS S S RF D
Add R5, R4, R2
IF S S IS S
Add R6, R7, R5
S S IF S
Sub R10, R8, R9
S S S

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB											
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB										
Mul R3, R2, R2			IF	IS	RF	D	D	EX	DF	DS									
Add R3, R1, R3				IF	IS	S	S	RF	D	D									
Add R5, R4, R2					IF	S	S	IS	S	S									
Add R6, R7, R5						S	S	IF	S	S									
Sub R10, R8, R9							S	S	S	S									


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC WB
Mul R3, R2, R2
IF IS RF D D EX DF DS TC
Add R3, R1, R3
IF IS S S RF D D EX
Add R5, R4, R2
IF S S IS S S RF
Add R6, R7, R5
S S IF S S IS
Sub R10, R8, R9
S S S S IF

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB													
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB												
Mul R3, R2, R2			IF	IS	RF	D	D	EX	DF	DS	TC	WB									
Add R3, R1, R3				IF	IS	S	S	RF	D	D	EX	DF									
Add R5, R4, R2					IF	S	S	IS	S	S	RF	EX									
Add R6, R7, R5						S	S	IF	S	S	IS	RF									
Sub R10, R8, R9							S	S	S	S	IF	IS									

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Mul R3, R2, R2			IF	IS	RF	D	D	EX	DF	DS	TC	WB								
Add R3, R1, R3				IF	IS	S	S	RF	D	D	EX	DF	DS							
Add R5, R4, R2					IF	S	S	IS	S	S	RF	EX	DF							
Add R6, R7, R5						S	S	IF	S	S	IS	RF	D							
Sub R10, R8, R9							S	S	S	S	IF	IS	S							


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC WB
Mul R3, R2, R2
IF IS RF EX DF DS TC WB
Add R3, R1, R3
IF IS RF EX DF DS TC
Add R5, R4, R2
IF IS RF EX DF DS
Add R6, R7, R5
IF IS RF EX DF DS
Sub R10, R8, R9
IF IS RF EX DF DS TC WB


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC WB
Mul R3, R2, R2
IF IS RF EX DF DS TC WB
Add R3, R1, R3
IF IS RF EX DF DS TC WB
Add R5, R4, R2
IF IS RF EX DF DS TC
Add R6, R7, R5
IF IS RF EX DF DS TC
Sub R10, R8, R9
IF IS RF EX DF DS TC WB


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC WB
Mul R3, R2, R2
IF IS RF EX DF DS TC WB
Add R3, R1, R3
IF IS RF EX DF DS TC WB
Add R5, R4, R2
IF IS RF EX DF DS TC WB
Add R6, R7, R5
IF IS RF EX DF
Sub R10, R8, R9
IF IS RF EX
D
S

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Mul R3, R2, R2			IF	IS	RF	D	D	EX	DF	DS	TC	WB								
Add R3, R1, R3				IF	IS	S	S	RF	D	D	EX	DF	DS	TC	WB					
Add R5, R4, R2					IF	S	S	IS	S	S	RF	EX	DF	DS	TC	WB				
Add R6, R7, R5						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS			
Sub R10, R8, R9							S	S	S	S	IF	IS	S	S	RF	EX	DF			

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Mul R3, R2, R2			IF	IS	RF	D	D	EX	DF	DS	TC	WB								
Add R3, R1, R3				IF	IS	S	S	RF	D	D	EX	DF	DS	TC	WB					
Add R5, R4, R2					IF	S	S	IS	S	S	RF	EX	DF	DS	TC	WB				
Add R6, R7, R5						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC		
Sub R10, R8, R9							S	S	S	S	IF	IS	S	S	RF	EX	DF	DS		

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Mul R3, R2, R2			IF	IS	RF	D	D	EX	DF	DS	TC	WB								
Add R3, R1, R3				IF	IS	S	S	RF	D	D	EX	DF	DS	TC	WB					
Add R5, R4, R2					IF	S	S	IS	S	S	RF	EX	DF	DS	TC	WB				
Add R6, R7, R5						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB	
Sub R10, R8, R9							S	S	S	S	IF	IS	S	S	RF	EX	DF	DS	TC	

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Mul R3, R2, R2			IF	IS	RF	D	D	EX	DF	DS	TC	WB								
Add R3, R1, R3				IF	IS	S	S	RF	D	D	EX	DF	DS	TC	WB					
Add R5, R4, R2					IF	S	S	IS	S	S	RF	EX	DF	DS	TC	WB				
Add R6, R7, R5						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB	
Sub R10, R8, R9							S	S	S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB

Optimal 14 / Actual 20 = 70% efficient

Reschedule

- Let’s see what happens if we reorder ops
  - Move the Sub up, to follow loads
  - Reverse order of first two Add ops

Ld R1, A	IF																			
Ld R2, B																				
Sub R10, R8, R9																				
Mul R3, R1, R2																				
Add R5, R4, R2																				
Add R3, R1, R3																				
Add R6, R7, R5																				

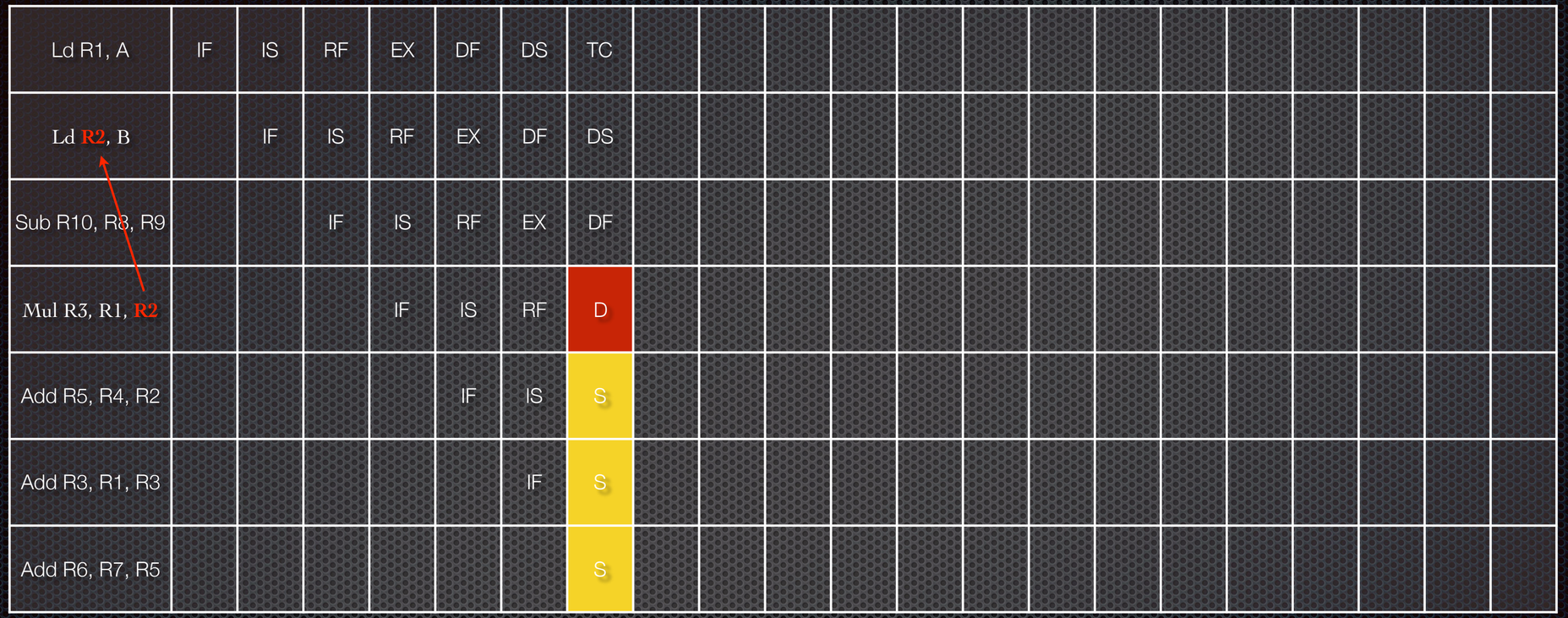
Ld R1, A	IF	IS																		
Ld R2, B		IF																		
Sub R10, R8, R9																				
Mul R3, R1, R2																				
Add R5, R4, R2																				
Add R3, R1, R3																				
Add R6, R7, R5																				

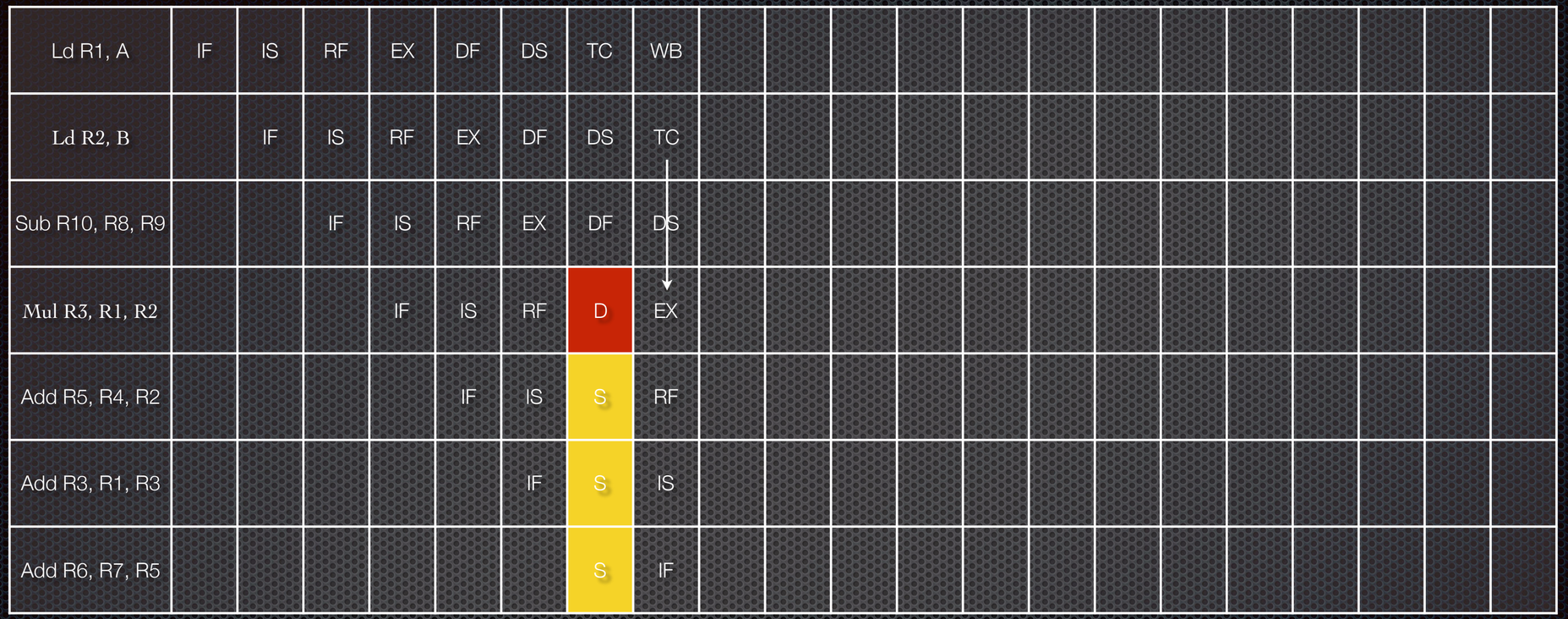
Ld R1, A	IF	IS	RF																
Ld R2, B		IF	IS																
Sub R10, R8, R9			IF																
Mul R3, R1, R2																			
Add R5, R4, R2																			
Add R3, R1, R3																			
Add R6, R7, R5																			

Ld R1, A	IF	IS	RF	EX															
Ld R2, B		IF	IS	RF															
Sub R10, R8, R9			IF	IS															
Mul R3, R1, R2				IF															
Add R5, R4, R2																			
Add R3, R1, R3																			
Add R6, R7, R5																			

Ld R1, A	IF	IS	RF	EX	DF															
Ld R2, B		IF	IS	RF	EX															
Sub R10, R8, R9			IF	IS	RF															
Mul R3, R1, R2				IF	IS															
Add R5, R4, R2					IF															
Add R3, R1, R3																				
Add R6, R7, R5																				

