

# Pipelining

Pre-soak    soak    soap    wash    dry    wipe



- Chapter 4.4 and 4.5
- Principles of pipelining
- Pipeline hazards
- Remedies

# Multi-stage process

- Sequential execution
  - One process begins after previous finishes
  - E.g., six stage process; each stage takes 5 sec
  - Time to complete is 30 sec
  - Throughput = 2/minute
- Pipelining
- Begin a stage of the next process as soon as the stage of the previous process is finished
  - Third car in soap, second car in wash, first car in dry
  - Time to complete the wash is ?
  - Throughput is =

# Pipelining for instruction execution

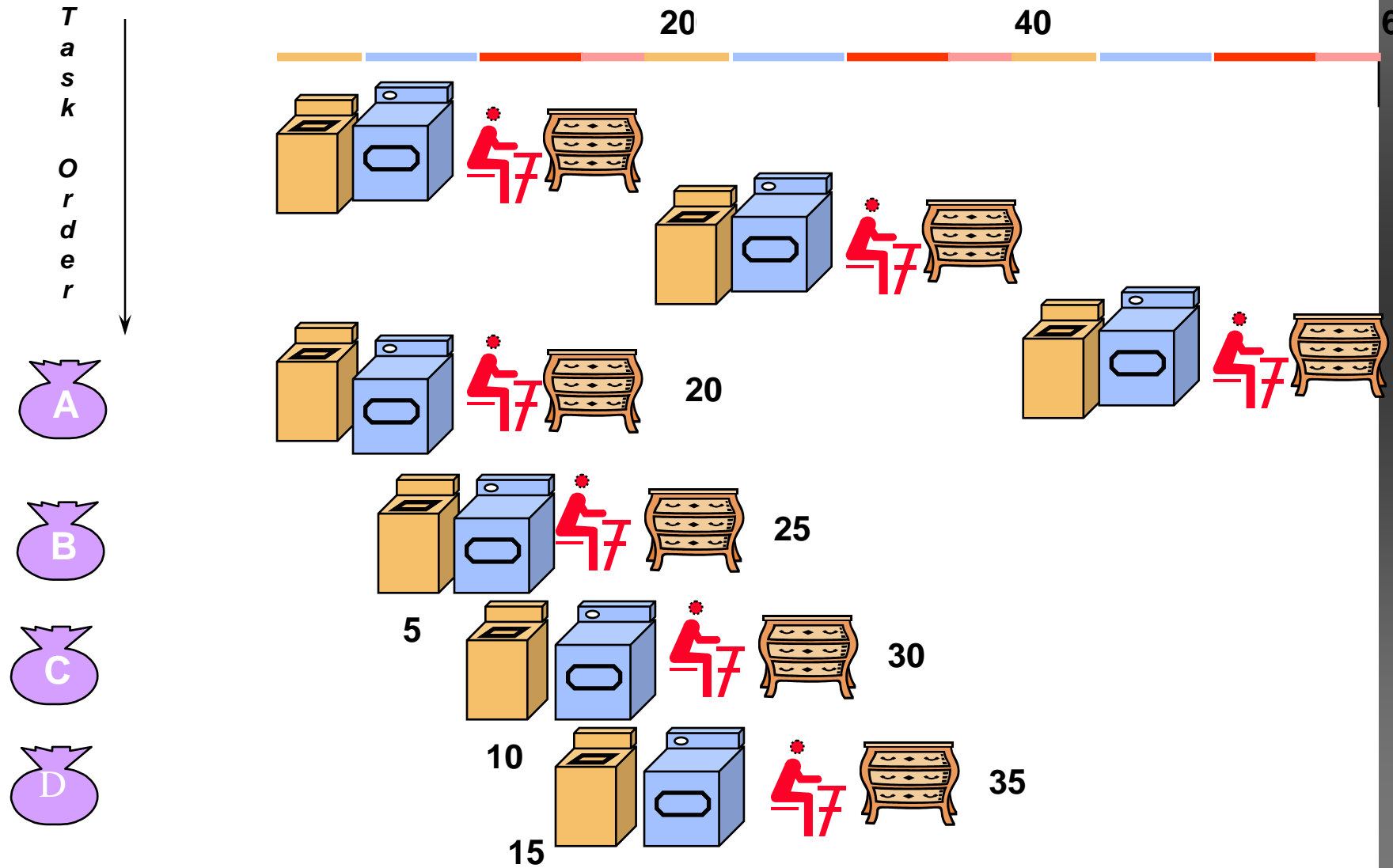
- Each Instruction has several sub-steps
- Each instruction has 5 or 6 stages
  - IF, ID, EX, MEM, WB
- Multiple cycles per instruction
- Typical cycle time (1 GHz Processor – 1 nsec)
- Average instruction can take 3 to 5 cycles
- Sequential execution of one instruction after another
- Too slow
- Alternatives?

