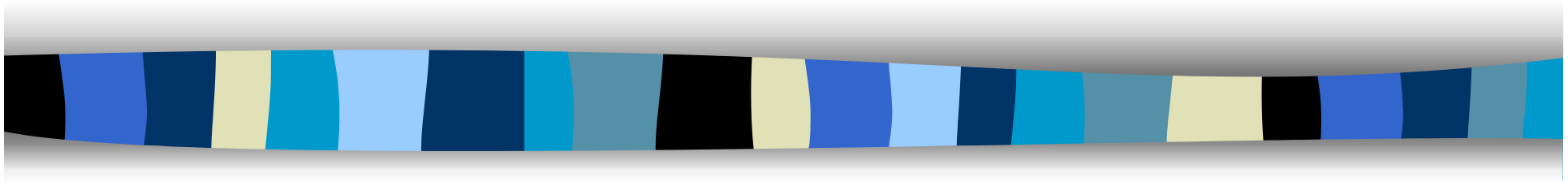
Lecture 3

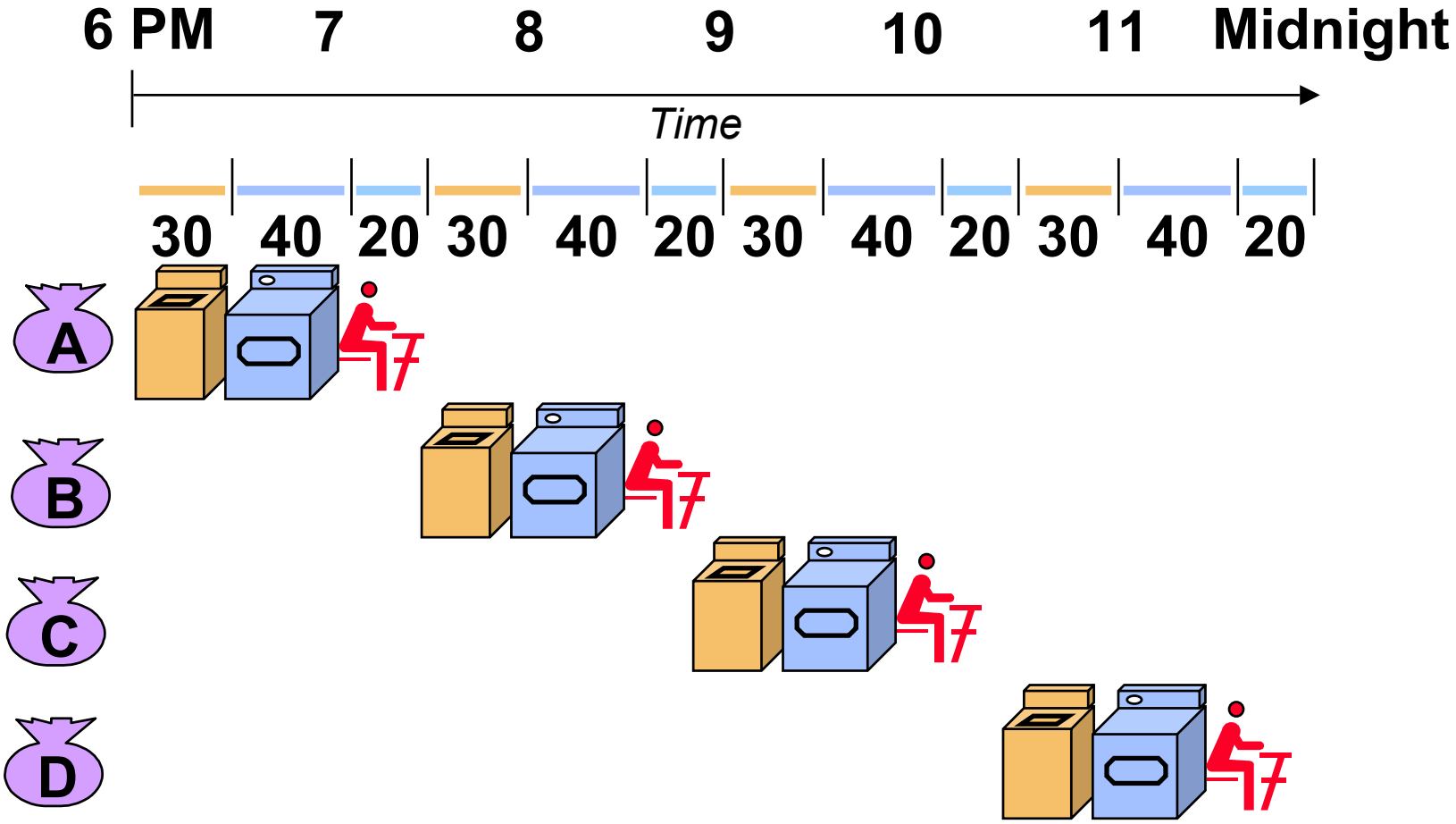


- Control Hazard
- Exception
- Multi-cycle DLX Pipe

Administration

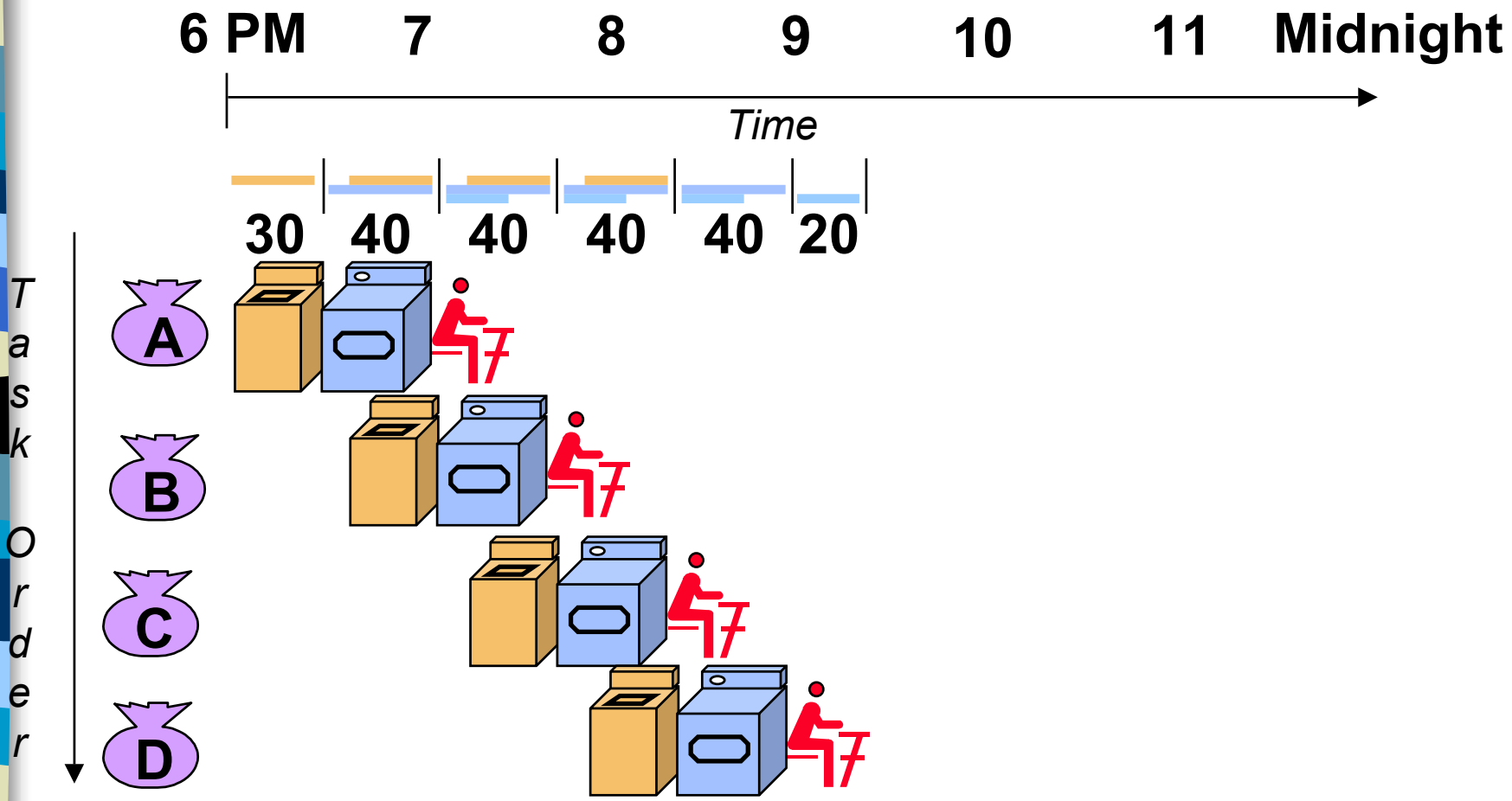
- DLXView Website
  - www.ecs.umass.edu/ece/koren/ece668/windlx.html
- Homework #1
- Readings
  - Computer Organization & Design, the hardware/software interface. David A. Patterson and John L. Hennessy