Ld R1, A	IF	IS	RF	EX	DF	DS														
Ld R2, B		IF	IS	RF	EX	DF														
Sub R10, R8, R9			IF	IS	RF	EX														
Mul R3, R1, R2				IF	IS	RF														
Add R5, R4, R2					IF	IS														
Add R3, R1, R3						IF														
Add R6, R7, R5																				


Ld R1, A
IF IS RF EX DF DS TC
Ld R2, B
IF IS RF EX DF DS
Sub R10, R8, R9
IF IS RF EX DF
Mul R3, R1, R2
IF IS RF D
Add R5, R4, R2
IF IS S
Add R3, R1, R3
IF S
Add R6, R7, R5
S


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC
Sub R10, R8, R9
IF IS RF EX DF DS
Mul R3, R1, R2
IF IS RF D EX
Add R5, R4, R2
IF IS S RF
Add R3, R1, R3
IF S IS
Add R6, R7, R5
S IF

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Sub R10, R8, R9			IF	IS	RF	EX	DF	DS	TC											
Mul R3, R1, R2				IF	IS	RF	D	EX	DF											
Add R5, R4, R2					IF	IS	S	RF	EX											
Add R3, R1, R3						IF	S	IS	RF											
Add R6, R7, R5							S	IF	IS											


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC WB
Sub R10, R8, R9
IF IS RF EX DF DS TC WB
Mul R3, R1, R2
IF IS RF EX DF D EX DF DS
Add R5, R4, R2
IF IS S RF EX DF
Add R3, R1, R3
IF S IS RF D
Add R6, R7, R5
S IF IS S


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC WB
Sub R10, R8, R9
IF IS RF EX DF DS TC WB
Mul R3, R1, R2
IF IS RF EX DF DS TC
D
Add R5, R4, R2
IF IS RF EX DF DS
S
Add R3, R1, R3
IF IS RF EX
D
S
Add R6, R7, R5
IF IS RF
S
S


Ld R1, A
IF IS RF EX DF DS TC WB
Ld R2, B
IF IS RF EX DF DS TC WB
Sub R10, R8, R9
IF IS RF EX DF DS TC WB
Mul R3, R1, R2
IF IS RF D EX DF DS TC WB
Add R5, R4, R2
IF IS S RF EX DF DS TC
Add R3, R1, R3
IF S IS RF D EX DF
Add R6, R7, R5
S IF IS S RF EX

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Sub R10, R8, R9			IF	IS	RF	EX	DF	DS	TC	WB										
Mul R3, R1, R2				IF	IS	RF	D	EX	DF	DS	TC	WB								
Add R5, R4, R2					IF	IS	S	RF	EX	DF	DS	TC	WB							
Add R3, R1, R3						IF	S	IS	RF	D	EX	DF	DS							
Add R6, R7, R5							S	IF	IS	S	RF	EX	DF							

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Sub R10, R8, R9			IF	IS	RF	EX	DF	DS	TC	WB										
Mul R3, R1, R2				IF	IS	RF	D	EX	DF	DS	TC	WB								
Add R5, R4, R2					IF	IS	S	RF	EX	DF	DS	TC	WB							
Add R3, R1, R3						IF	S	IS	RF	D	EX	DF	DS	TC						
Add R6, R7, R5							S	IF	IS	S	RF	EX	DF	DS						

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Sub R10, R8, R9			IF	IS	RF	EX	DF	DS	TC	WB										
Mul R3, R1, R2				IF	IS	RF	D	EX	DF	DS	TC	WB								
Add R5, R4, R2					IF	IS	S	RF	EX	DF	DS	TC	WB							
Add R3, R1, R3						IF	S	IS	RF	D	EX	DF	DS	TC	WB					
Add R6, R7, R5							S	IF	IS	S	RF	EX	DF	DS	TC					

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB												
Ld R2, B		IF	IS	RF	EX	DF	DS	TC	WB											
Sub R10, R8, R9			IF	IS	RF	EX	DF	DS	TC	WB										
Mul R3, R1, R2				IF	IS	RF	D	EX	DF	DS	TC	WB								
Add R5, R4, R2					IF	IS	S	RF	EX	DF	DS	TC	WB							
Add R3, R1, R3						IF	S	IS	RF	D	EX	DF	DS	TC	WB					
Add R6, R7, R5							S	IF	IS	S	RF	EX	DF	DS	TC	WB				

Optimal 14 / Actual 16 = 88% efficient

Conclusion

- Instruction rescheduling significantly boosts efficiency
- NO HARDWARE CHANGE REQUIRED!
- Fairly simple compiler analysis
- Only possible for separable operations

Exercise: Determine the efficiency of the following, and then reschedule the instructions to improve the efficiency

Ld R1, A											
Add R5, R4, R1											
Ld R2, B											
Add R7, R6, R2											
Ld R3, C											
Add R8, R6, R3											
Add R9, R7, R8											
Add R10, R5, R2											

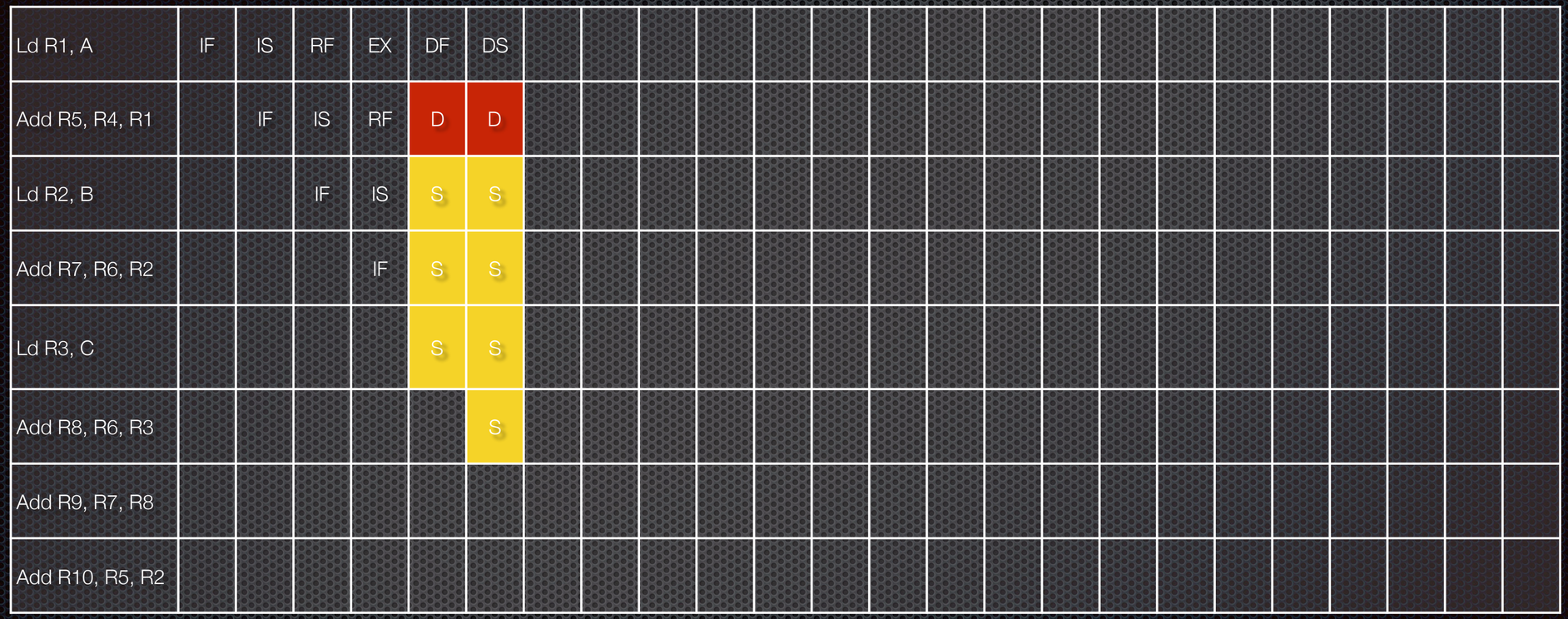
Ld R1, A	IF																						
Add R5, R4, R1																							
Ld R2, B																							
Add R7, R6, R2																							
Ld R3, C																							
Add R8, R6, R3																							
Add R9, R7, R8																							
Add R10, R5, R2																							

Ld R1, A	IF	IS																					
Add R5, R4, R1		IF																					
Ld R2, B																							
Add R7, R6, R2																							
Ld R3, C																							
Add R8, R6, R3																							
Add R9, R7, R8																							
Add R10, R5, R2																							

Ld R1, A	IF	IS	RF																				
Add R5, R4, R1		IF	IS																				
Ld R2, B			IF																				
Add R7, R6, R2																							
Ld R3, C																							
Add R8, R6, R3																							
Add R9, R7, R8																							
Add R10, R5, R2																							

Ld R1, A	IF	IS	RF	EX																			
Add R5, R4, R1		IF	IS	RF																			
Ld R2, B			IF	IS																			
Add R7, R6, R2				IF																			
Ld R3, C																							
Add R8, R6, R3																							
Add R9, R7, R8																							
Add R10, R5, R2																							

Ld R1, A	IF	IS	RF	EX	DF																		
Add R5, R4, R1		IF	IS	RF	D																		
Ld R2, B			IF	IS	S																		
Add R7, R6, R2				IF	S																		
Ld R3, C					S																		
Add R8, R6, R3																							
Add R9, R7, R8																							
Add R10, R5, R2																							


Ld R1, A
IF IS RF EX DF DS
Add R5, R4, R1
IF IS RF D D
Ld R2, B
IF IS S S
Add R7, R6, R2
IF S S
Ld R3, C
S S
Add R8, R6, R3
S
Add R9, R7, R8
Add R10, R5, R2

Ld R1, A	IF	IS	RF	EX	DF	DS	TC																
Add R5, R4, R1		IF	IS	RF	D	D	EX																
Ld R2, B			IF	IS	S	S	RF																
Add R7, R6, R2				IF	S	S	IS																
Ld R3, C					S	S	IF																
Add R8, R6, R3						S	S																
Add R9, R7, R8							S																
Add R10, R5, R2																							

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB															
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF															
Ld R2, B			IF	IS	S	S	RF	EX															
Add R7, R6, R2				IF	S	S	IS	RF															
Ld R3, C					S	S	IF	IS															
Add R8, R6, R3						S	S	IF															
Add R9, R7, R8							S	S															
Add R10, R5, R2								S															

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB															
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS														
Ld R2, B			IF	IS	S	S	RF	EX	DF														
Add R7, R6, R2				IF	S	S	IS	RF	D														
Ld R3, C					S	S	IF	IS	S														
Add R8, R6, R3						S	S	IF	S														
Add R9, R7, R8							S	S	S														
Add R10, R5, R2								S	S														

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB															
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS													
Add R7, R6, R2				IF	S	S	IS	RF	D	D													
Ld R3, C					S	S	IF	IS	S	S													
Add R8, R6, R3						S	S	IF	S	S													
Add R9, R7, R8							S	S	S	S													
Add R10, R5, R2								S	S	S													

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB															
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB												
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX												
Ld R3, C					S	S	IF	IS	S	S	RF												
Add R8, R6, R3						S	S	IF	S	S	IS												
Add R9, R7, R8							S	S	S	S	IF												
Add R10, R5, R2								S	S	S	S												

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF												
Ld R3, C					S	S	IF	IS	S	S	RF	EX												
Add R8, R6, R3						S	S	IF	S	S	IS	RF												
Add R9, R7, R8							S	S	S	S	IF	IS												
Add R10, R5, R2								S	S	S	S	IF												

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS											
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF											
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D											
Add R9, R7, R8							S	S	S	S	IF	IS	S											
Add R10, R5, R2								S	S	S	S	IF	S											

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC										
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS										
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D										
Add R9, R7, R8							S	S	S	S	IF	IS	S	S										
Add R10, R5, R2								S	S	S	S	IF	S	S										

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC									
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX									
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF									
Add R10, R5, R2								S	S	S	S	IF	S	S	IS									

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB								
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX	DF								
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF	D								
Add R10, R5, R2								S	S	S	S	IF	S	S	IS	S								

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB								
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS							
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF	D	D							
Add R10, R5, R2								S	S	S	S	IF	S	S	IS	S	S							