# Multicycle

- Multicycle Seq implementation:

Cycle:	1	2	3	4	5	6	7	8	9	10	11	12	13
Instr:													
i	IF	ID	EX	M	WB								
i+1						IF	ID	EX					
i+2									IF	ID	EX	M	
i+3													IF
i+4													

# Laundry example (Wash, dry, fold, store)



# Ideal Pipelining

Cycle:\nInstr:	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				

Ideal throughput: 1 instruction/cycle

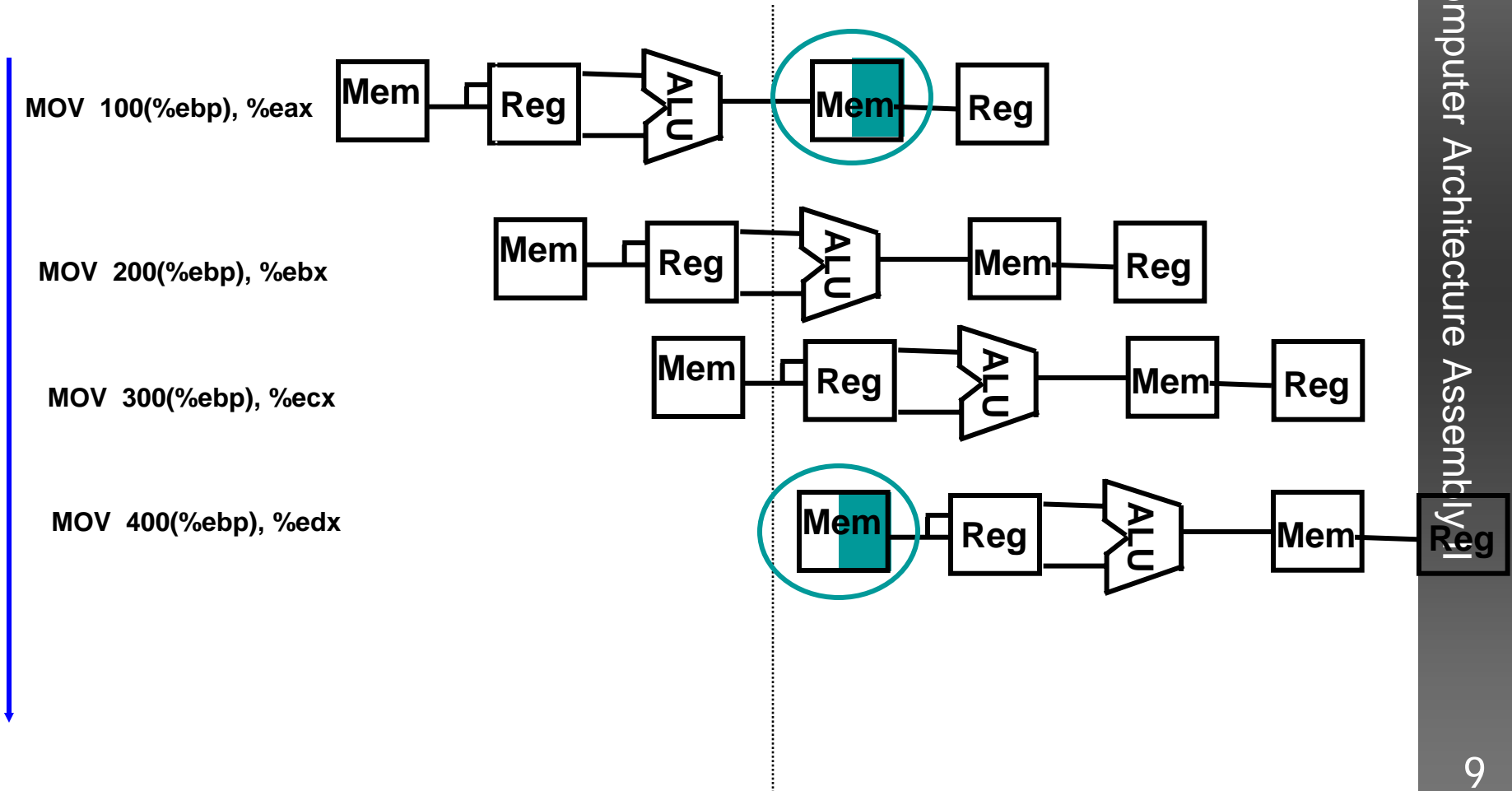
# Pipeline hazards

- Car wash example
  - One stage (soap) may take longer (additional rust proof)
  - Car behind (soak stage) need to idle
  - Pipeline stall
- Clothes wash example
- Resource may not be available
  - Washer-dryer combo
    - Resource shared and in use
  - Dried or washed clothes not emptied
  - Cycle taking longer
  - Pipeline stall

# Pipelining hazards in CPU

- Hazards: A stage of the instruction cannot execute in the next clock cycle
- Three types of hazards
- Structural hazards
  - Two different instructions use the same hardware
  - Resource conflict (e.g., One memory port)
- Data hazards
  - Two different instructions have a dependency on data
  - Two instructions use the same register/memory
- Control hazards
  - One instruction's execution affects which instruction is next
  - E.g, jmp
  - Instruction that affects PC (%eip)