Review: Basic Pipeling
Sequential Laundry



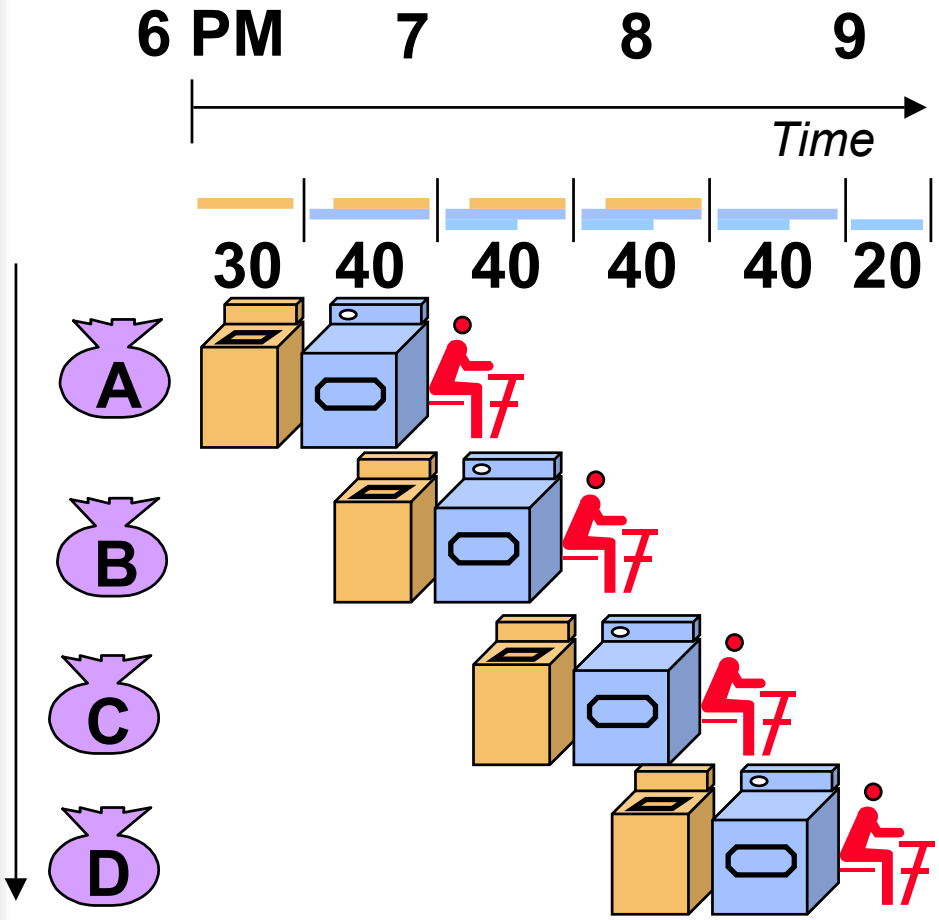
- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take‘

Pipelined Laundry
Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

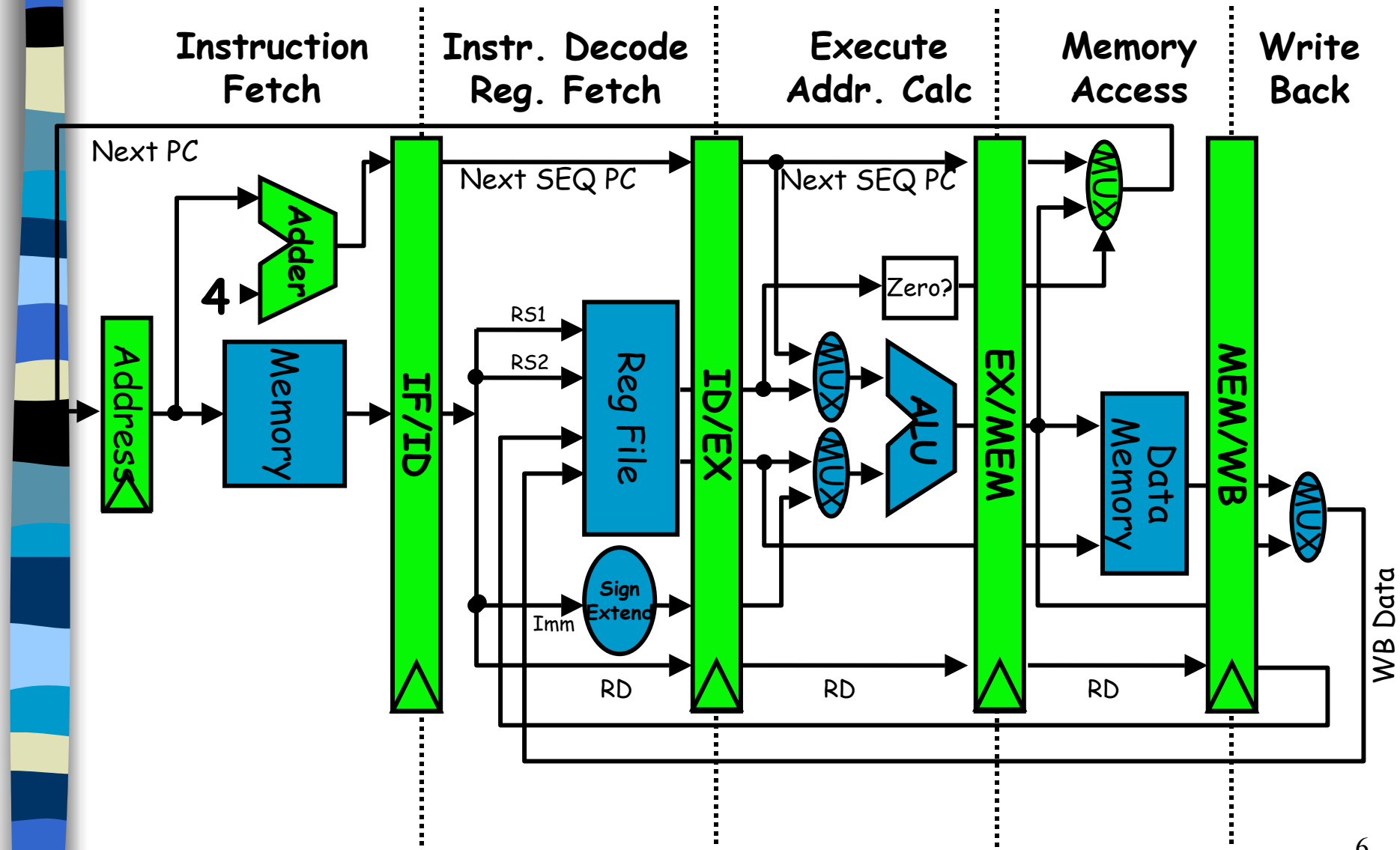
Pipelining Lessons



- Pipelining does not help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup

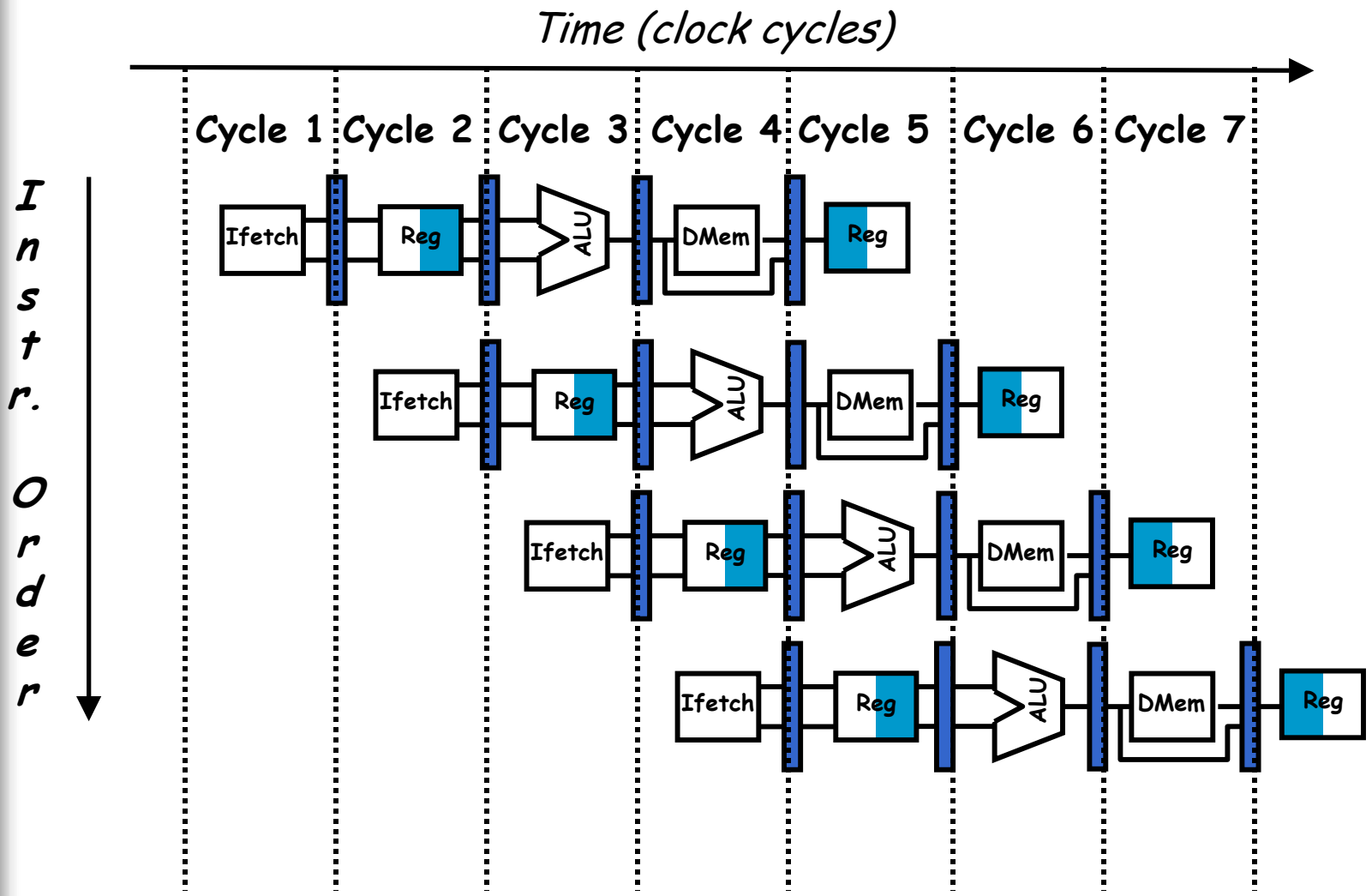
5 Steps of DLX Datapath

Figure 3.4, Page 134 , CA:AQA 2e



Visualizing Pipelining

Figure 3.3, Page 133 , CA:AQA 2e

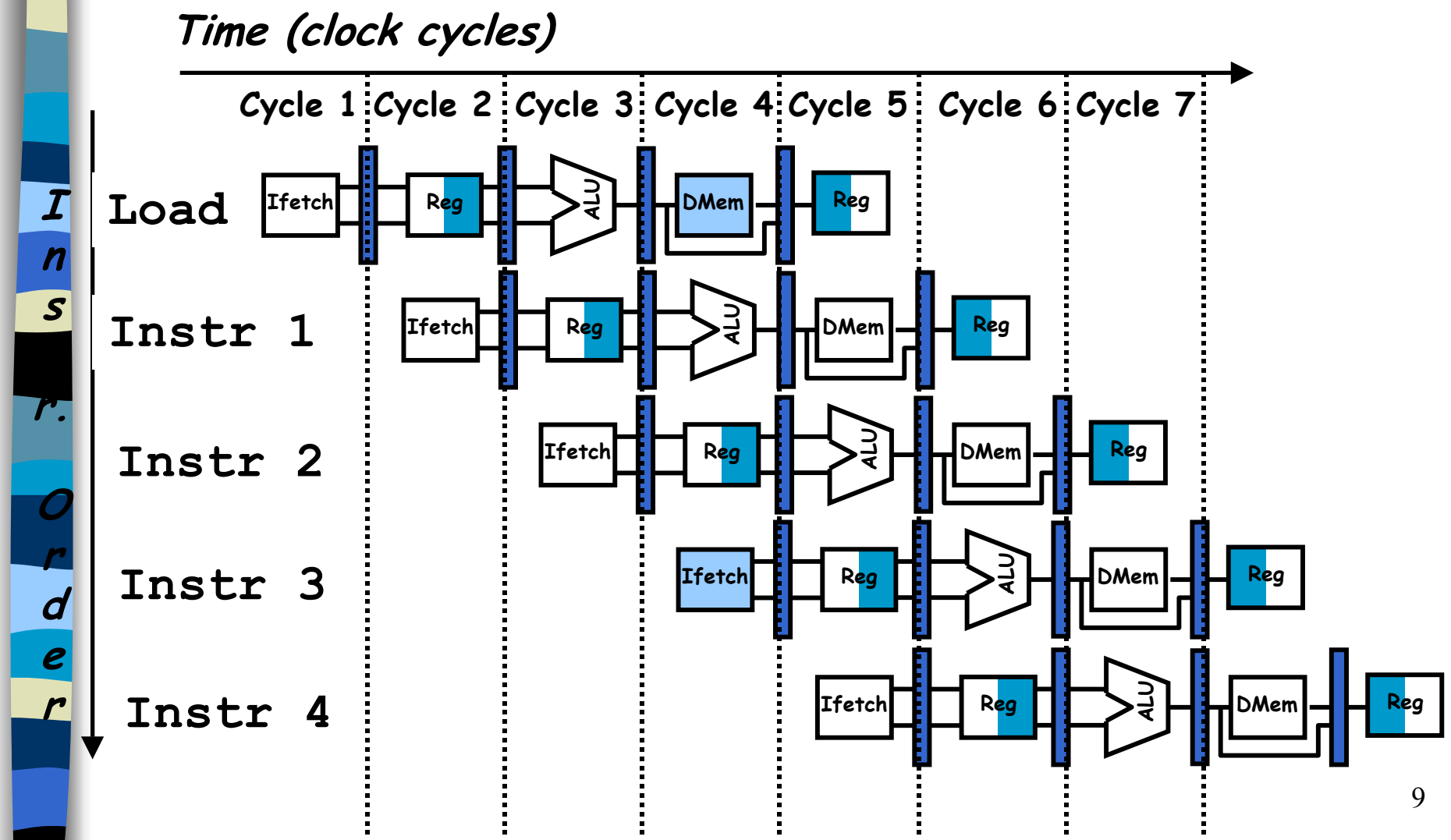


Hazards

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Pipelining of branches & other instructions
- Common solution is to stall the pipeline until the hazard resolved.