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB								
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC						
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF	D	D	EX						
Add R10, R5, R2								S	S	S	S	IF	S	S	IS	S	S	RF						

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB								
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB					
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF	D	D	EX	DF					
Add R10, R5, R2								S	S	S	S	IF	S	S	IS	S	S	RF	EX					

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB								
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB					
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF	D	D	EX	DF	DS				
Add R10, R5, R2								S	S	S	S	IF	S	S	IS	S	S	RF	EX	DF				

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB								
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB					
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF	D	D	EX	DF	DS	TC			
Add R10, R5, R2								S	S	S	S	IF	S	S	IS	S	S	RF	EX	DF	DS			

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB								
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB					
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF	D	D	EX	DF	DS	TC	WB		
Add R10, R5, R2								S	S	S	S	IF	S	S	IS	S	S	RF	EX	DF	DS	TC		

Ld R1, A	IF	IS	RF	EX	DF	DS	TC	WB																
Add R5, R4, R1		IF	IS	RF	D	D	EX	DF	DS	TC	WB													
Ld R2, B			IF	IS	S	S	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2				IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB									
Ld R3, C					S	S	IF	IS	S	S	RF	EX	DF	DS	TC	WB								
Add R8, R6, R3						S	S	IF	S	S	IS	RF	D	D	EX	DF	DS	TC	WB					
Add R9, R7, R8							S	S	S	S	IF	IS	S	S	RF	D	D	EX	DF	DS	TC	WB		
Add R10, R5, R2								S	S	S	S	IF	S	S	IS	S	S	RF	EX	DF	DS	TC	WB	

Optimal 15 / Actual 23 = 65% efficient

Rescheduling

Ld R3, C	IF	IS	RF	EX	DF	DS	TC	WB																
Ld R1, A		IF	IS	RF	EX	DF	DS	TC	WB															
Ld R2, B			IF	IS	RF	EX	DF	DS	TC	WB														
Add R8, R6, R3				IF	IS	RF	EX	DF	DS	TC	WB													
Add R5, R4, R1					IF	IS	RF	EX	DF	DS	TC	WB												
Add R7, R6, R2						IF	IS	RF	EX	DF	DS	TC	WB											
Add R10, R5, R2							IF	IS	RF	D	EX	DF	DS	TC	WB									
Add R9, R7, R8								IF	IS	S	RF	EX	DF	DS	TC	WB								

Optimal 15 / Actual 16 = 94% efficient

Scoreboarding

Managing Dependences for Multiple Functional Units

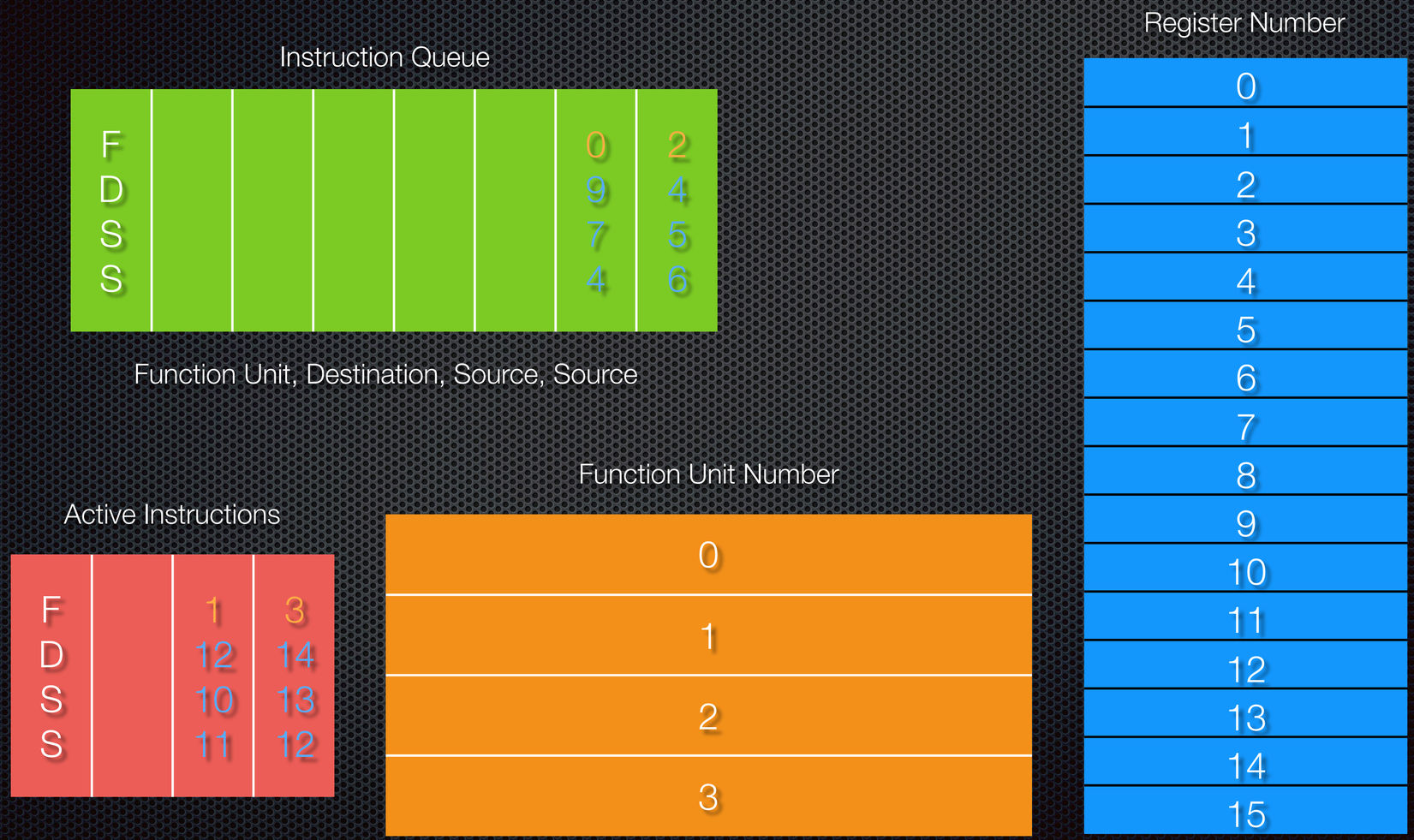
Multiple Functional Units

- Provide parallelism
- Multiple instructions execute simultaneously
- Potential for conflicts on operands
- Seymour Cray design for CDC-6600

Operand Dependences

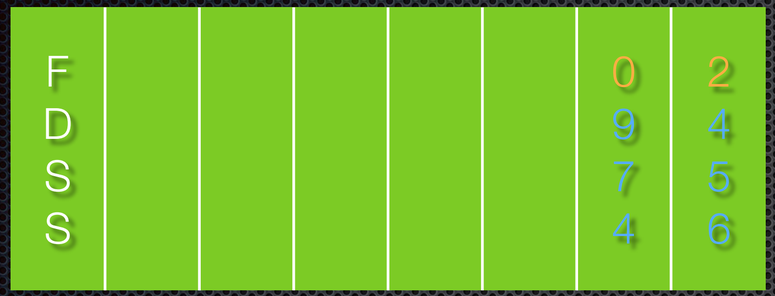
- RAW: Read After Write
  - Need to read (use) a value that has not yet been written
- WAW: Write After Write
  - Writes occur out of order because units vary in timing
- WAR: Write After Read
  - Need to read a value before it gets overwritten

Instructions, Functional Units, and Registers

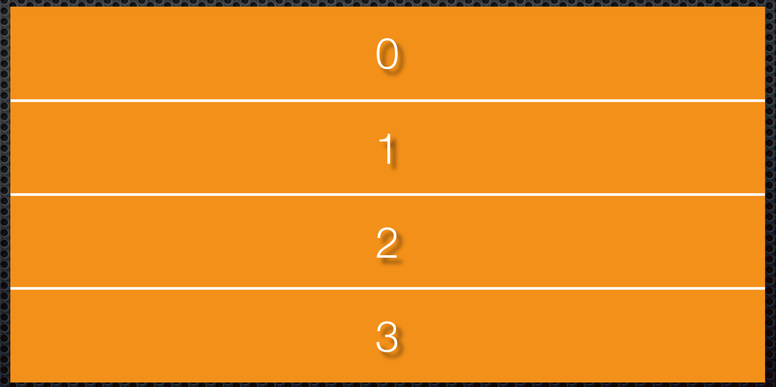


Result Register Designator

Specifies register is reserved by a given unit





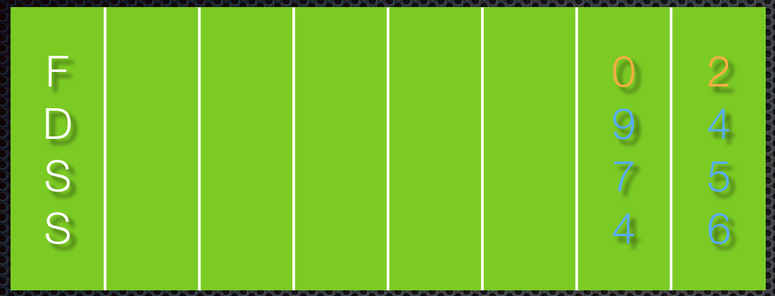


RRD

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	1
13	
14	3
15	

Entry-Operand Register Designator

Specifies operand sources and status for a unit: Valid/Pending, if Pending source unit is given





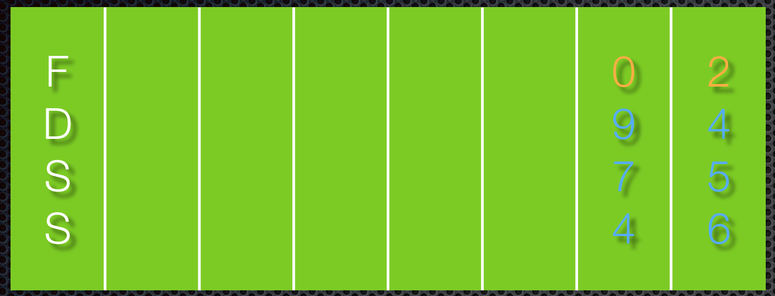
EORD

0						
1	10	V		11	V	
2						
3	13	V		12	P	1

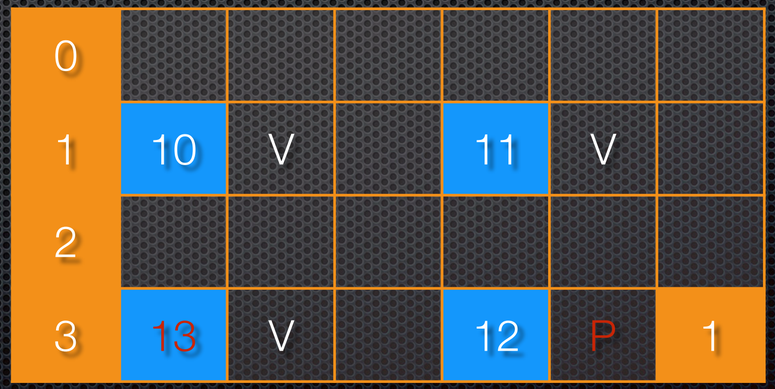
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	1
13	
14	3
15	

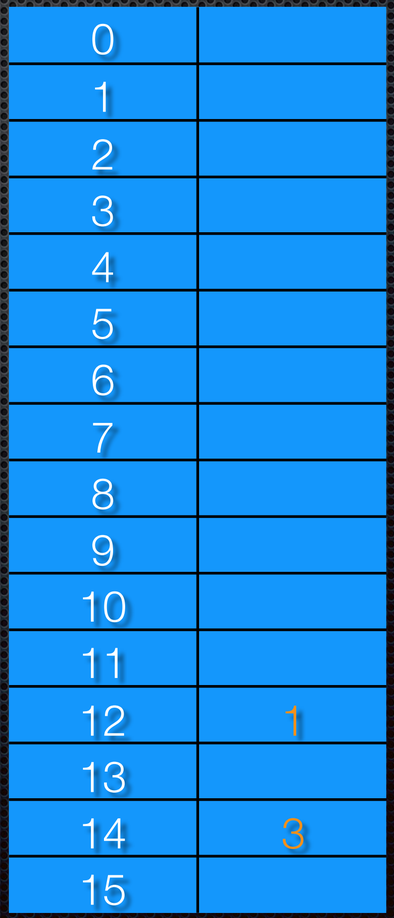
Issue Rule

An instruction can issue from the queue when its functional unit is available, and its result register is not reserved by an active instruction. This prevents WAW hazards: it's impossible for two active instructions to be pending with the same destination register.