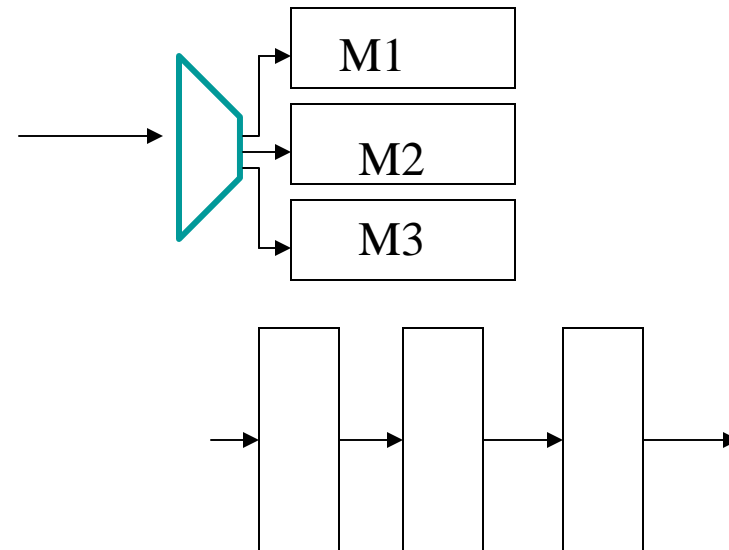
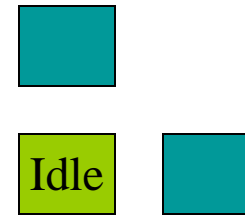
# Structural Hazard



Cant read memory twice in the same cycle

# Dealing with Structural hazards


- Stall
  - Block until previous operation completes
  - Decreases throughput or Ins/cycle
- Replicate resource
  - Separate memory banks
  - Dual port memory
  - Can read different addresses in the same cycle
  - Need H/W, cost
- Pipeline the resource
  - Can be used for multi-cycle resource