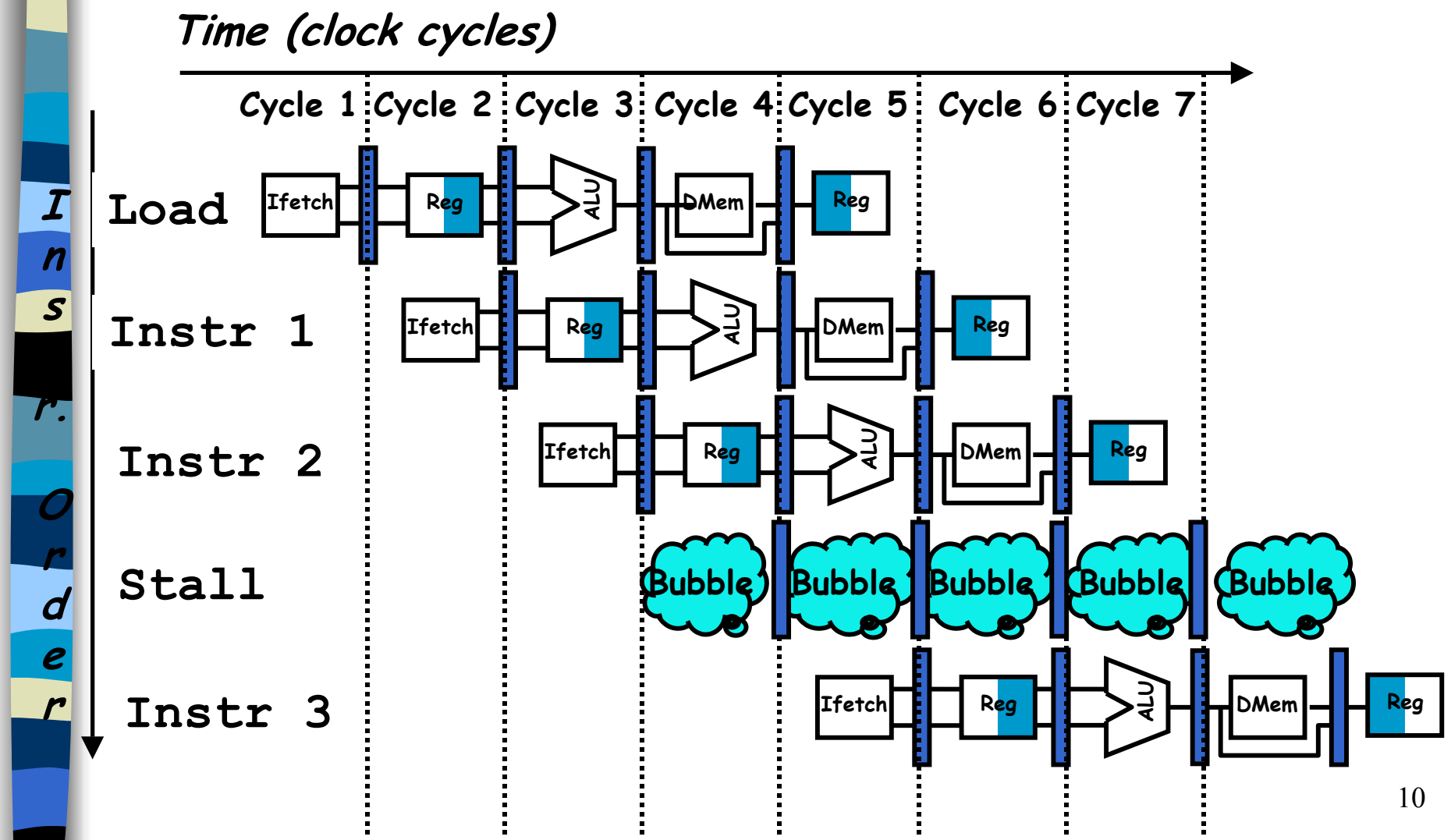
One Memory Port/Structural Hazards

Figure 3.6, Page 142 , CA:AQA 2e



One Memory Port/Structural Hazards

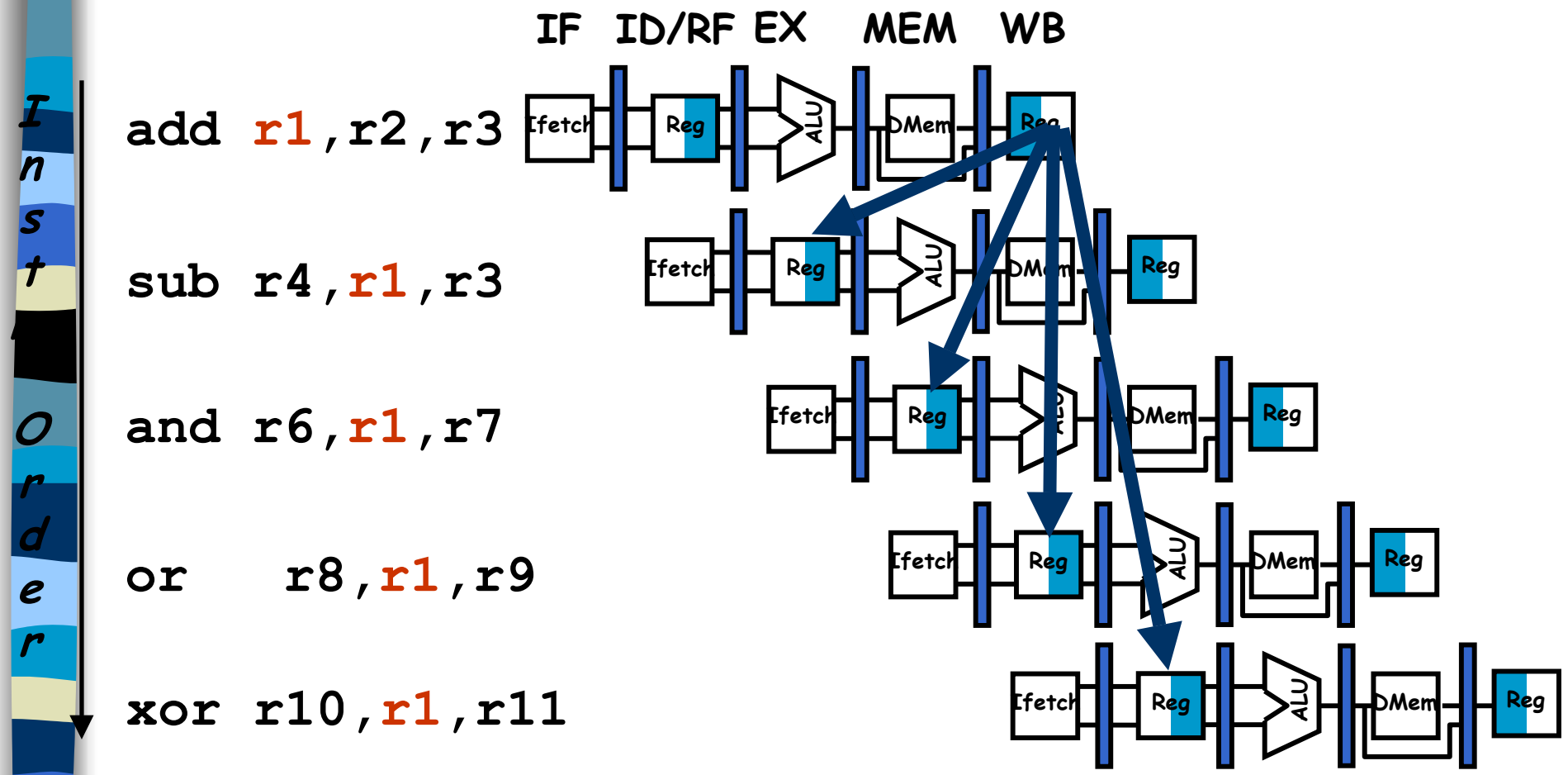
Figure 3.7, Page 143 , CA:AQA 2e



Data Hazard on R1

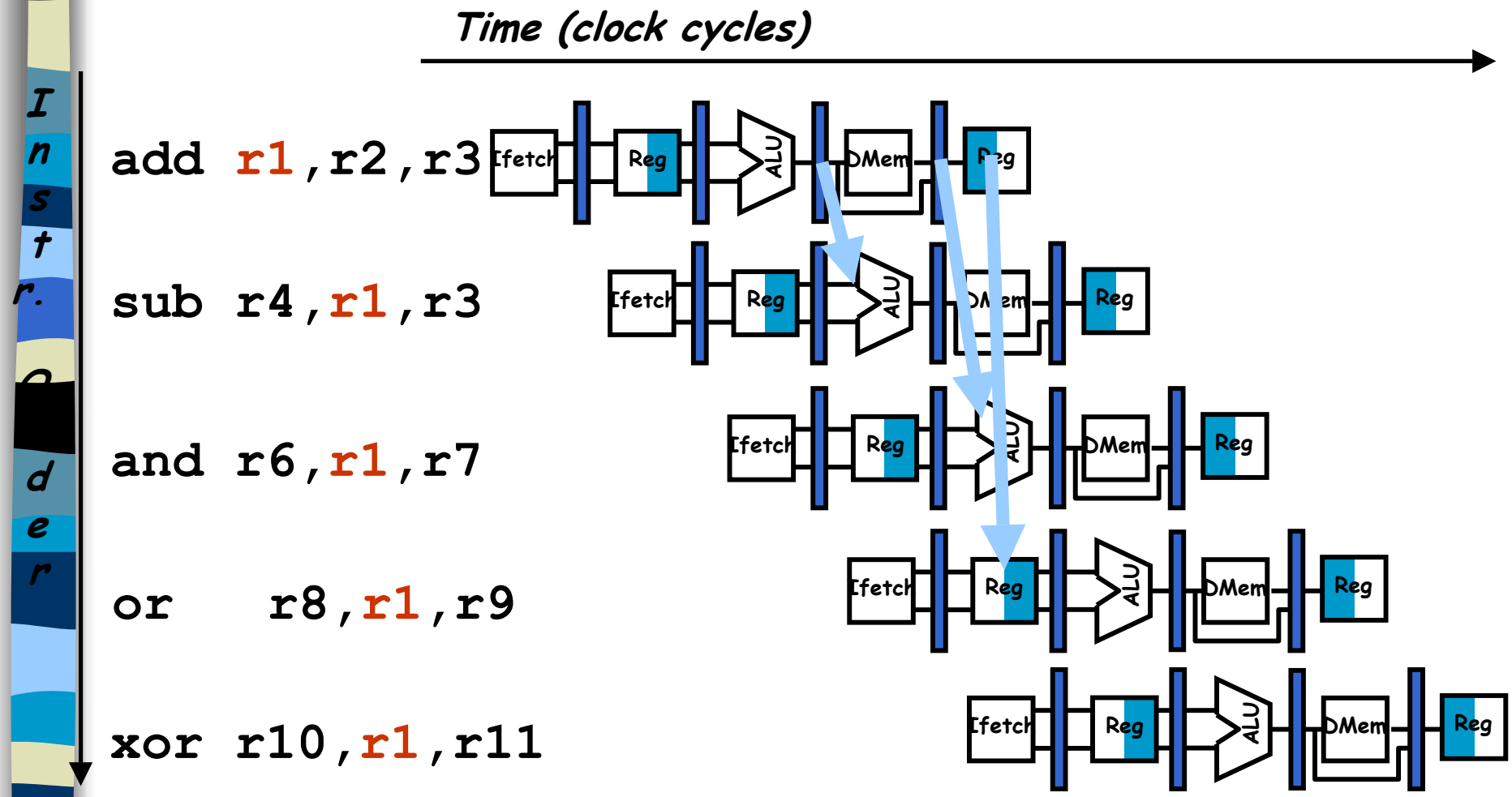
Figure 3.9, page 147 , CA:AQA 2e

Time (clock cycles)



Forwarding to Avoid Data Hazard

Figure 3.10, Page 149 , CA:AQA 2e



Software Scheduling to Avoid Load Hazards

Try producing fast code for

a = b + c;

d = e – f;

assuming a, b, c, d ,e, and f in memory.

Slow code:

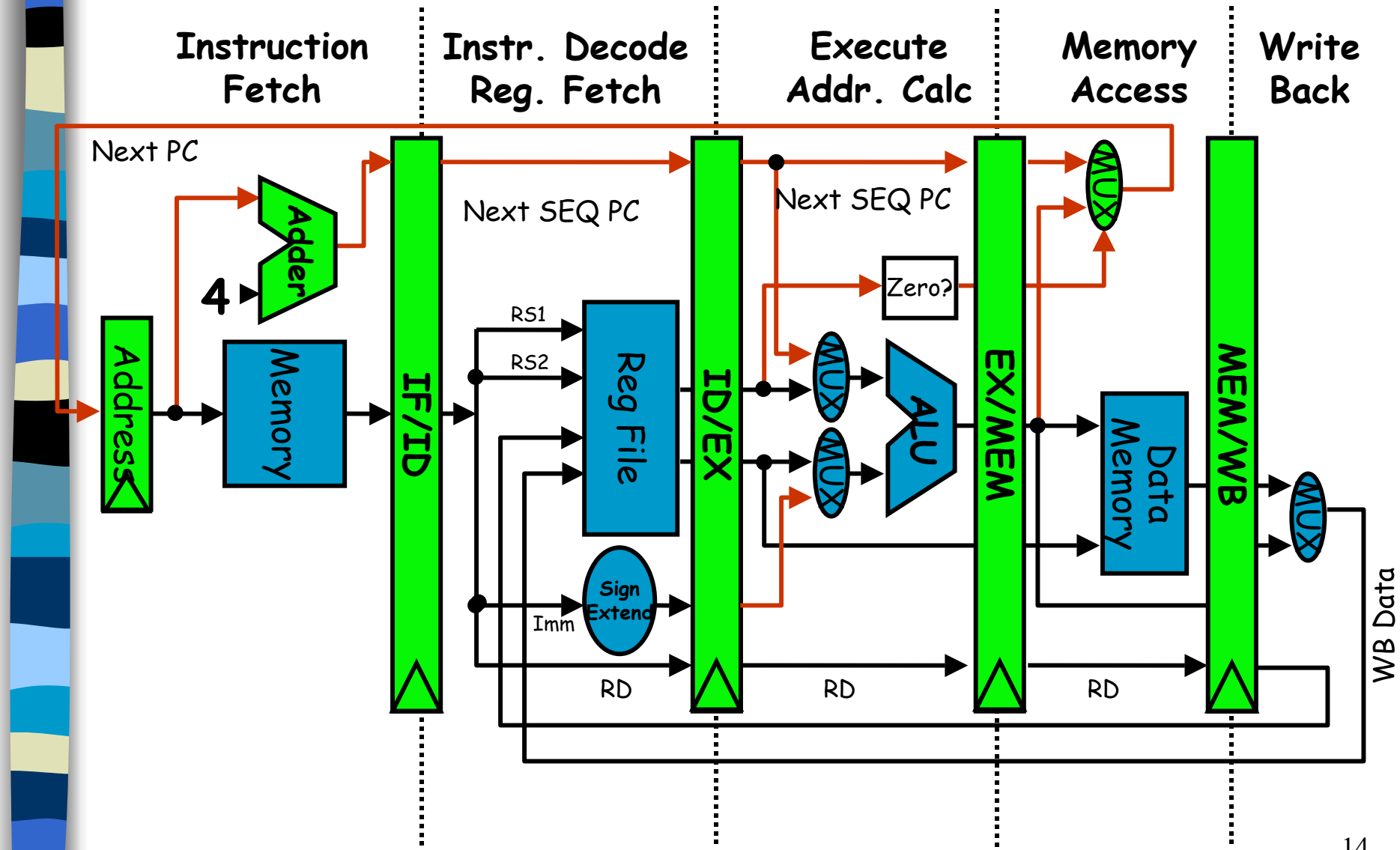
```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

Fast code:

```
LW    Rb,b
LW    Rc,c
LW    Re,e
ADD   Ra,Rb,Rc
LW    Rf,f
SW    a,Ra
SUB   Rd,Re,Rf
SW    d,Rd
```

Branch Operation

Figure 3.4, Page 134 , CA:AQA 2e



Control Hazard on Branches
Three Stage Stall

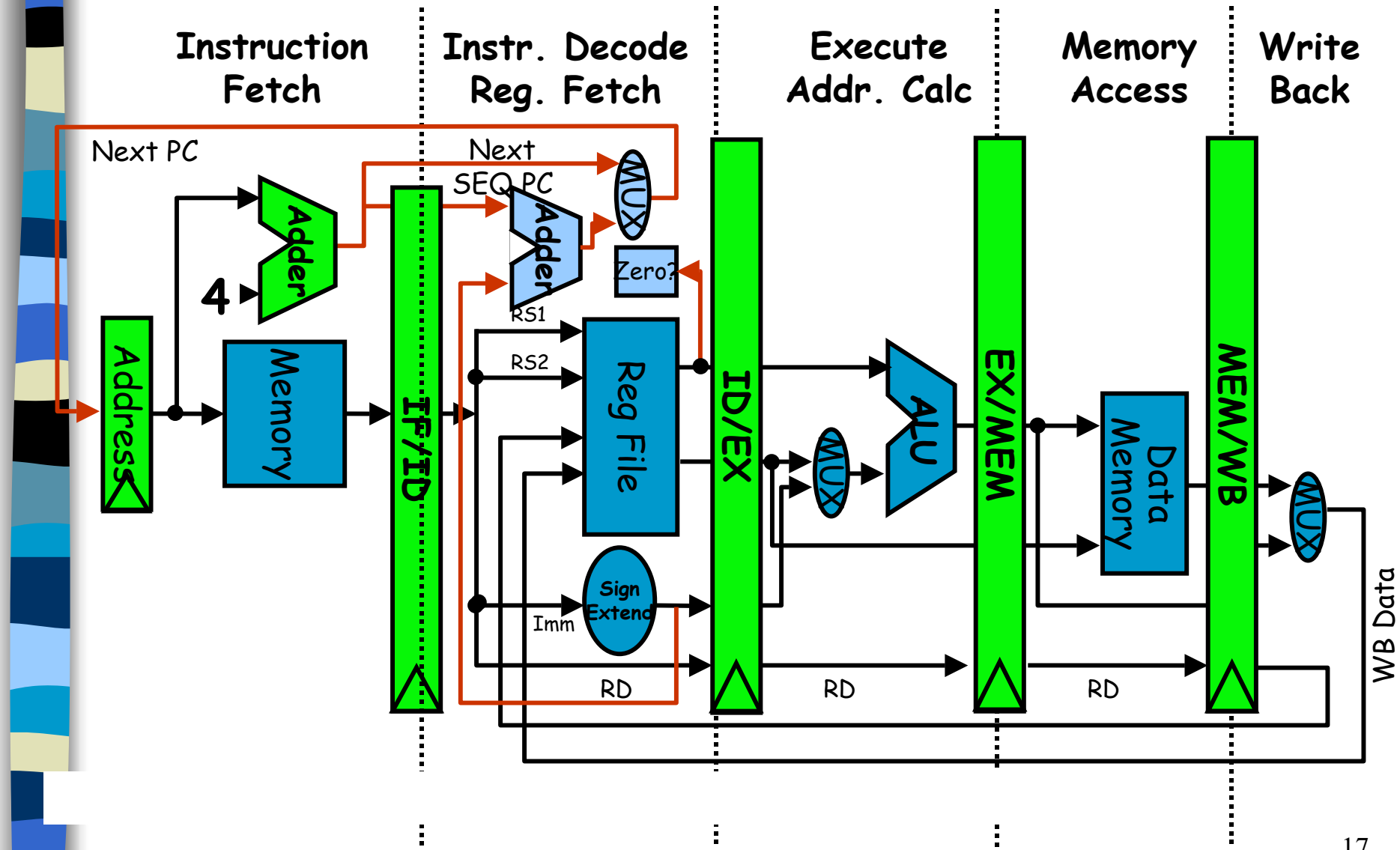
Branch Inst	IF	ID	EX	MEM	WB					
Br successor		IF	S	S	IF	ID	EX	MEM	WB	
Br successor+1						IF	ID	EXE		

Branch Stall Impact

- If CPI = 1, 30% branch, Stall 3 cycles => new CPI = 1.9!
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- DLX branch tests if register = 0 or ≠0
- DLX Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

New Pipelined DLX Datapath

Figure 3.22, page 163, CA:AQA 2/e



Control Hazard on Branches
One Stage Stall

Branch Inst	**IF**	**ID**	**EX**	**MEM**	**WB**				
Br successor		**IF**	**IF**	**ID**	**EX**	**MEM**	**WB**		
Br successor+1				**IF**	**ID**	**EXE**			

Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

- Execute successor instructions in sequence
- squash instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% DLX branches not taken on average
- PC+4 already calculated, so use it to get next instruction

How many stall cycles?

Branch is not taken

Branch Inst	**IF**	**ID**	**EX**	**MEM**	**WB**				
Br successor		**IF**	**ID**	**EX**		**MEM**	**WB**		
Br successor+1			**IF**	**ID**		**EXE**			

Zero Cycle Branch Penalty!

Branch is taken

Branch Inst	**IF**	**ID**	**EX**	**MEM**	**WB**														
Inst i+1		**IF**	**S**		**S**		**S**		**S**										
Br target				**IF**		**ID**		**EX**		**MEM**		**WB**							
Br target+1							**IF**		**ID**		**EXE**			**MEM**	**WB**				

One Cycle Branch Penalty!

Four Branch Hazard Alternatives

#3: Predict Branch Taken

- 53% DLX branches taken on average
- But haven't calculated branch target address in DLX
  - DLX still incurs 1 cycle branch penalty
  - Other machines: branch target known before outcome

Four Branch Hazard Alternatives

#4: Delayed Branch

– schedule instructions into branch-delay slots

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

Branch delay of length *n*

– 1 slot delay allows proper decision and branch target address in 5 stage pipeline

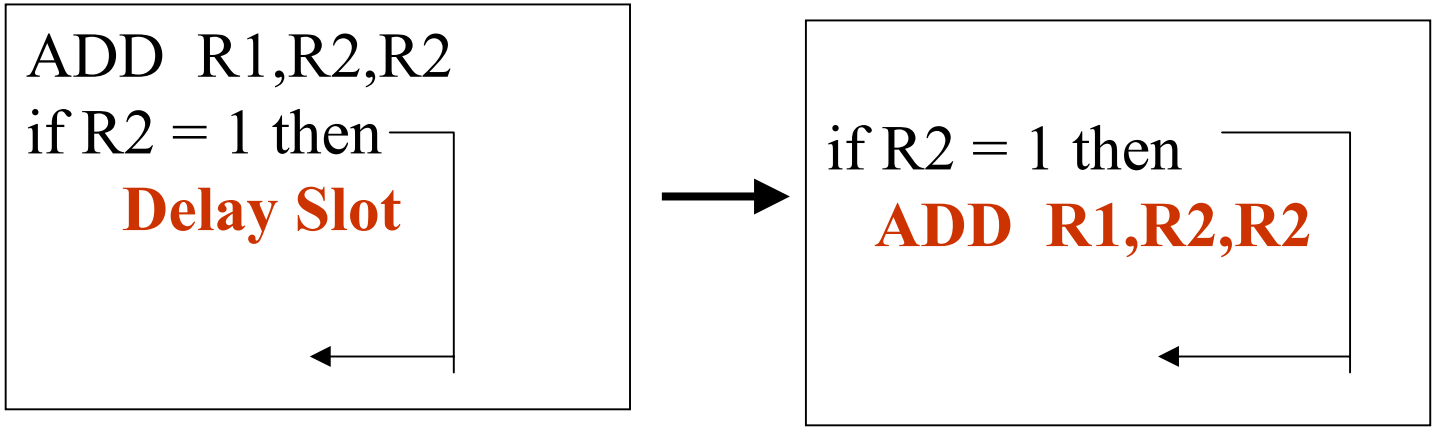
– DLX uses this

Delayed Branch

- Where to get instructions to fill branch delay slot?
  - Before branch instruction
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken

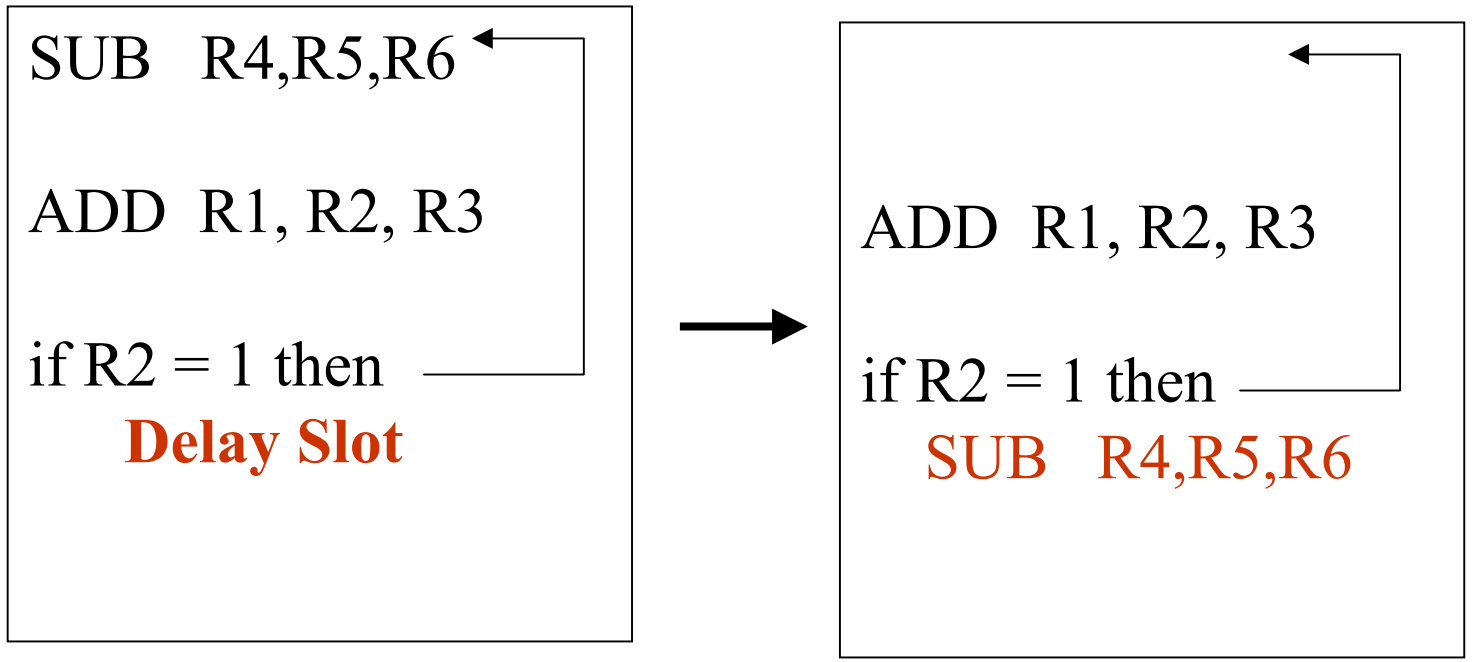
Scheduling the branch-delay slot

- Before branch instruction



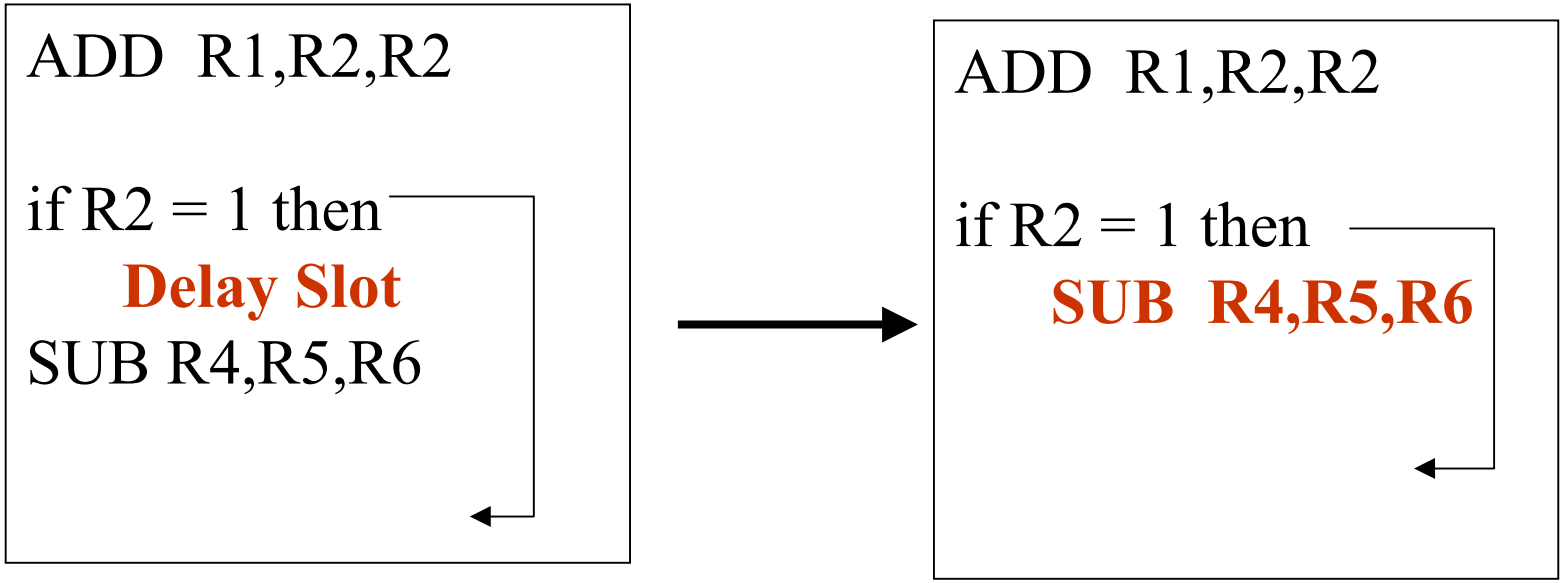
Scheduling the branch-delay slot

- From the target address



Scheduling the branch-delay slot

- From fall through



Instruction Scheduling

- Branch is untaken

Branch Inst	IF	ID	EX	MEM	WB				
Branch delay inst (i+1)		IF	ID	EX	MEM	WB			
Inst i+2			IF	ID	EX	MEM	WB		
Inst i+3				IF	ID	EXE			

- The branch is taken

Branch Inst	IF	ID	EX	MEM	WB				
Branch delay inst (i+1)		IF	ID	EX	MEM	WB			
Branch target			IF	ID	EX	MEM	WB		