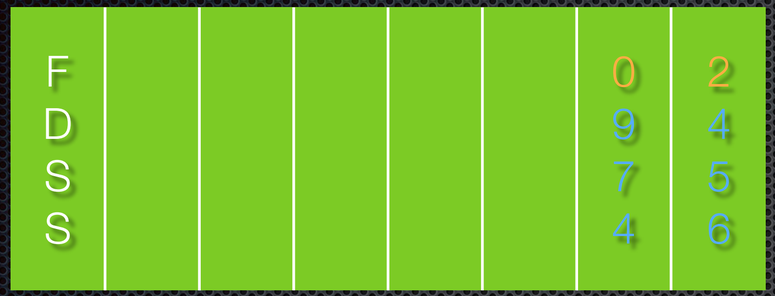




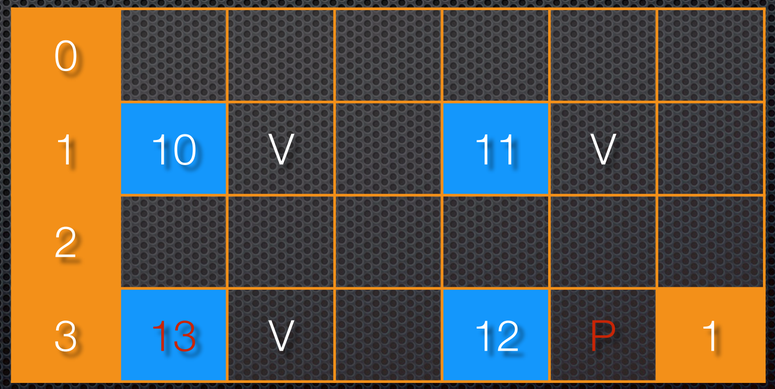


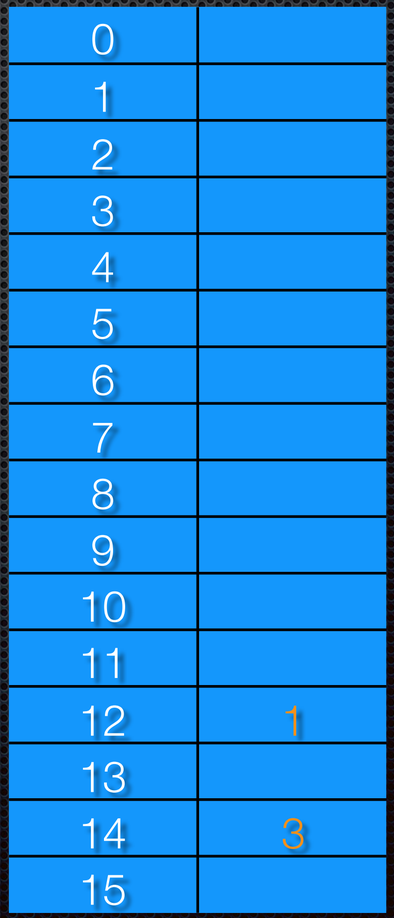
Release Rule

Operands are only released to a functional unit when all of them are valid, preventing RAW hazards.
Release can occur in parallel with register writing, via forwarding.



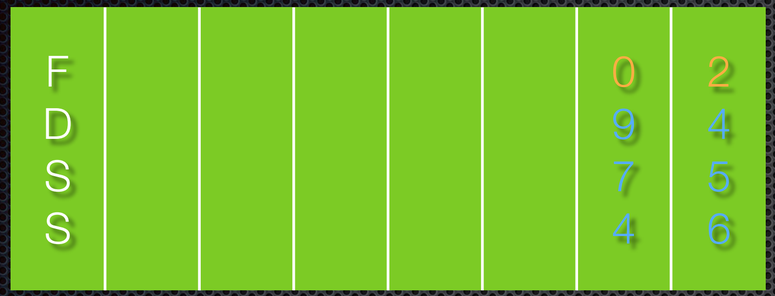


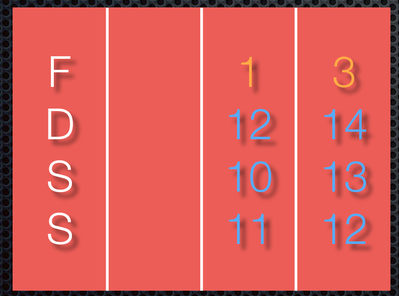


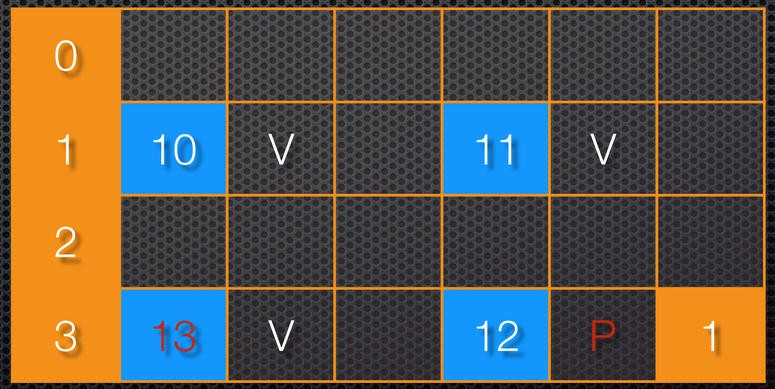


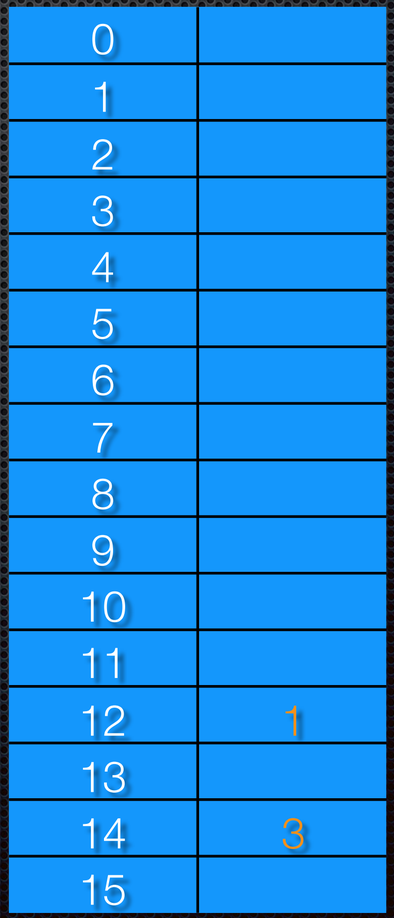
Stall Rule

An instruction will be stalled if its destination is listed as an entry operand for any other pending instruction. This prevents WAR hazards: it's not allowed to write until the value in its destination has been released to all dependent instructions.



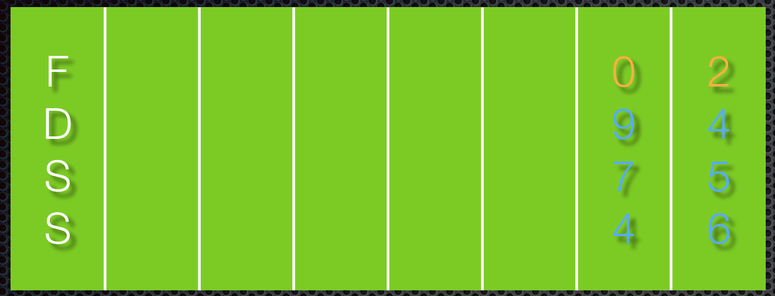




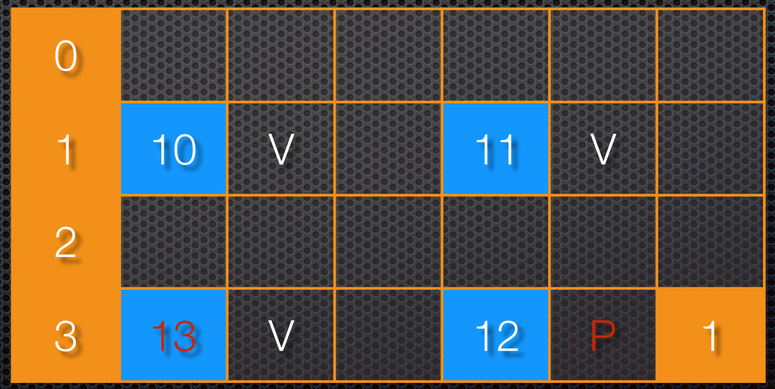


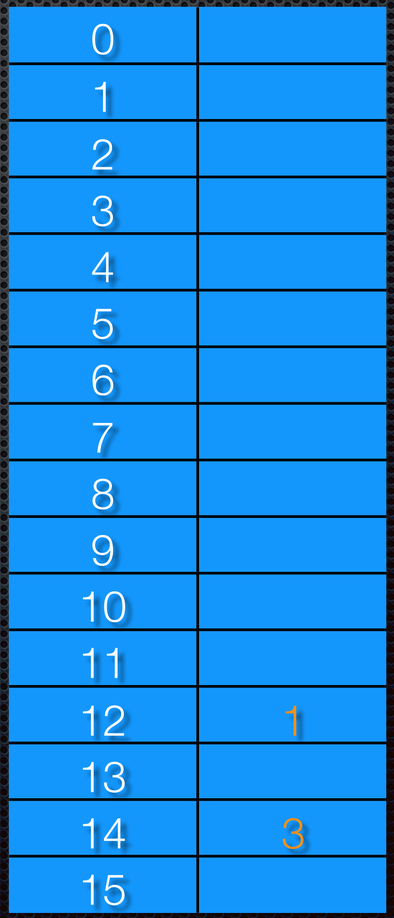
Current status

Instruction in F1 has valid operands. Instruction in F3 is stalled waiting for result from F1 (RAW hazard), and the operand in R13 has not been released to it. The next instruction is free to issue.



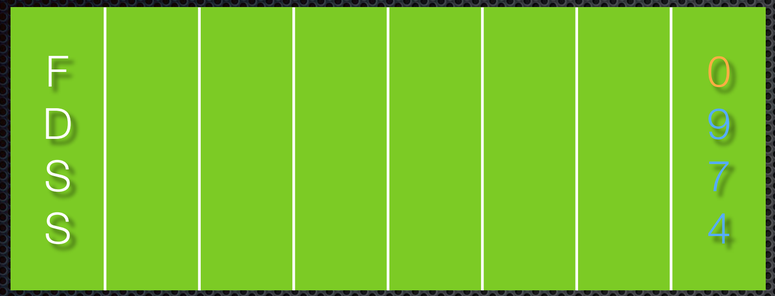






Cycle 2

Result from F1 is valid, and forwarded to F3 while being written in R12. R13 is also released to F3. F1 is busy until result is written. Instruction issues to F2, reserving R4 for result. Valid operands in R5 and R6 are released to F2.



0	
1	
2	
3	
4	2
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	3
15	

F	2	1	3
D	4	12	14
S	5	10	13
S	6	11	12

0						
1	10	V		11	V	
2	5	V		6	V	
3	13	V		12	V	

Cycle 3

New instructions arrive. F1 instruction completes. Result from F3 is valid, and being written in R14. Instruction in F2 is executing. Instruction in queue issues to F0 because F0 and R9 are available, but must wait for result from F2 (RAW hazard).

F					2	3	1
D					13	7	9
S					10	13	13
S					8	12	12

F	0	2	3
D	9	4	14
S	7	5	13
S	4	6	12

0	7	V		4	P	2
1						
2	5	V		6	V	
3	13	V		12	V	

0	
1	
2	
3	
4	2
5	
6	
7	
8	
9	0
10	
11	
12	
13	
14	
15	

Cycle 4

F3 instruction completes. Result from F2 is valid, and being written in R4 and forwarded to F0.
Operands are released to instruction in F0.
Instruction in queue cannot issue to F1 because F0 has the same destination reserved (WAW hazard).