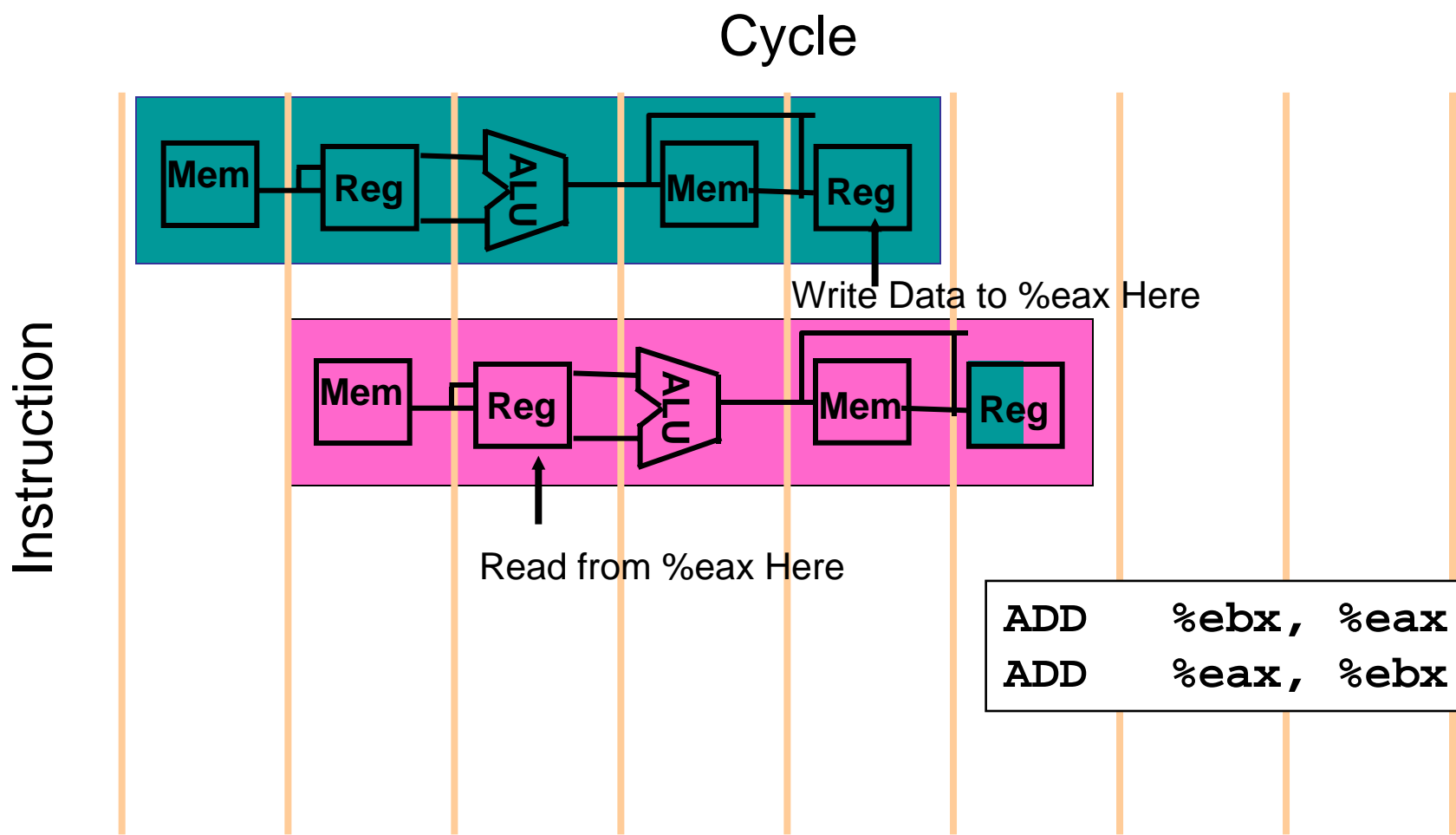
# Pipeline Data Hazards

- Data hazards
  - an instruction uses the result of a previous instruction

ADD	%ecx, %eax	or	MOV	%eax, 100(%ebp)
ADD	%eax, %edx		MOV	100(%ebp), %ebx



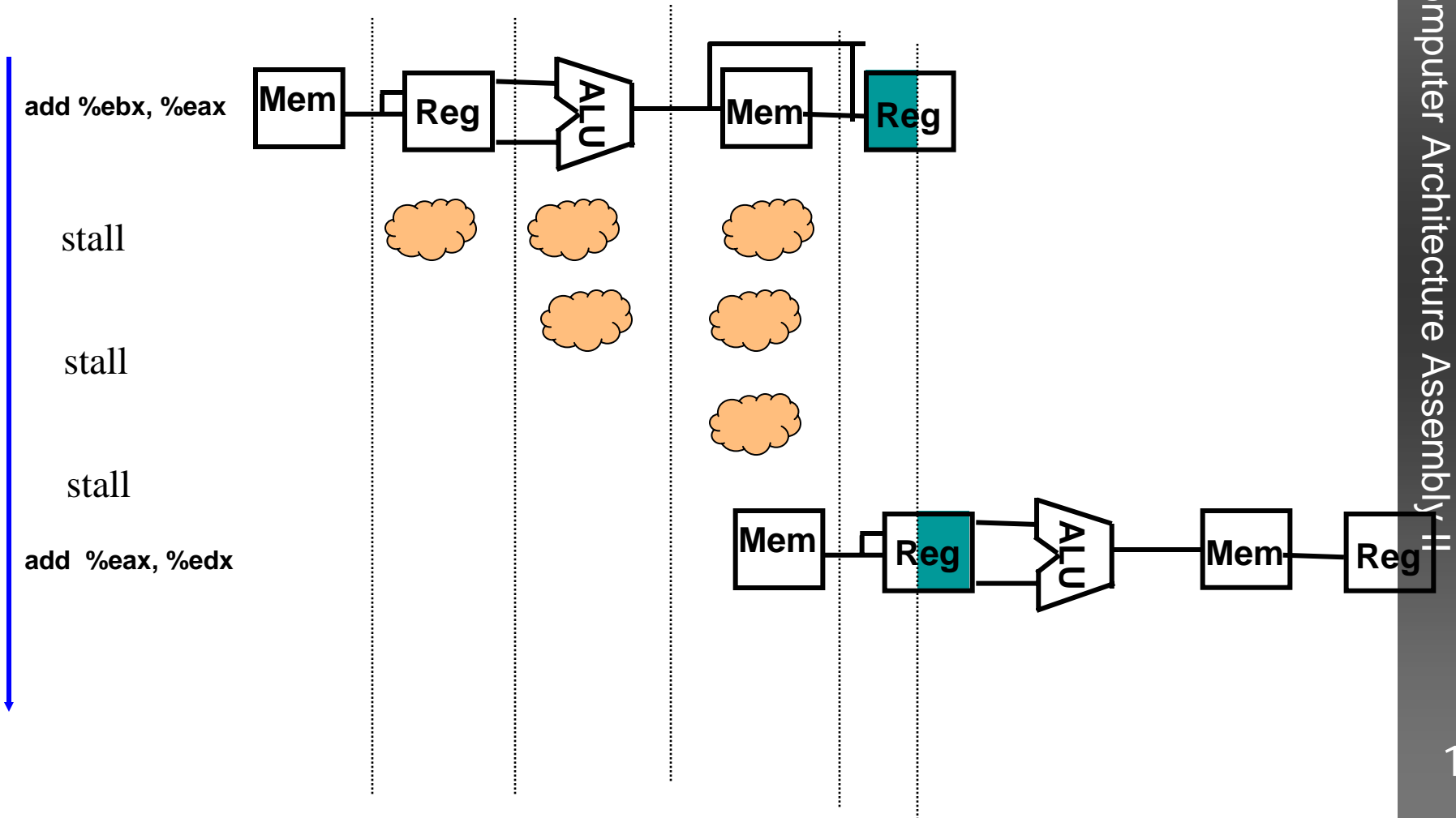
# Data Hazards (RAW)