Branch target+1				IF	ID	EXE			

Limitation on delayed-branch scheduling

- The restrictions on the instructions that are scheduled into the delay slots
  - It must be OK to execute the instruction in the delayed slot when the prediction goes into the unpredicted direction
  - Canceling branch
- The compiler ability to predict at compile time whether a branch is likely to be taken or not
  - Branch prediction

Evaluating Branch Alternatives

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

Scheduling	*Penalty unconditional*	*untaken*	*taken*
Stall pipeline	1	1	1
Predict taken	1	1	1
Predict not taken	1	0	1

frequency: unconditional branch 4%
taken branch 16%
untaken branch 10%

Speedup from different branch scheme?

Static Branch Prediction

- Improving strategy on placing instructions in the delay slot
  - Add direction to the prediction
    - Backward branch predicted taken: typically due to iteration and are usually taken
    - Forward branch untaken
- Profile-based prediction
  - Record branch behavior, predict branch based on earlier run
  - Tends to be a big win over direction-based methods

Pipelining Complications : Exceptions

- Exceptions: 5 instructions executing in 5 stage pipeline
  - How to stop the pipeline?
  - How to restart the pipeline?ns
  - Who caused the interrupt?

- Exceptions in DLX

Stage	*Problem interrupts occurring*
IF	Page fault on instruction fetch; misaligned memory access; memory-protection violation
ID	Undefined or illegal opcode
EX	Arithmetic interrupt
MEM	Page fault on data fetch; misaligned memory access; memory-protection violation

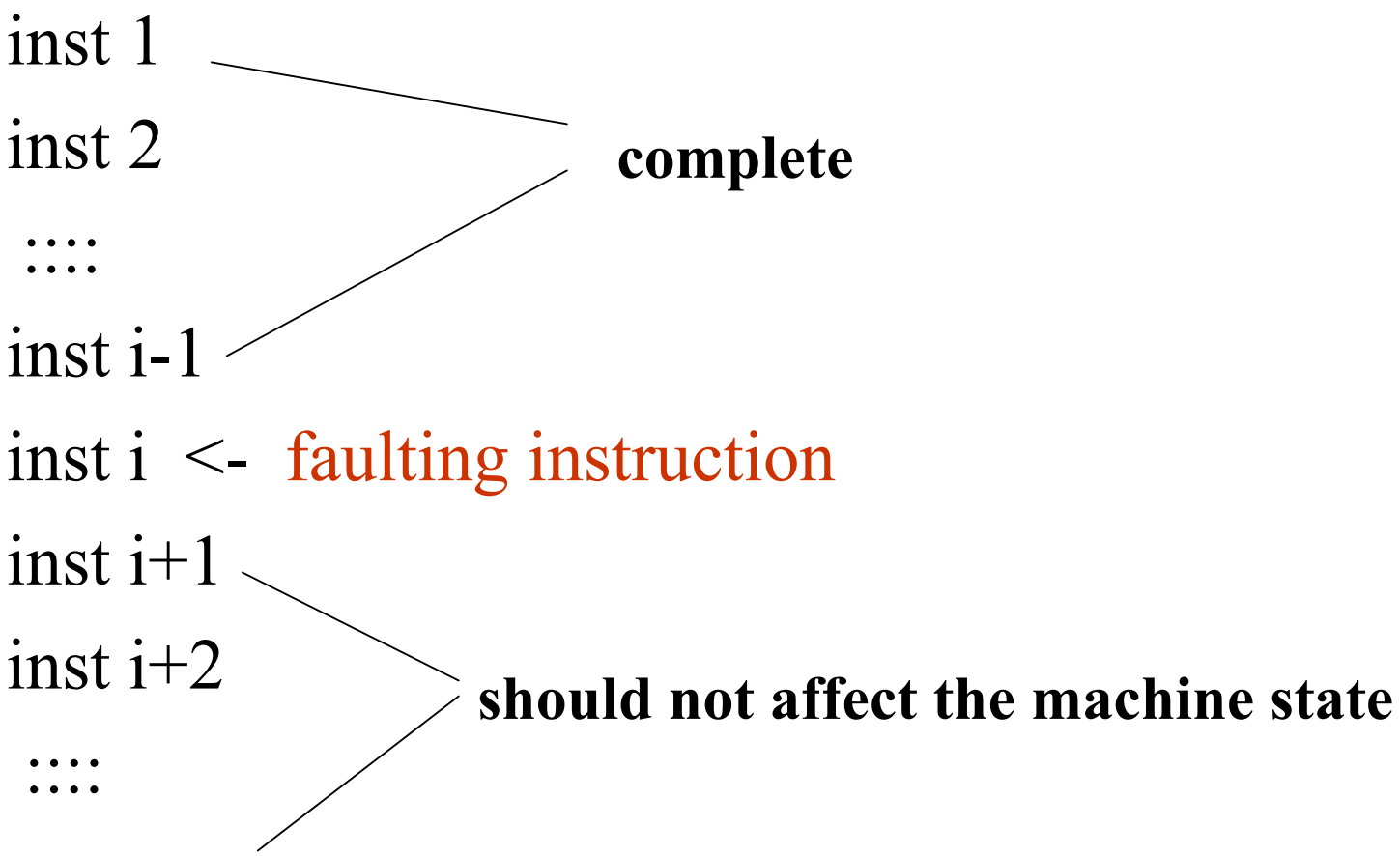
Biggest Problem: Within & Resume

- Save & Restore State
  - Page fault
    - shutdown the pipeline
    - save the state
    - start the OS page fault handler
    - restarting at the correct PC

What does DLX do?

- Force a trap instruction into the pipe on the next IF
- Turn off all writes for the faulting and following instructions
- Let all preceding instructions complete if they can
- Save the restart PC
- Call the OS to handle the exception
- return to the user code

Precise Exceptions

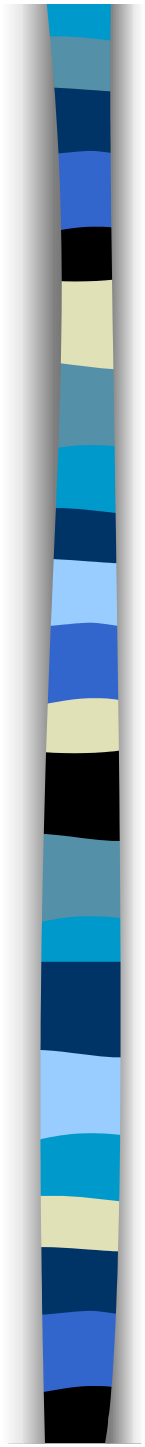


Exceptions in DLX

LW	IF	ID	EX	MEM	WB	
ADD		IF	ID	EX	MEM	WB

Can exception occur out of order?

LW get a data page fault & Add get a instruction page fault !



How to implement precise exception in DLX

- Exception status vector
  - turn off all writes once an exception indication is set
- Handle exception in the WB stage

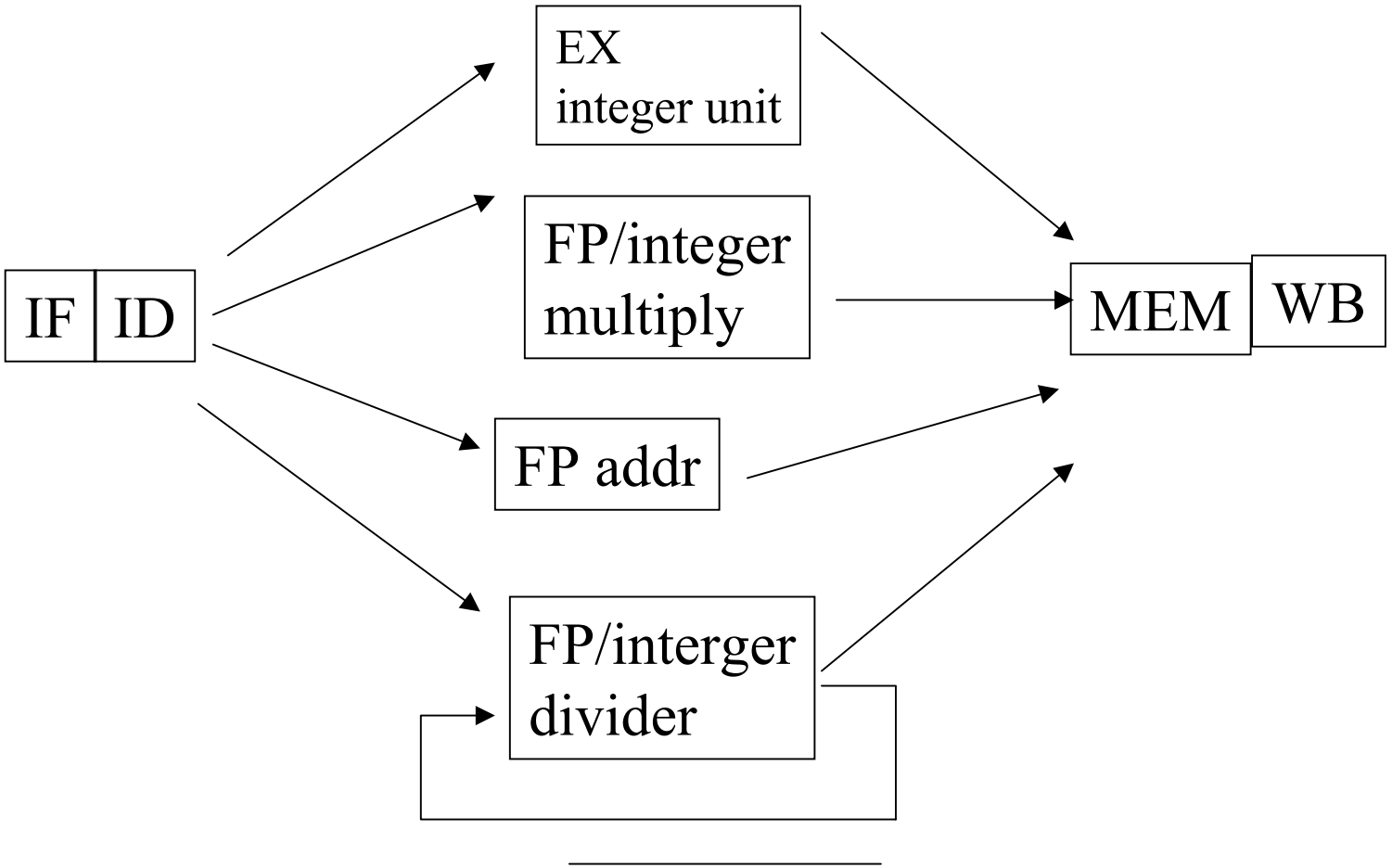
It is more difficult to implement precise exceptions for floating point operations because they may complete out of order

ISA Complications in Precise Handling

- Complex instruction set may change the CPU state in the middle of the instruction execution
  - auto-increment causes register change during instruction execution

ADD R1, (R2)+ : Reg[R1]<-Reg[R1]+Mem[Reg[R2]];
Reg[R2]<-Reg[R2]+d