F					2	3	1
D					13	7	9
S					10	13	13
S					8	12	12

F		0	2
D		9	4
S		7	5
S		4	6

0	7	V		4	P	2
1						
2	5	V		6	V	
3						

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	0
10	
11	
12	
13	
14	
15	

Cycle 5

F2 instruction completes. Result from F0 is valid, and being written in R9. Instruction in queue can issue to F1 and receive its operands because F0 is writing in 9 and will finish before F1 starts.

F						2	3
D						13	7
S						10	13
S						8	12

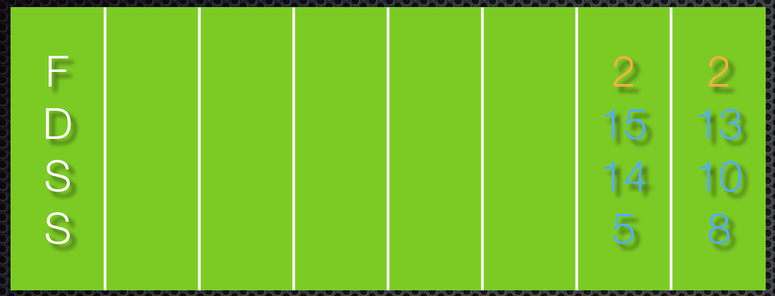
F		1	0
D		9	9
S		13	7
S		12	4

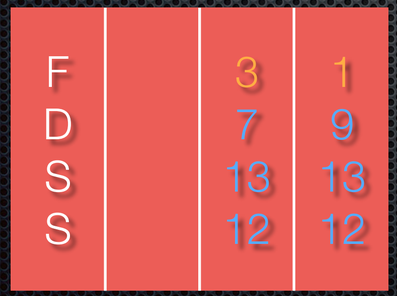
0	7	V		4	V	
1	13	V		12	V	
2						
3						

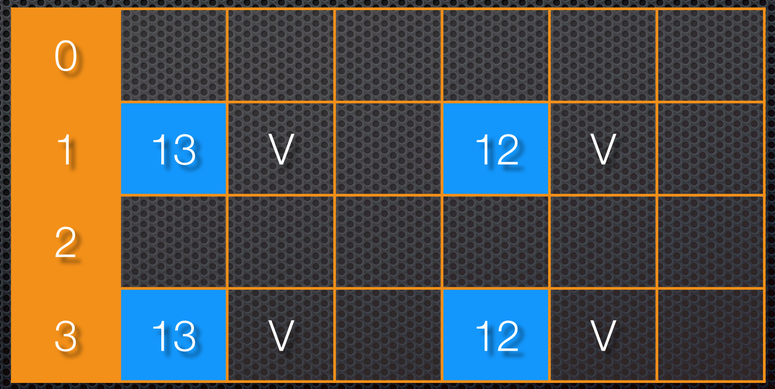
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	1
10	
11	
12	
13	
14	
15	

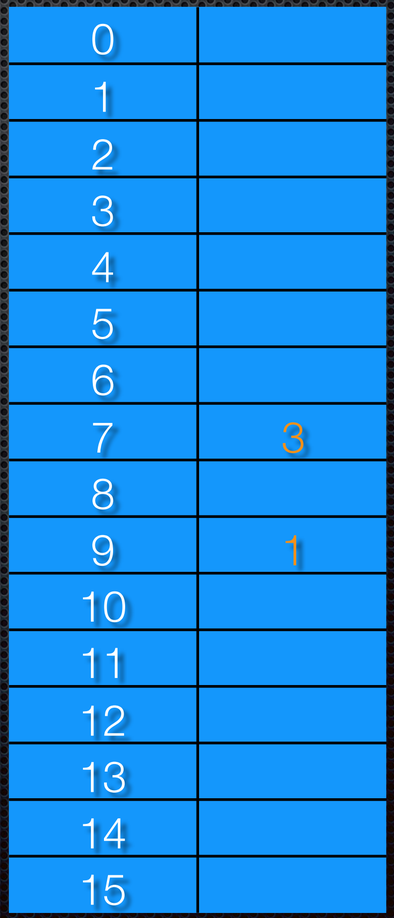
Cycle 6

F0 instruction completes. Instruction in queue can issue to F3 and receive its operands because F3 and R7 are available. It's fine to read the same operands as F0 (RAR isn't a hazard).



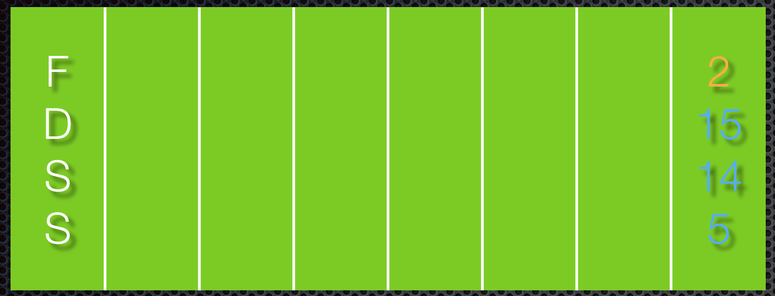






Cycle 7

F1 instruction is writing its result to R9.
Instruction in queue issues and receives operands because F2 and R13 are available and no writes are pending to R10 or R8.
F2 stalls because its destination is an entry operand for F3. (WAR Hazard)



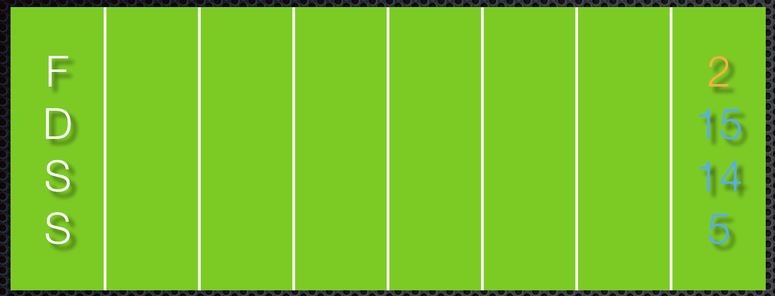
F	2	3	1
D	13	7	9
S	10	13	13
S	8	12	12

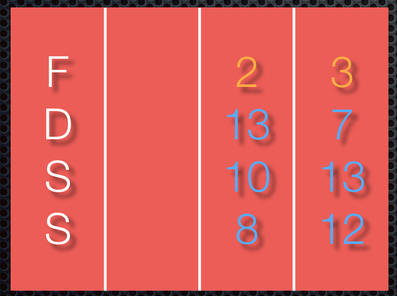
0						
1	13	V		12	V	
2	10	V		8	V	
3	13	V		12	V	

0	
1	
2	
3	
4	
5	
6	
7	3
8	
9	
10	
11	
12	
13	2
14	
15	

Cycle 8

F1 instruction completes. F3 is writing to R7. F2 starts execution.
Instruction in queue cannot issue because F2 is busy (structural hazard).



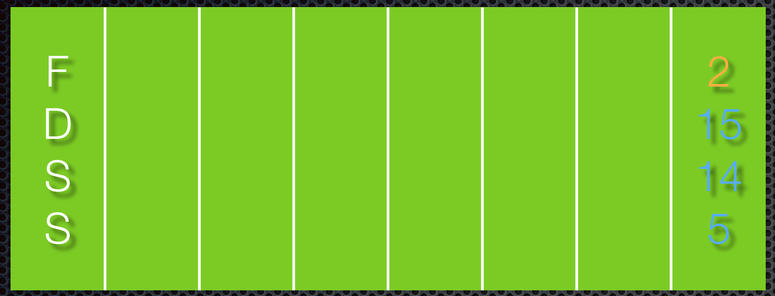


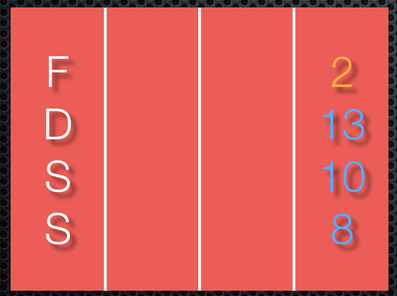
0						
1						
2	10	V		8	V	
3	13	V		12	V	

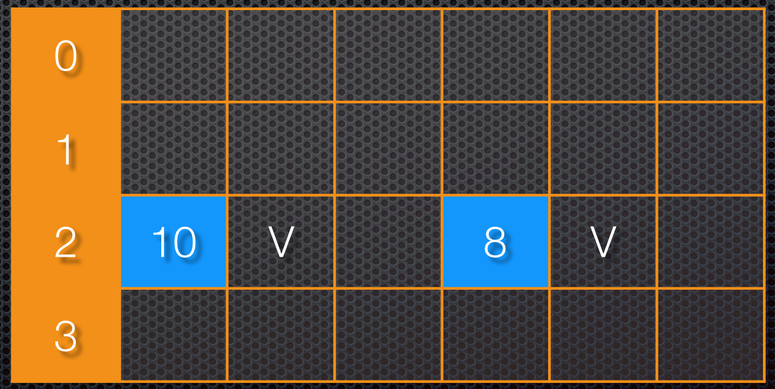
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	2
14	
15	

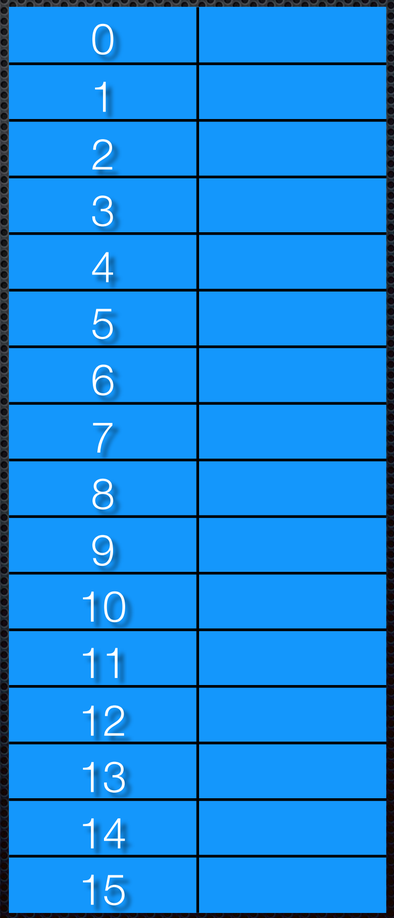
Cycle 9

F3 instruction completes. F2 is writing to R13.
Instruction in queue cannot issue because F2 is busy (structural hazard).









Summary

- Scoreboard avoids dependence hazards
- Centralized control of issue/release/stall
- Overly restrictive
- Simple to implement

Reservation Stations

Tomasulo's Algorithm for Managing Dependence Among Multiple Functional Units

Reservation Station

- One or more associated with each functional unit
- Contains:
  - Operation
  - Operand values, if already computed
  - Operand source stations, if not yet available
  - Operand and reservation station status
- Robert Tomasulo design for IBM 360 Model 91

Registers

- Extended with result designator, indicating reservation station that will write to it
- Keeps track of most recently issued instruction that targets this register
- Unlike scoreboard, this can change before the register is updated
- Issuing instructions check the registers to see where each operand is coming from

Station 5 has operation X with valid values, executing on unit 2

Instruction Queue

OP						Z	X
D						9	4
S						7	5
S						4	6

Reservation Stations

1	OP	V1	V2	S1	S2
2						
3						
4						
5	X	3	17	0	0
6						
7						
8						

Function Units

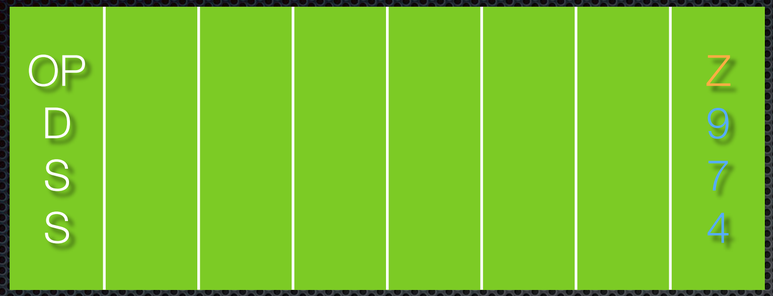
0
1
2
3

Common Data Bus

Reg#	Desig	Value
0	0	14
1	0	25
2	0	84
3	0	63
4	0	22
5	5	
6	0	97
7	0	45
8	0	48
9	0	18
10	0	27
11	0	83
12	0	16
13	0	47
14	0	69
15	0	51

Instruction in queue issues to station 6 with invalid V1 value, and copy of R5 designator for station 5. V2 is a valid copy of R6.