# Dealing with data hazards

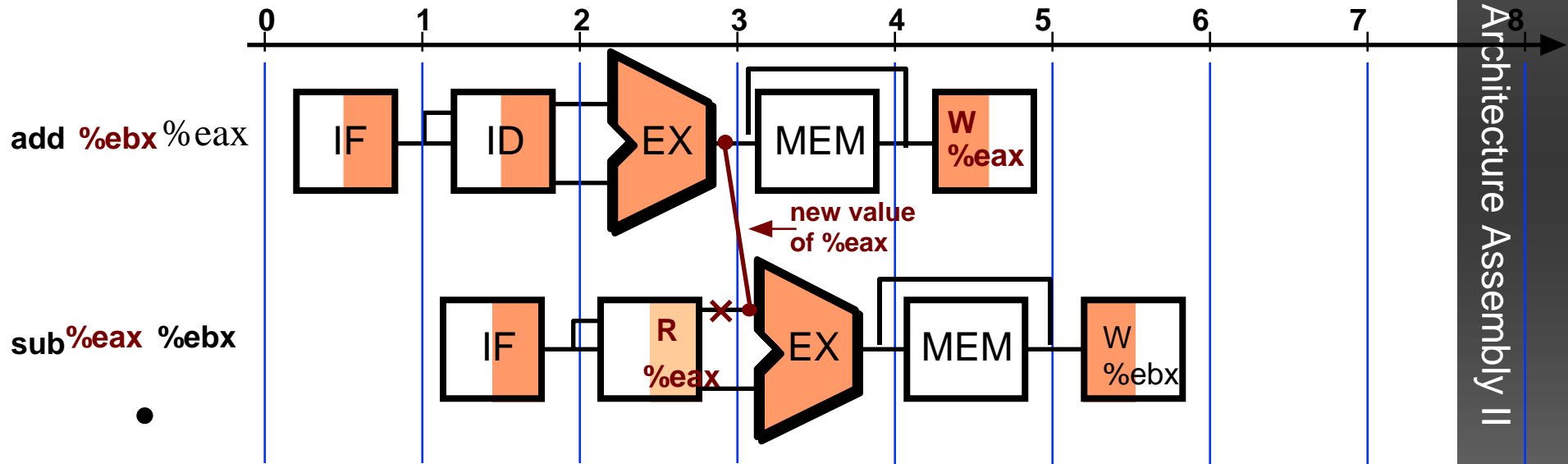
- Stalling
  - Hardware detects RWA conflicts and stalls
- Forwarding
  - connect new value directly to next stage
  - Make data available internally before it is stored

# Data Hazard - Stalling



# Data Hazards - Forwarding

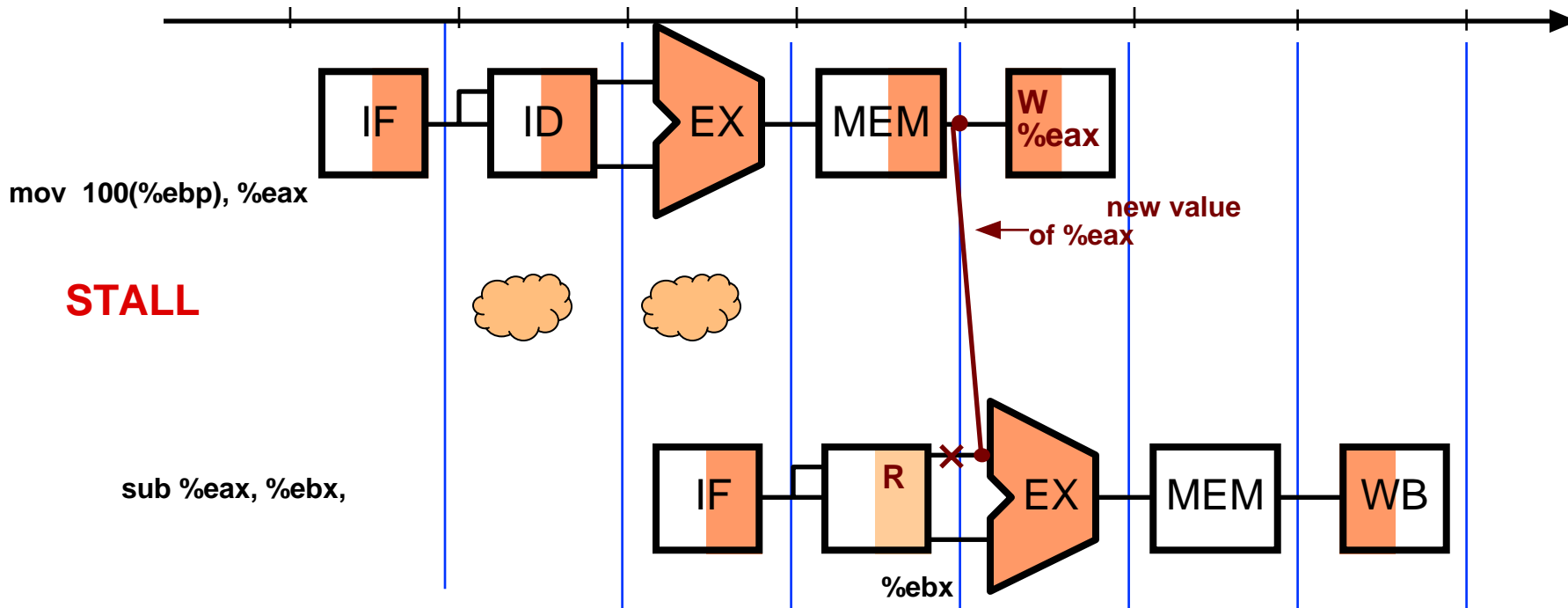
- Key idea: connect new value directly to next stage
- Input to next stage is the new result



