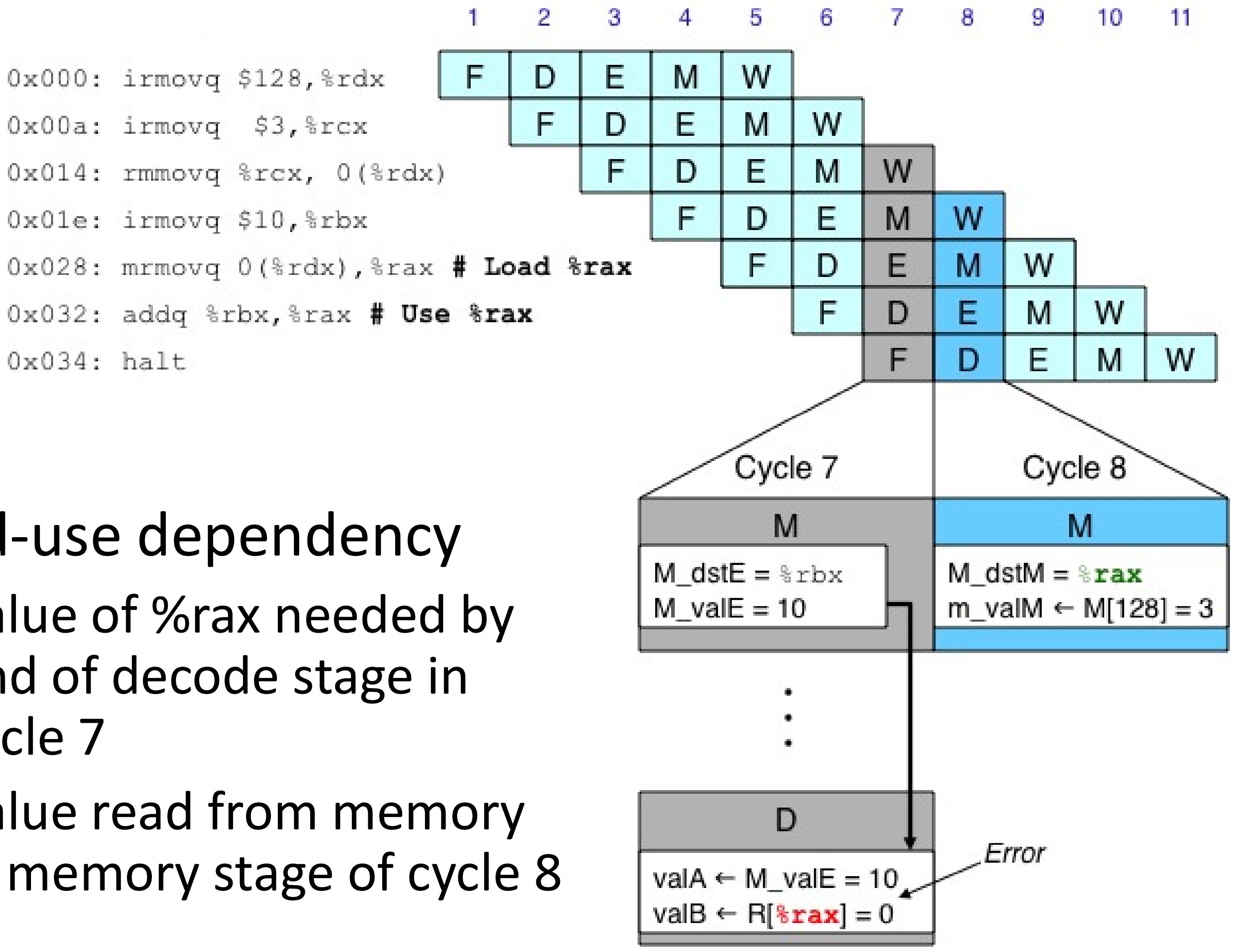
Computer Systems Organization CS2.201

Topic 4

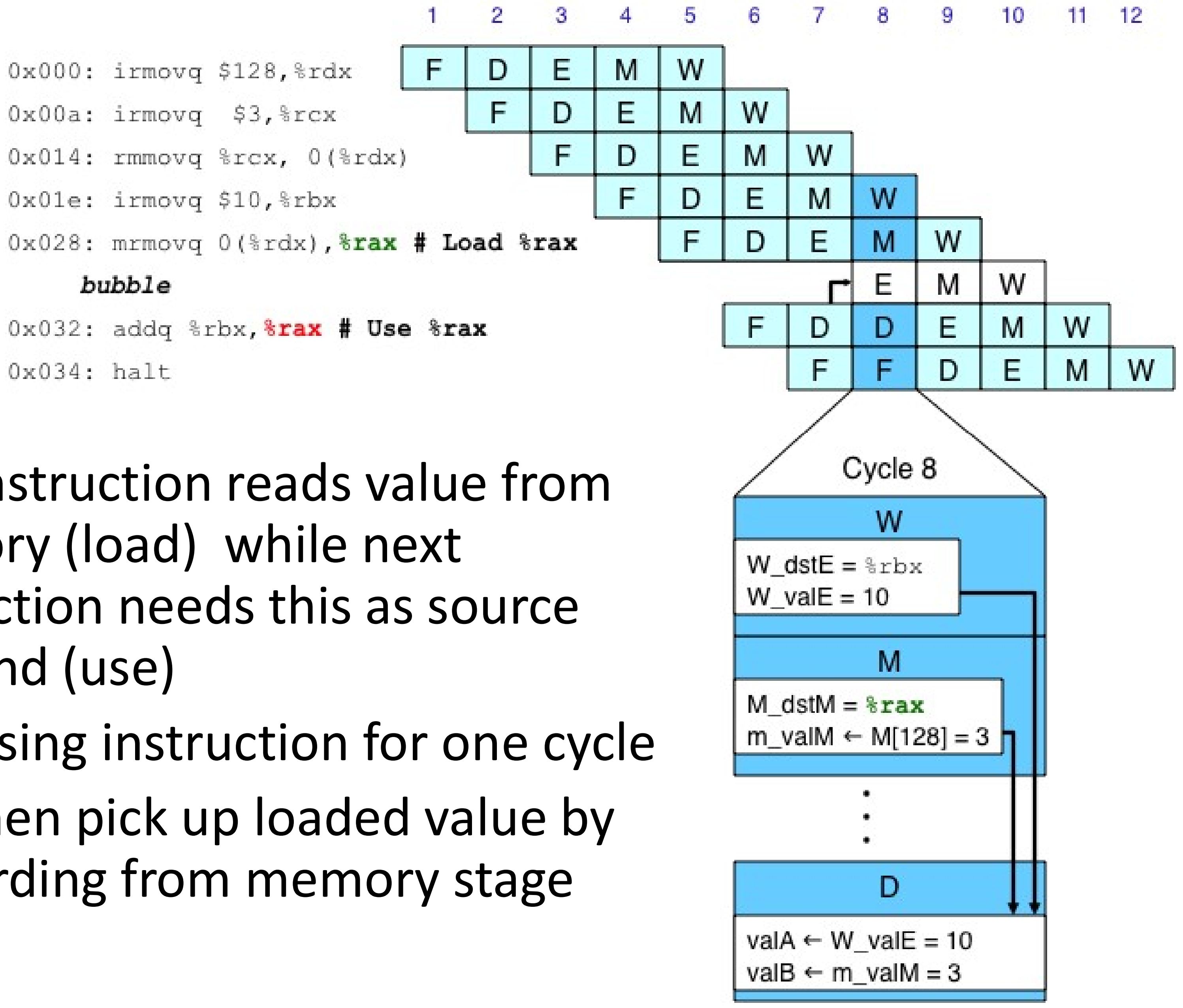
Based on chapter 4 from Computer Systems by Randal E. Bryant and David R. O'Hallaron

Limitation of Forwarding



- Load-use dependency
  - Value of %rax needed by end of decode stage in cycle 7
  - Value read from memory in memory stage of cycle 8

Avoiding Load/Use Hazard



- One instruction reads value from memory (load) while next instruction needs this as source operand (use)
- Stall using instruction for one cycle
- Can then pick up loaded value by forwarding from memory stage

Control for Load/Use Hazard

	1	2	3	4	5	6	7	8	9	10	11	12
`0x000: irmovq $128,%rdx`	F	D	E	M	W							
`0x00a: irmovq $3,%rcx`		F	D	E	M	W						
`0x014: rmmovq %rcx, 0(%rdx)`			F	D	E	M	W					
`0x01e: irmovq $10,%ebx`				F	D	E	M	W				
`0x028: mrmovq 0(%rdx),%rax # Load %rax`					F	D	E	M	W			
bubble								E	M	W		
`0x032: addq %ebx,%rax # Use %rax`						F	D	D	E	M	W	
`0x034: halt`							F	F	D	E	M	W

– Stall instructions in fetch and decode stages

– Inject bubble into execute stage

Condition	F	D	E	M	W
Load/Use Hazard	**stall**	**stall**	**bubble**	**normal**	**normal**

Load/Use Data Hazard

- Use of stall to handle a load/use hazard is called a load interlock
- Load interlocks combined with forwarding suffice to handle all possible forms of data hazards
- Can nearly achieve goal of issuing one new instruction on every clock cycle