Instruction Queue

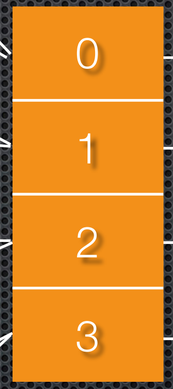


Copying value avoids WAW and WAR hazards. R4 now designates its value is pending from station 6.

Reservation Stations

	OP	V1	V2	S1	S2
1					
2					
3					
4					
5	X	3	17	0	0
6	X	0	97	5	0
7					
8					

Function Units

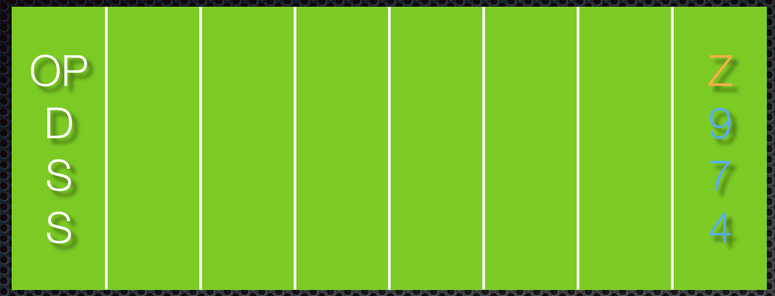


Common Data Bus

Reg#	Desig	Value
0	0	14
1	0	25
2	0	84
3	0	63
4	6	22
5	5	99
6	0	97
7	0	45
8	0	48
9	0	18
10	0	27
11	0	83
12	0	16
13	0	47
14	0	69
15	0	51

Function unit 2 produces a result for station 5 and broadcasts it on the CDB, where the station number will match the S1 designator in station 6

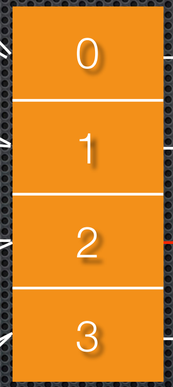
Instruction Queue



Reservation Stations

	OP	V1	V2	S1	S2
1					
2					
3					
4					
5	X	3	17	0	0
6	X	0	97	5	0
7					
8					

Function Units



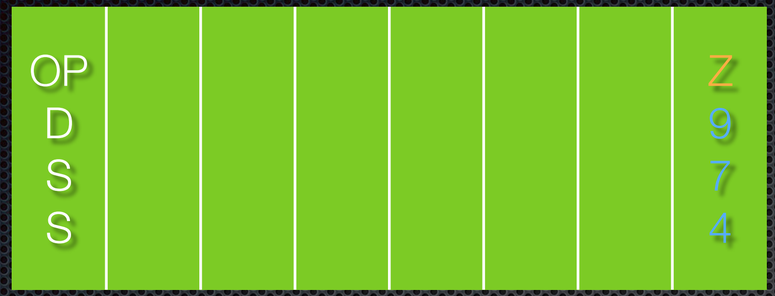
5

Common Data Bus

Reg#	Desig	Value
0	0	14
1	0	25
2	0	84
3	0	63
4	6	22
5	5	99
6	0	97
7	0	45
8	0	48
9	0	18
10	0	27
11	0	83
12	0	16
13	0	47
14	0	69
15	0	51

Station 6 snoops the value, and copies it, clearing the S1 designator. It is now ready to execute. This avoids RAW hazards.

Instruction Queue

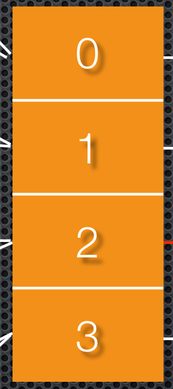


Register 5 now has a valid result, and a 0 designator

Reservation Stations

	OP	V1	V2	S1	S2
1					
2					
3					
4					
5	X	3	17	0	0
6	X	77	97	0	0
7					
8					

Function Units



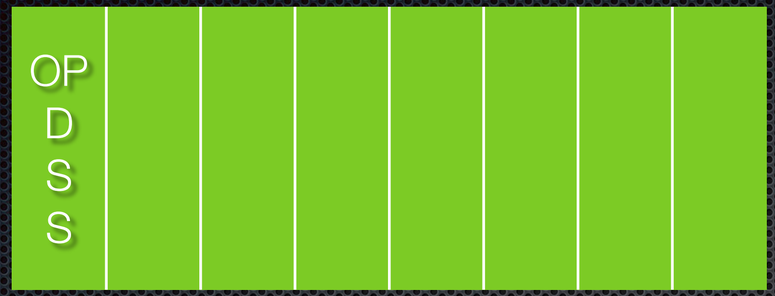
5

Common Data Bus

Reg#	Desig	Value
0	0	14
1	0	25
2	0	84
3	0	63
4	6	22
5	0	77
6	0	97
7	0	45
8	0	48
9	0	18
10	0	27
11	0	83
12	0	16
13	0	47
14	0	69
15	0	51

Instruction in queue issues to station 3, copying value from R7, and designator from R4 that result will come from station 6

Instruction Queue

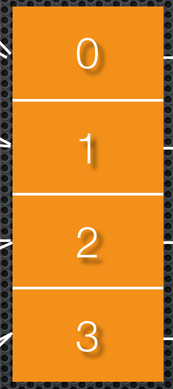


Register 9 designates that 3 will write it.

Reservation Stations

	OP	V1	V2	S1	S2
1					
2					
3	Z	45	0	0	6
4					
5					
6	X	77	97	0	0
7					
8					

Function Units



Common Data Bus

Reg#	Desig	Value
0	0	14
1	0	25
2	0	84
3	0	63
4	6	22
5	0	77
6	0	97
7	0	45
8	0	48
9	3	18
10	0	27
11	0	83
12	0	16
13	0	47
14	0	69
15	0	51

Register Renaming

- Reservation stations act as extra registers
- If a register has a designator other than 0 that differs from the station that is about to write it, the write is cancelled
  - An instruction is pending that will overwrite it
  - All consumers of this value already have copies
- Result designator acts as a pointer to the “station register” that has taken this logical register’s name

Comparison with Scoreboard

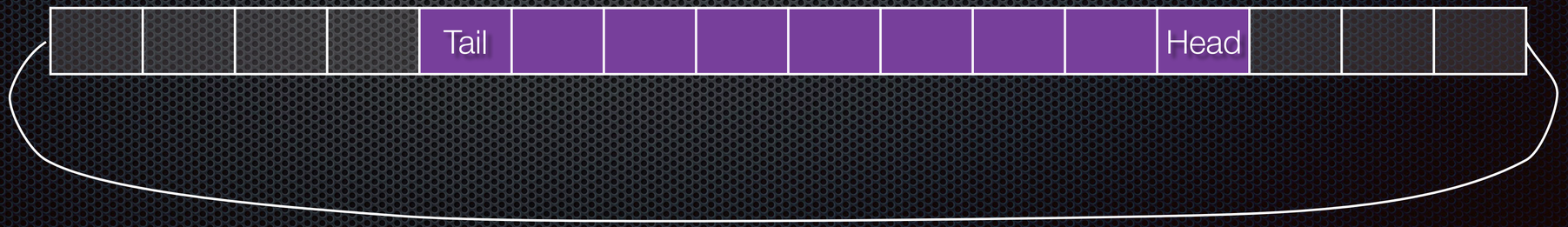
- Distributed (vs. centralized)
- Allows out-of-order execution
- Allows more instructions to proceed
- More complex to implement
- Adds a one cycle delay for forwarding
- Neither can execute past a branch

Sohi IEEE TC 1990

Instruction Issue Logic for High Performance Interruptible, Multiple Functional Unit, Pipelined Computers

Register Update Unit

- Similar to reservation stations, but arranged in a queue
- Head points to next instruction to commit
- Tail is next available issue slot
- If Head = Tail, then RUU is full and no issue allowed



RUU Issue

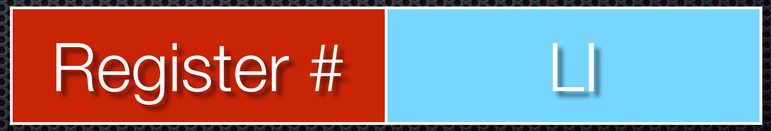
- Instruction goes into current tail pointer station
- Tail pointer is updated
- If source register values are available, copied to station
- If value(s) not available, station gets tag(s)

RUU Register Extensions

- Registers can have multiple instances
- Queue structure keeps instances in order
- Registers extended with Number of Instances and Last Instance fields (NI, LI)
- When an instruction with destination R_i issues, NI_i and LI_i are incremented, LI is modulo n, NI is max 2^n-1
- When NI is max, issue is blocked
- Committing instruction with R_i destination decrements NI_i

RUU Tag

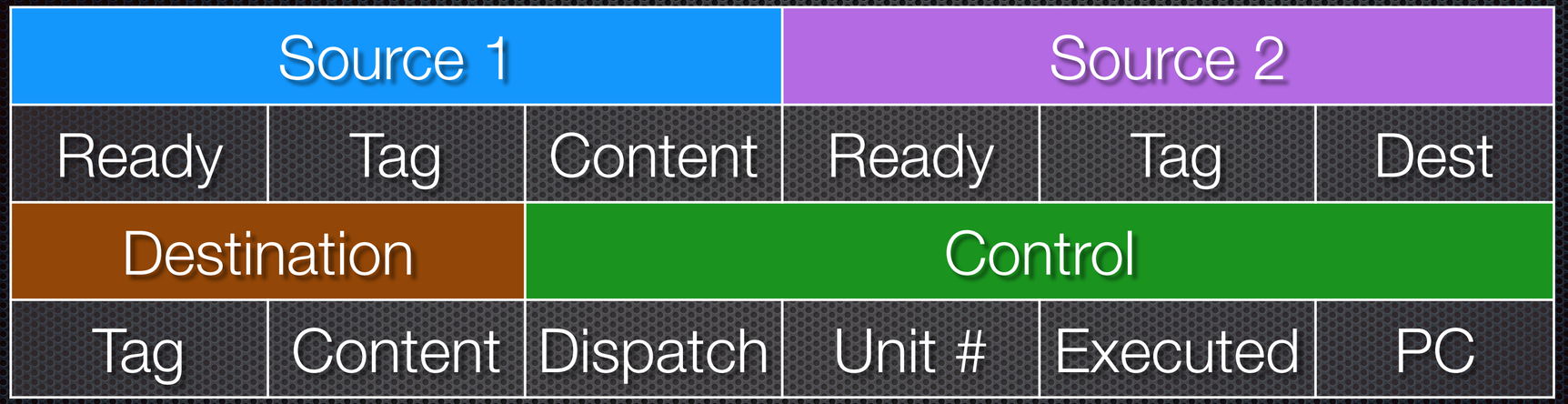
- Tags in the RUU are R_i concatenated with LI_i
- A tag thus indicates the most recent destination instance for the register
- Up to 7 instances of a register are allowed in this implementation



RUU Entry Source Fields

- Ready bit
- Tag subfield
- Content subfield
- If not ready, then watch result bus for matching tag
- When tag is seen, copy result bus into content subfield and switch to Ready

RUU Entry Structure



Determining Issue

- When operands are all ready, an instruction can be issued to a functional unit
- Loads and stores get priority (other schemes possible)
- Others issued in order received

Commit

- When Executed is set, instruction is done
- Results go to register file
- NI for destination register is decremented
- Head pointer is updated

Results

Number of Entries in RUU	Relative Speedup	Instruction Issue Rate
3	0.853	0.380
4	0.940	0.419
6	1.079	0.481
8	1.248	0.557
10	1.383	0.617
12	1.508	0.673
15	1.584	0.706
20	1.649	0.735
25	1.682	0.750
30	1.700	0.758
40	1.735	0.774
50	1.737	0.775

With bypass

Number of Entries in RUU	Relative Speedup	Instruction Issue Rate
3	0.824	0.366
4	0.912	0.407
6	1.031	0.460
8	1.082	0.483
10	1.103	0.492
12	1.216	0.542
15	1.225	0.546
20	1.295	0.578
25	1.330	0.593
30	1.393	0.621
40	1.439	0.642
50	1.471	0.656