- Problem: what about move from memory instructions?

# Data Hazards - Forwarding

- Data to be loaded in %eax available only after memory read of location 100 + (%ebp)



# Data hazards; Compiler optimization

With loads, data available only after M stage; requires stalls

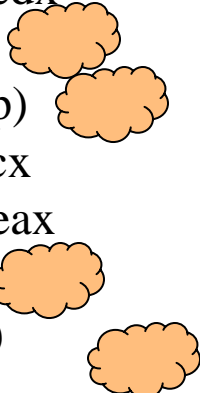
Compiler can reorder instructions

Execute instructions out of order to fill in for pipeline stalls

```

mov -8(%ebp), %eax
mov -12(%ebp), %edx
add %edx, %eax
mov %eax, 4(%ebp)
mov -16(%ebp) %ecx
mov -20(%ebp), %eax
sub %ecx, %eax
mov %eax, 8(%ebp)


```



```

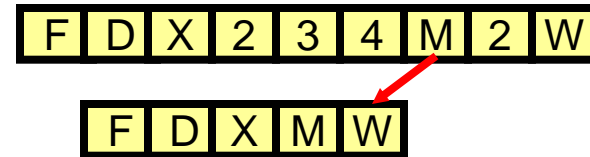
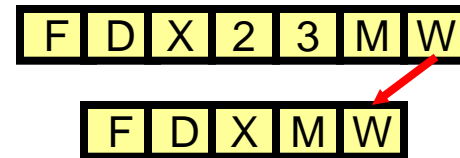
mov -8(%ebp), %eax
mov -12(%ebp), %edx
mov -16(%ebp), %ecx
add %edx, %eax
mov -20(%ebp), %edx
mov %eax, 4(%ebp)
sub %edx, %eax
mov %eax, 8(%ebp)

```



# Other Types of Data Hazards

- WAW (write after write)
  - variable-length pipeline
  - EX can take 3 cycles
  - later instruction must *write* after earlier instruction *writes*
- WAR (write after read)
  - Variable length pipelines
  - Instructions with late read or memory access
  - later instruction must *write* after earlier instruction *reads*



# Pipeline Data Hazards

- Control hazards
  - the location of an instruction depends on a previous instruction

```

                JMP     LOOP
                ...
LOOP:         ADD     %edx, %eax
    
```