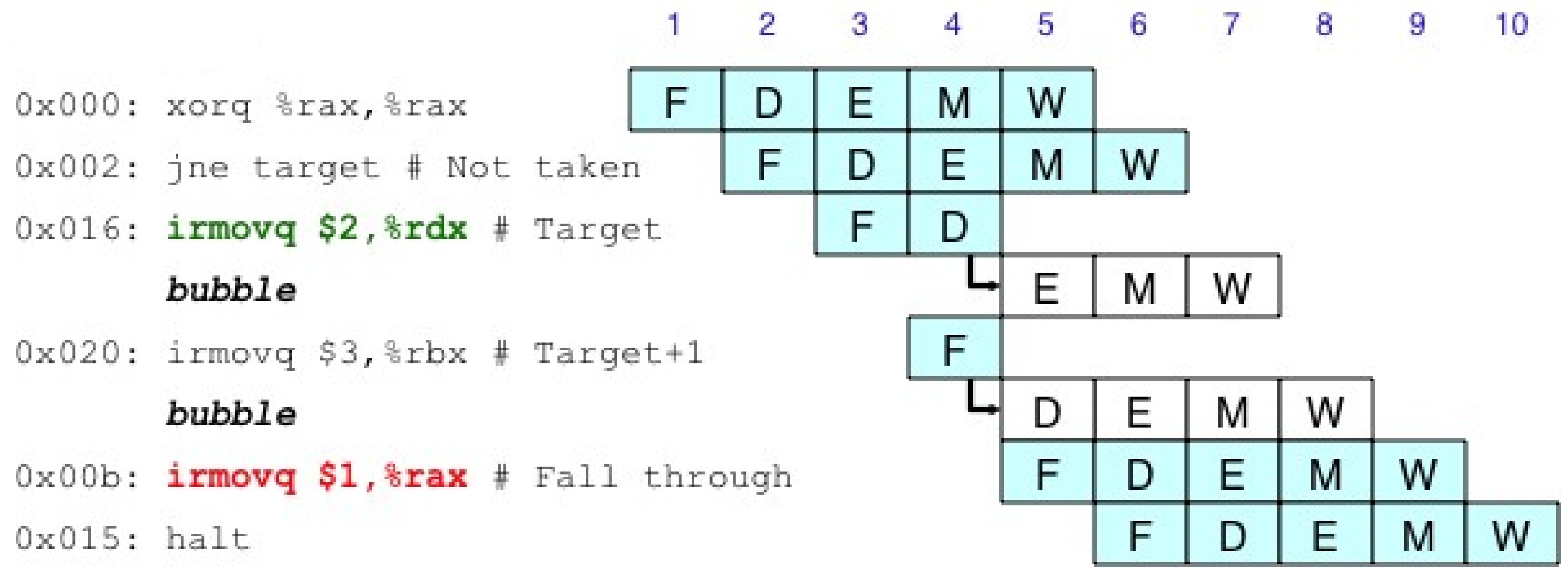
Branch Misprediction Example

```
0x000:    xorq %rax,%rax
0x002:    jne  t               # Not taken
0x00b:    irmovq $1, %rax      # Fall through
0x015:    halt
0x016: t:
0x016:    irmovq $2, %rdx      # Target
0x020:    irmovq $3, %rcx      # Should not execute
Ox02a:    halt
```

Branch Misprediction Example

- Since jump instruction is predicted as being taken, instruction for jump target will be fetched at cycle 3 and instruction following this in cycle 4
- Branch logic detects jump should not be taken in cycle 4
  - Two instructions fetched meanwhile that should not continue being executed
  - Neither of these instructions caused a change in the programmer visible state (can occur when an instruction reaches execute stage)

Handling Misprediction


1
2
3
4
5
6
7
8
9
10
0x000: xorq %rax,%rax
0x002: jne target # Not taken
0x016: irmovq $2,%rdx # Target
bubble
0x020: irmovq $3,%rbx # Target+1
bubble
0x00b: irmovq $1,%rax # Fall through
0x015: halt
F D E M W

Predict branch as taken

- Fetch 2 instructions at target

Cancel when mispredicted

- Detect branch not-taken in execute stage
- On following cycle, replace instructions in execute and decode by bubbles
- No side effects have occurred yet