Without bypass

Discussion

Sleiman 16

Efficiently Scaling Out-of-Order Cores for Simultaneous Multithreading

Threading Reduces OOO

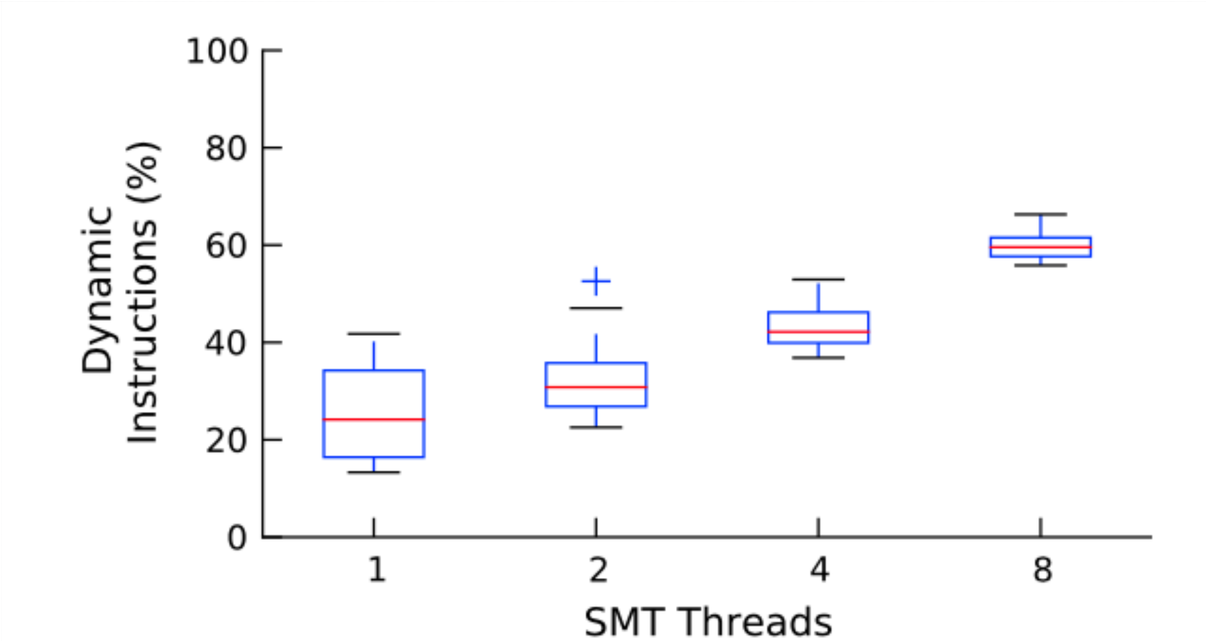


Figure 1: Fraction of instructions wasting OOO resources.

With more threads, dependent instructions end up naturally separated, reducing OOO issue

In-Order (INO) Dependences

- Data (WAR, WAW, RAW)
- Speculation beyond a branch
- Structural (resource limitations)

Out of Order Resources

- Reorder buffer
- Issue queue
- Load/store queue
- Physical register file
- Expensive hardware and energy

Shelf for INO

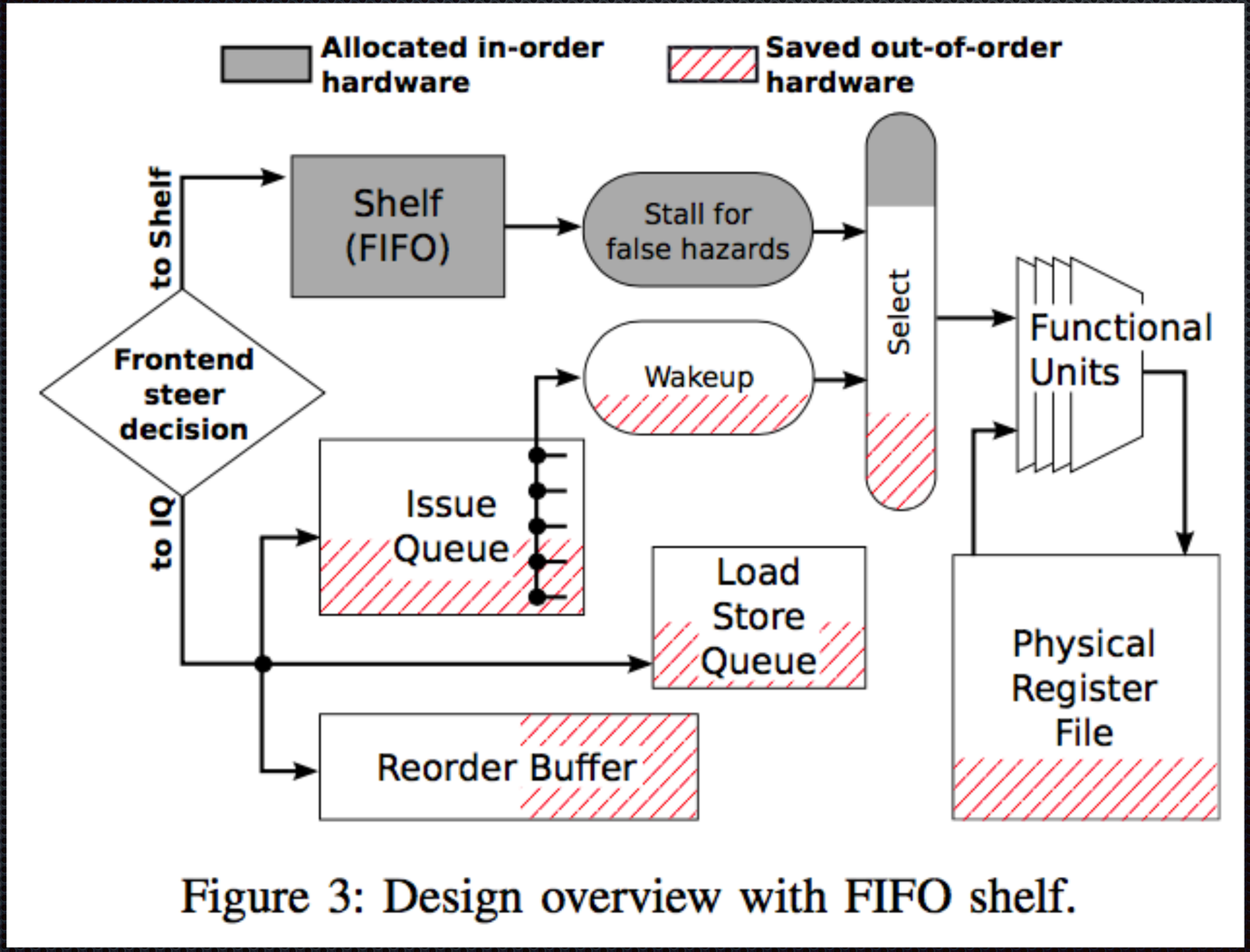


Figure 3: Design overview with FIFO shelf.

Extra Bookeeping

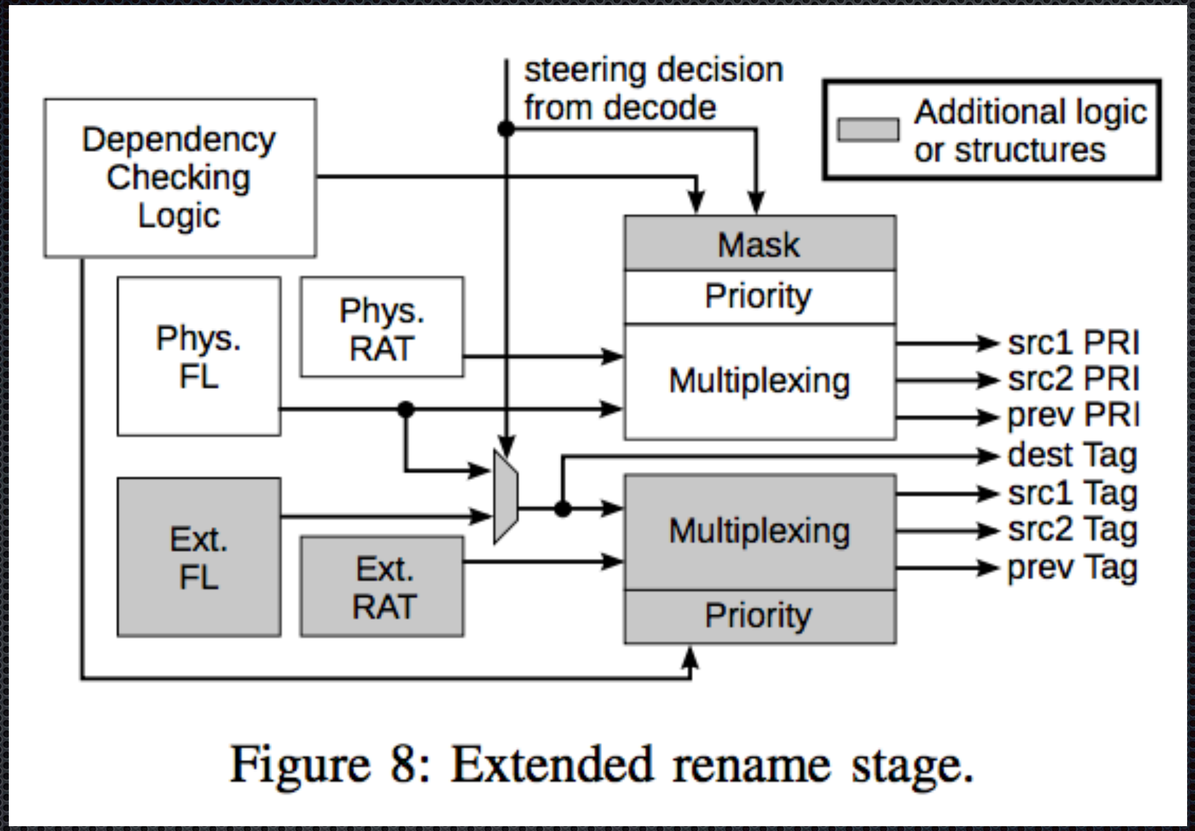


Figure 8: Extended rename stage.

Interactions between INO and OOO instructions require extensive bookeeping and interlocks

Steering

- Calculates completion cycle for INO and OOO and chooses earliest, with ties being INO
  - Operand availability (ready cycle)
  - Instruction latency
  - Sampled load dependence table
  - Earliest issue cycle
- Schedule errors update ready cycle table for subsequent instructions

Evaluation

- Uses gem5 simulator (not validated)
- Random mixes of SPEC benchmarks (not parallel)
- System throughput metric

Versus Baseline

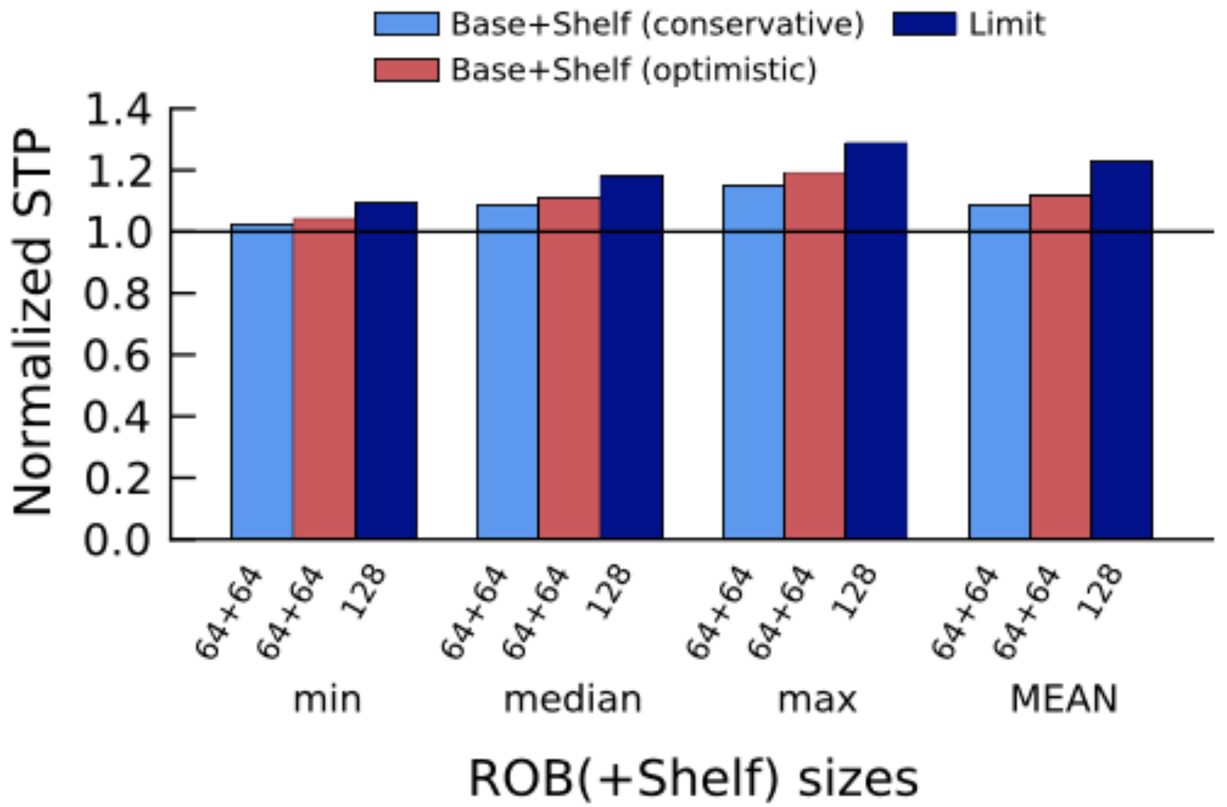


Figure 10: Performance of the shelf with conservative and optimistic microarchitecture assumptions.