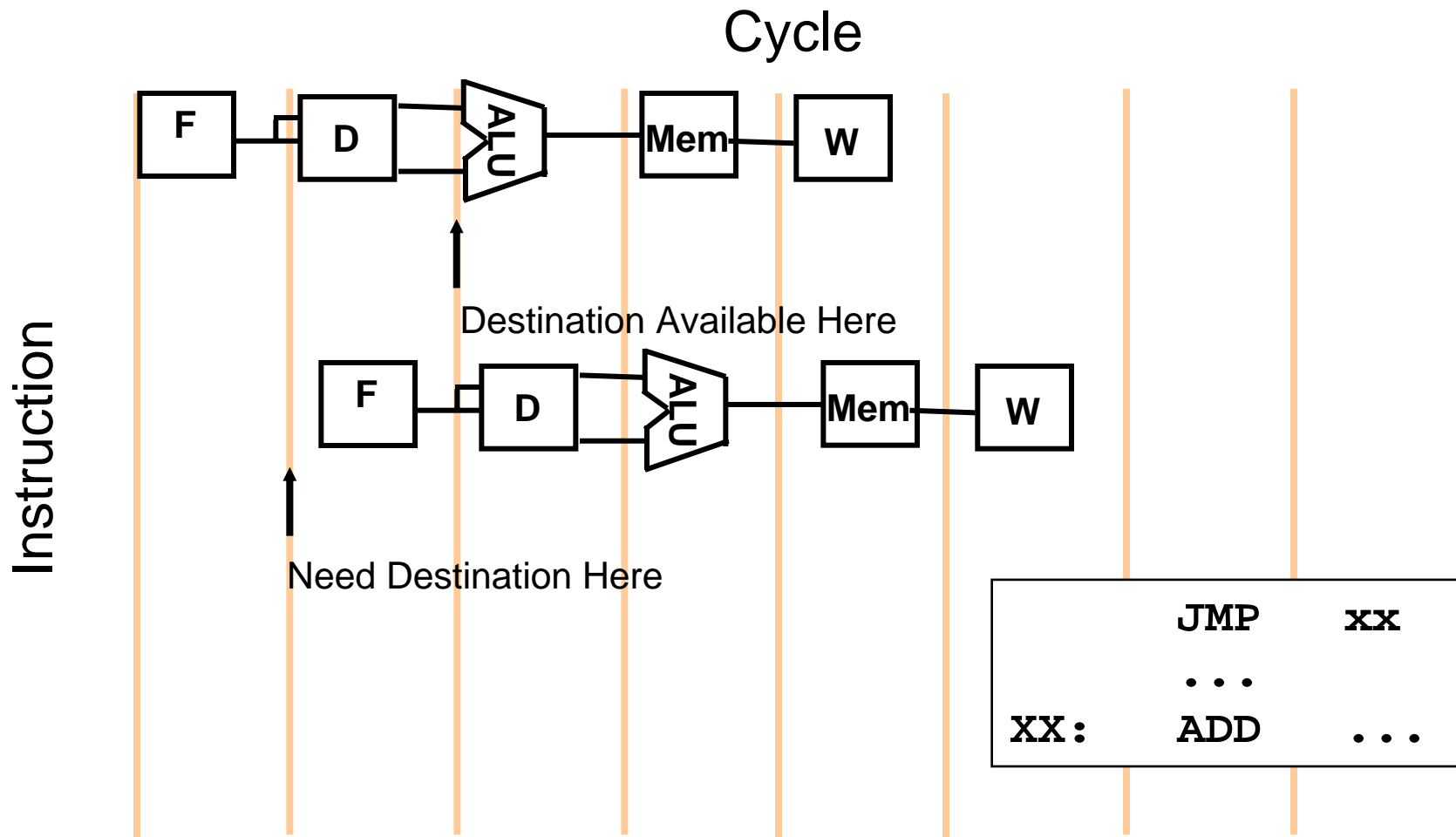


We cant fetch the next instruction until we know the branch address

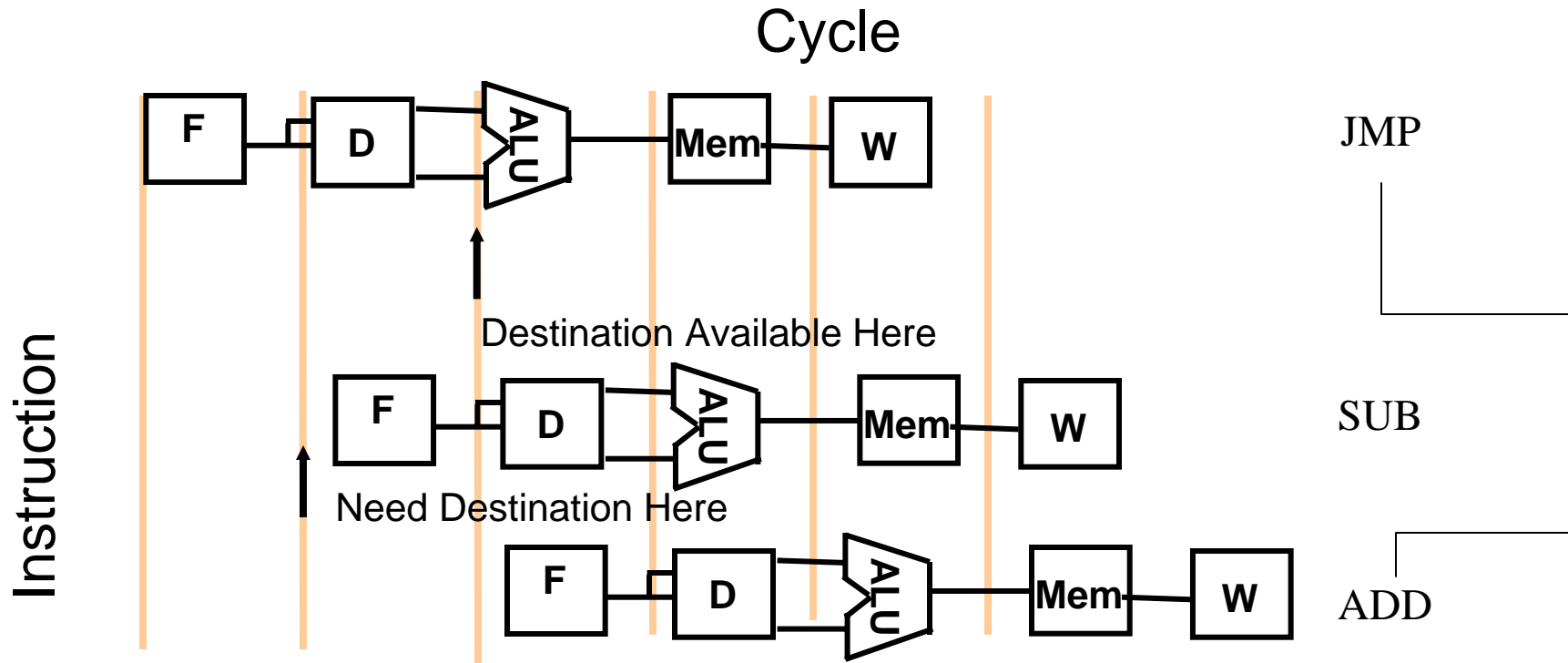
Branch taken  $PC \leftarrow \text{Immediate}$