Branch Misprediction Example

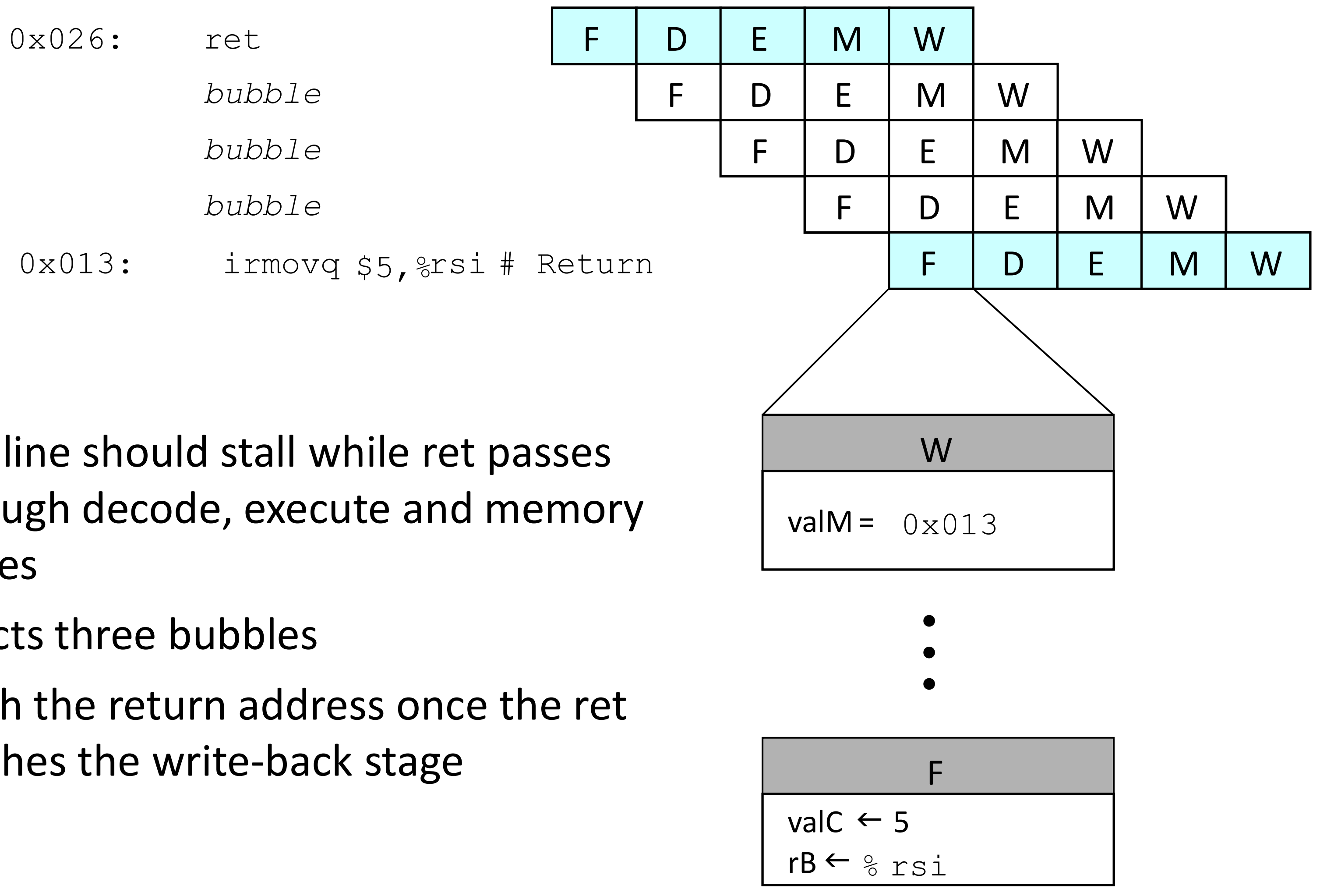
- Pipeline can cancel the two mis-fetched instructions (called instruction squashing) by injecting bubbles into the execute and decode stages on the following cycle
  - Also fetch instruction following the jump instruction
- The two misfetched instructions will simply disappear from the pipeline and therefore do not have any effect on the programmer visible state

Return Example

```
0x000:     irmovq Stack,%rsp   # Intialize stack pointer
0x00a:     call p              # Procedure call
0x013:     irmovq $5,%rsi      # Return point
0x01d:     halt
0x020: .pos 0x20
0x020: p:  irmovq $-1,%rdi     # procedure
0x02a:     ret
0x02b:     irmovq $1,%rax      # Should not be executed
0x035:     irmovq $2,%rcx      # Should not be executed
0x03f:     irmovq $3,%rdx      # Should not be executed
0x049:     irmovq $4,%rbx      # Should not be executed
0x100: .pos 0x100
0x100: Stack:                  # Stack: Stack pointer
```

– Previously executed three additional instructions

Correct Return Example



- Pipeline should stall while ret passes through decode, execute and memory stages
- Injects three bubbles
- Fetch the return address once the ret reaches the write-back stage

Pipeline Summary

- Data Hazards
  - Most handled by forwarding
    - No performance penalty
  - Load/use hazard requires one cycle stall
- Control Hazards
  - Cancel instructions when detect mispredicted branch
    - Two clock cycles wasted
  - Stall fetch stage while `ret` passes through pipeline
    - Three clock cycles wasted