Practical Steering vs Opt

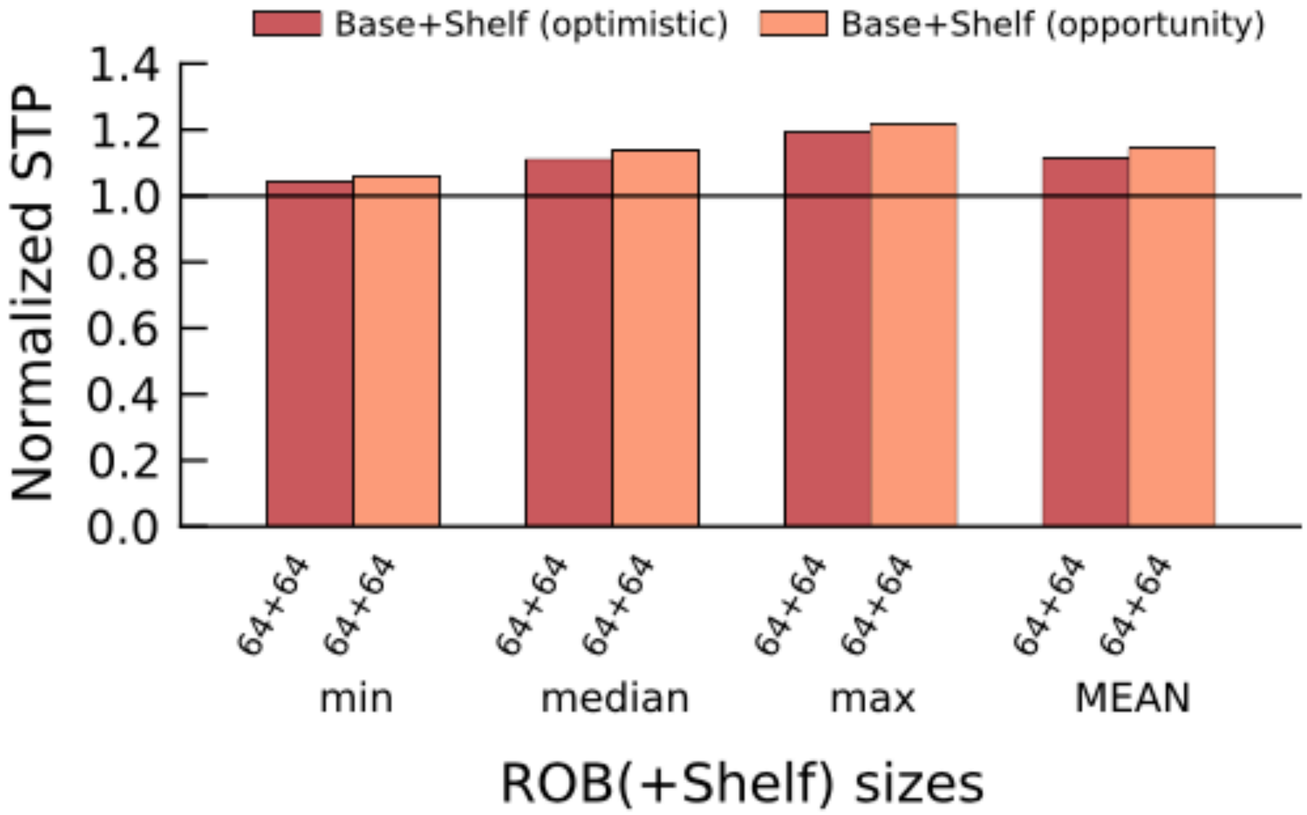


Figure 12: Performance impact of practical steering.

INO Fraction

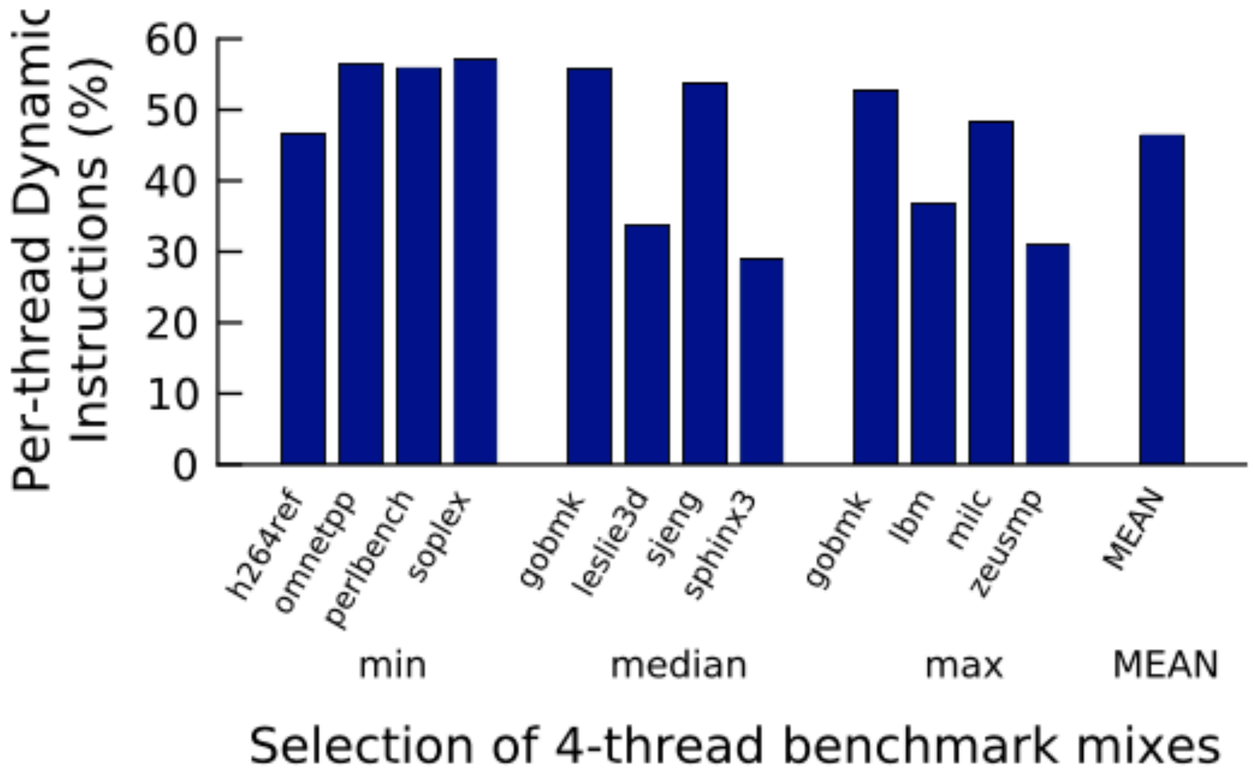


Figure 11: Fraction of in-sequence instructions for a selection of 4-thread benchmark mixes.

Energy Effect

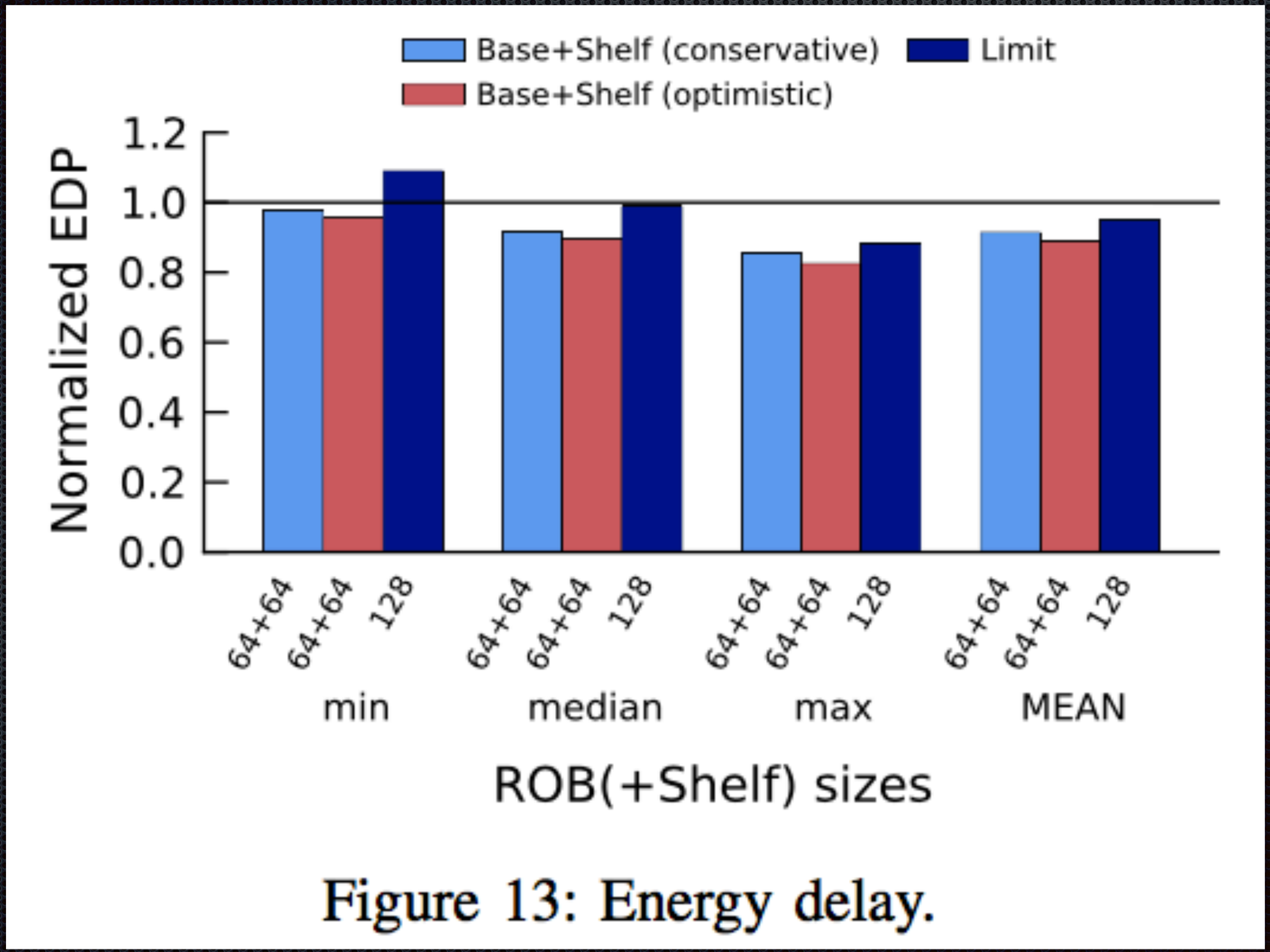


Figure 13: Energy delay.

Discussion