Branch not taken  $PC \leftarrow PC + 4$

# Control Hazards: Need to know which to fetch



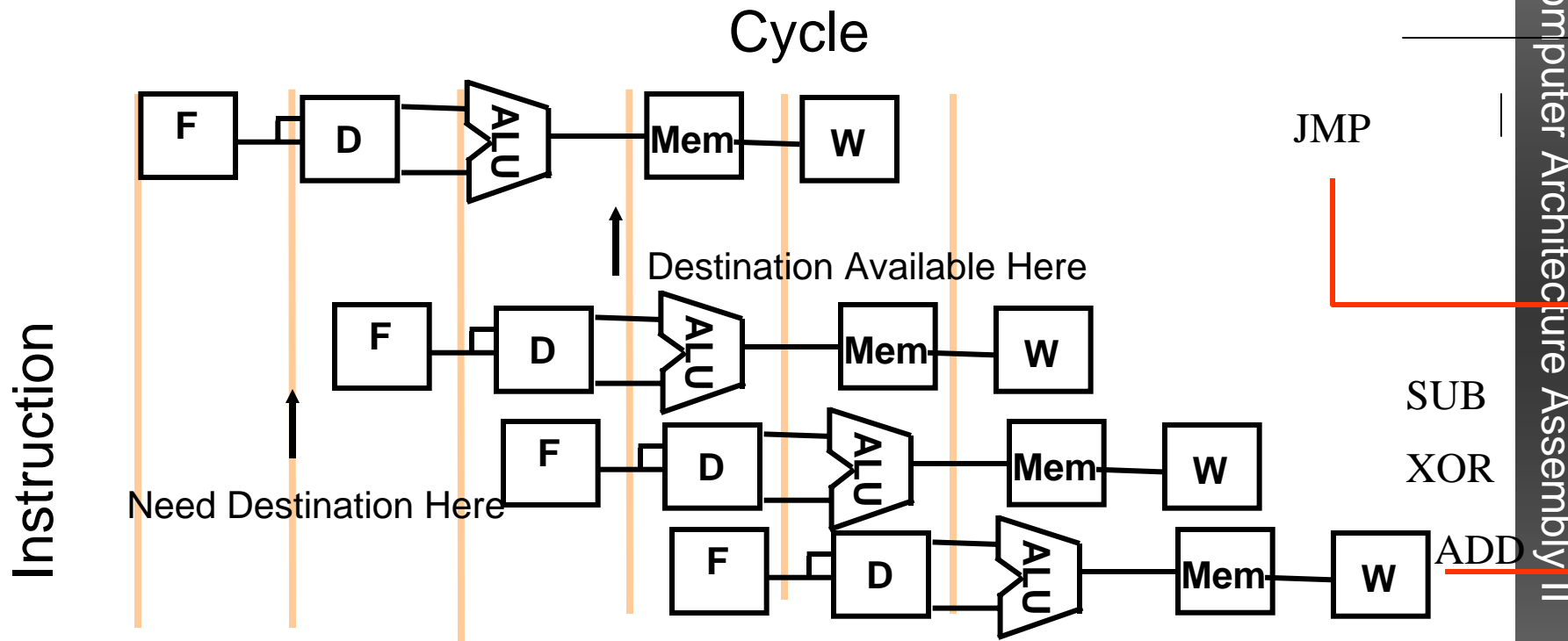
# Control Hazards: Assume branch not taken



Execute SUB, if wrong flush pipeline  
 Flush instruction in F stage... introduce NOP

	JMP	xx
	SUB	
xx:	ADD	...

# Control Hazards: In Some CPUs, address available after EX stage



Execute SUB, and XOR; if wrong flush pipeline  
 Flush instruction in F stage and D stage  
 ... introduce NOP

	JMP	xx
	SUB	
	XOR	
xx:	ADD	...

# Summary

- Pipelining principles
- Hazards
- Hazard prevention