A Multi-Cycle DLX Pipe

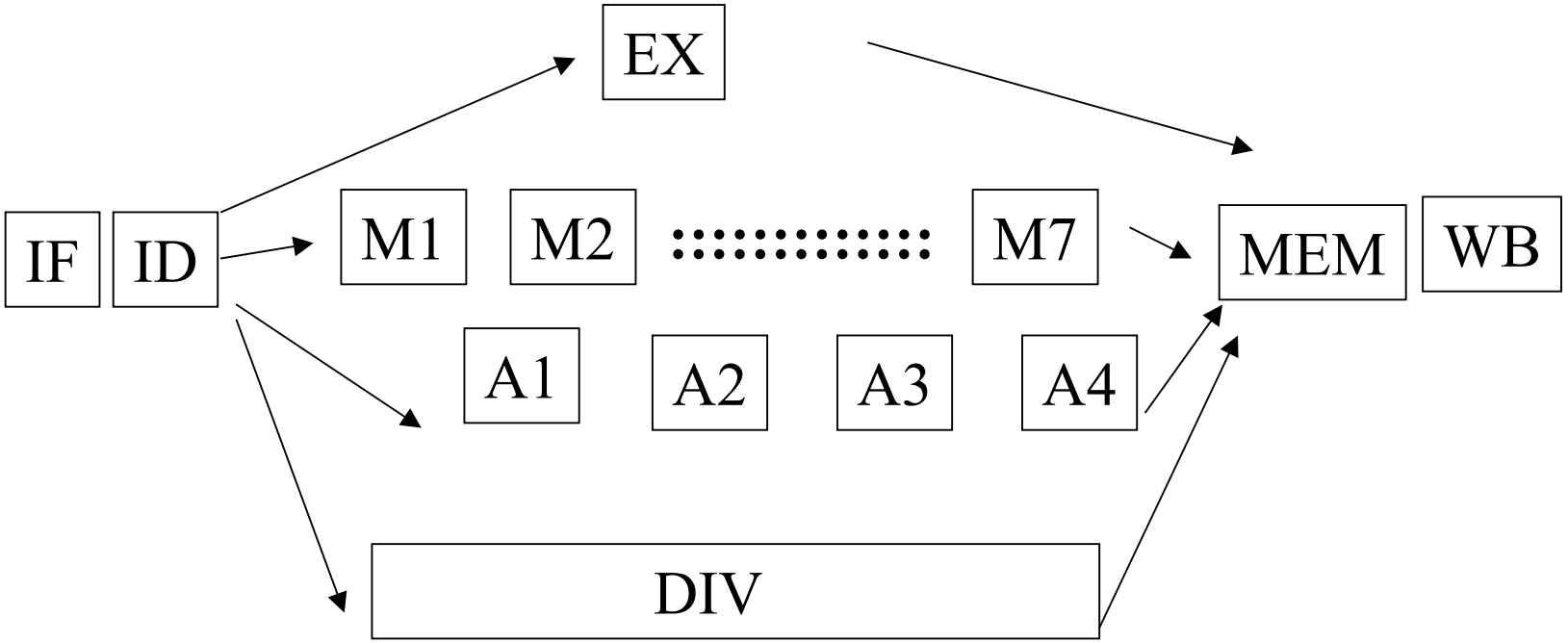


Latency v.s. Initiation Interval

- Latency: # cycles between an instruction producing a result and an instruction using the result
- Initiation/repeat interval: # cycles that must elapse between issuing two operations of a given type

Functional Unit	Latency	Initiation Interval
Integer ALU	0	1
Data memory	1	1
(integer & FP loads)	3	1
FP add	3	1
FP/integer multiply	6	1
FP/integer divide	24	24

DLX Pipe



Instruction Scheduling

- What is the instruction scheduling for the following code sequence?

MULTD :::
ADDD :::
LD :::
SD :::

Assume they are independent

Instruction scheduling

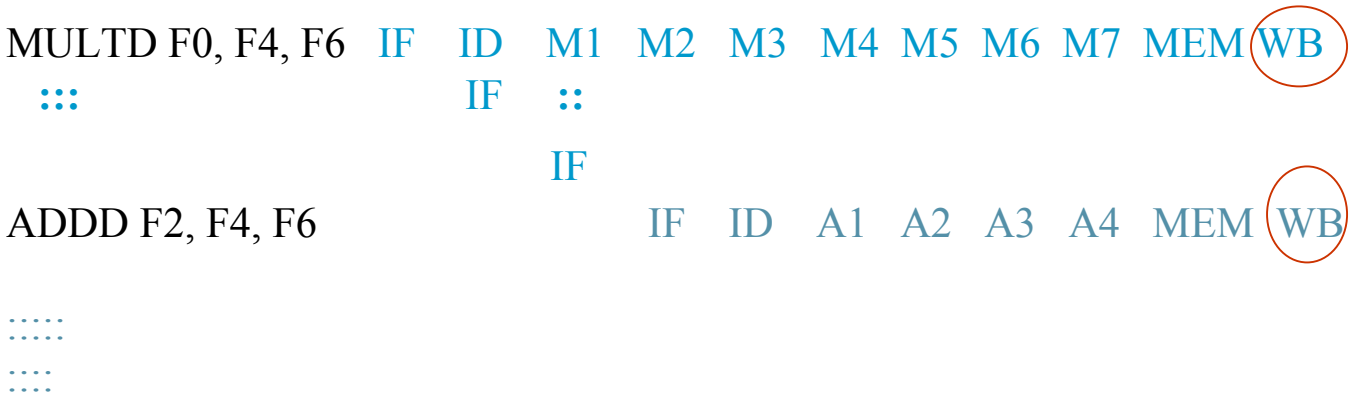
MULTD	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
ADDD		IF	ID	A1	A2	A3	A4	MEM	WB		
LD			IF	ID	EX	MEM	WB				
SD				IF	ID	EX	MEM	WB			

- **Introducing new hazards**
- **Instructions complete out of order -** complicates exception handling

New Hazard and Forwarding Problem

- Structure hazards increase
  - Unpiped divide causes a big 24-cycle zone
  - # of register writes in a cycle goes up
- WAW hazards now possible since instructions now reach WB out of order (how about WAR?)
- RAW hazards will be more frequent, due to longer latency instructions

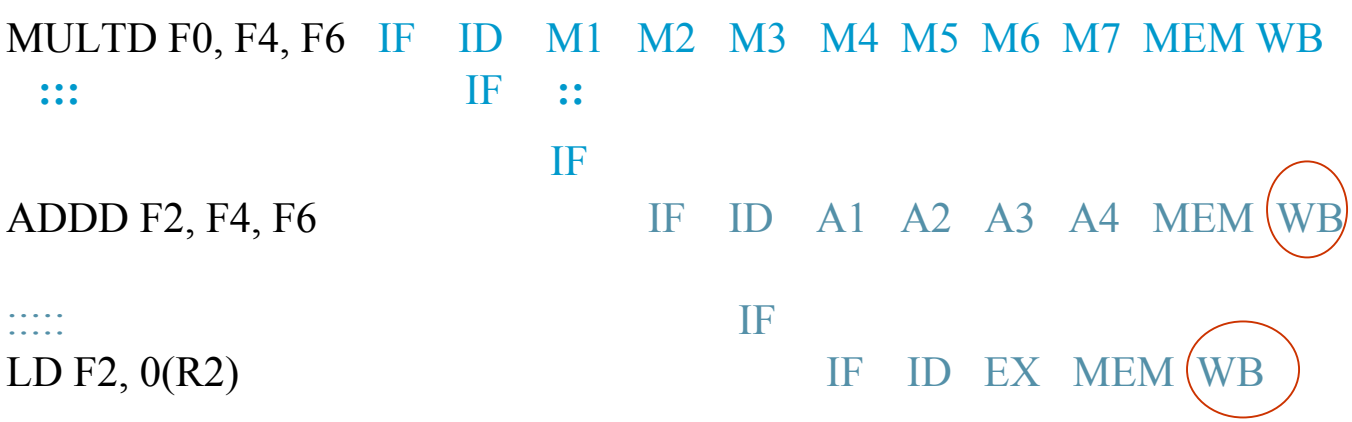
Dealing with Structural Hazards



Option 1: Track the use of the write ports, detect structural hazard in the ID stage

Option 2: Stall a conflicting instruction at MEM entry

Dealing with WAW Hazards



Option 1:Delay the LD until the ADDD enters MEM
Option 2: Stamp out the result of the ADDD

RAW Hazards

- What is the instruction scheduling for the following code fragment?

```
LD      F4, 0(R2)
MULTD   F0,  F4,  F6
ADDD    F2, F0, F8
SD      0(R2), F2
```

Precise Exception Problems

- Consider

```
DIVF    F0, F2, F4
ADDF    F6, F6, F8
SUBF    F10, F10, F12
```

Complication : DIVF may cause an exception when the ADDF has completed

Dealing with Out-of-Order Completion

- Ignore the problem or restrict the overlap in an FP pipeline
- Buffer the results of an operation until all the operations issued earlier are completed.
- Go imprecise with SW fixup

  Instruction 1 -- fault
  instruction 2 to instruction (n-1) -- not completed
  instruction i -- complete

  Let the exception handler executes instruction 1 to n-1, then resume execution at instruction i+1

- Stall issue until previous one guarantee to finish

Summary of Pipelining Basics

- Hazards limit performance
  - Structural: need more HW resources
  - Data: need forwarding, compiler scheduling
  - Control: early evaluation & PC, delayed branch, prediction
- Increasing length of pipe increases impact of hazards; pipelining helps instruction bandwidth, not latency
- Interrupts, Instruction Set, FP makes pipelining harder
- Compilers reduce cost of data and control hazards
  - Load delay slots
  - Branch delay slots
  - Branch prediction