COMP60001/COMP70086
Advanced Computer Architecture
Chapter 1.2

Pipelining: a quick review of introductory computer architecture

Objective: bring everyone up to speed, and also establish some key ideas that will come up later in the course in more complicated contexts

October 2025
Paul H J Kelly

These lecture notes are partly based on the course text, Hennessy and Patterson's *Computer Architecture, a quantitative approach (6th ed), and on the lecture slides of David Patterson's Berkeley course (CS252)*

Course materials online on
https://scientia.doc.ic.ac.uk/2526/modules/60001/materials and
https://www.doc.ic.ac.uk/~phjk/AdvancedCompArchitecture/aca20/

Pre-requisites

- This is a second-to-third-level computer architecture course
  - We aim to get from what you'd learn in DoC's first year up to understanding the design, and design alternatives, in current commercially-available processors
  - It's a stretch but my job is to help!

- You *can* take this course provided you're prepared to catch up if necessary
  - I will introduce all the key ideas, but if they are new to you, you will need to do some homework!

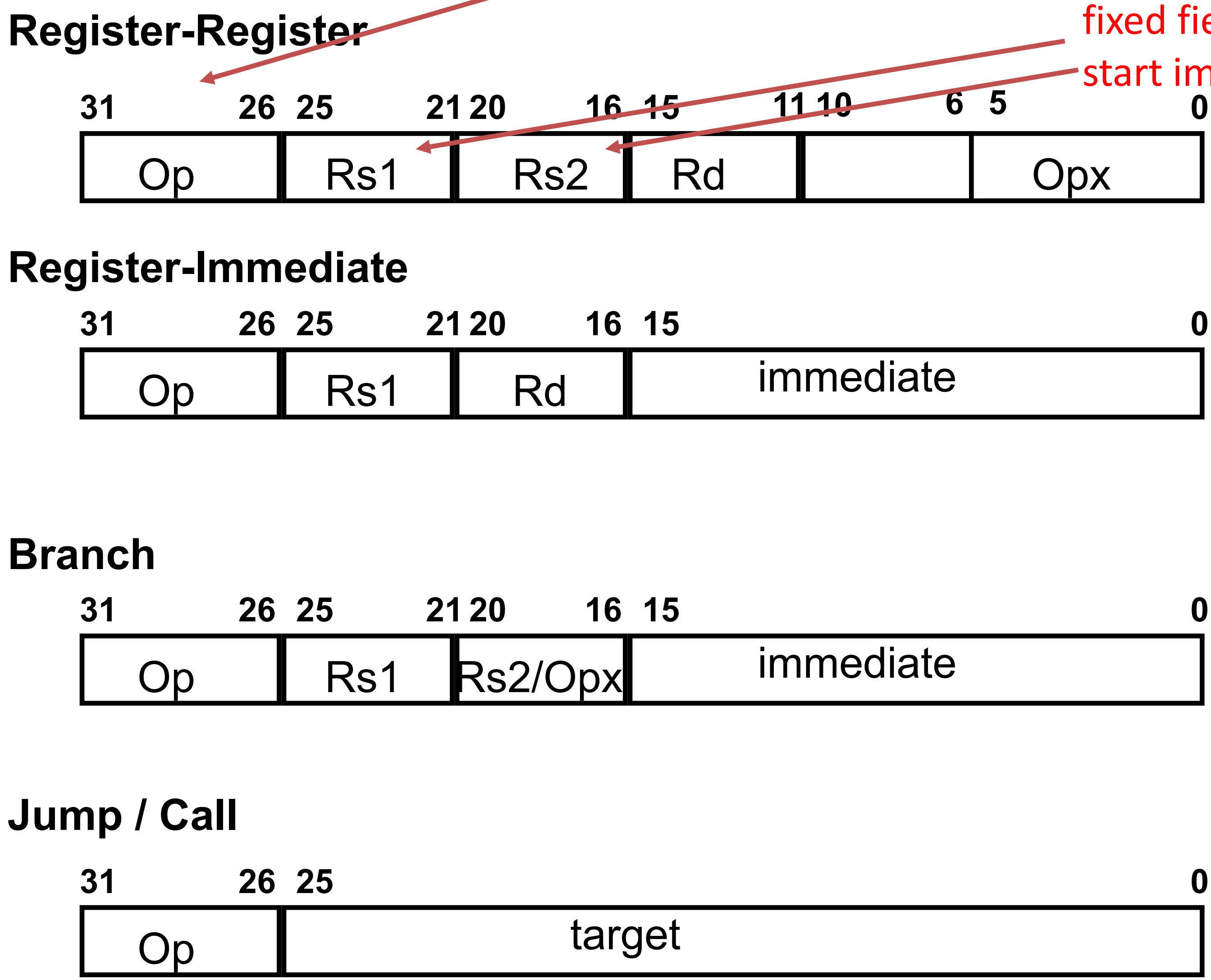
- We are keen to help you succeed – and I count on you to ask questions – both live and on EdStem

- This lecture introduces pipelining – we're looking for the issues in simple designs that help us understand the more complicated designs that are coming up

Example: MIPS

Opcode specifies how other fields will be interpreted

5-bit register specifier at fixed field so access can start immediately



Example: MIPS

Opcode specifies how other fields will be interpreted

5-bit register specifier at fixed field so access can start immediately

Register-Register

31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Op	Rs1	Rs2	Rd		Opx

ADD R8,R6,R4 // R8 ← R6+R4

Register-Immediate

31 – 26	25 – 21	20 – 16	15 – 0
Op	Rs1	Rd	immediate

LW R2, 100(R3) // R2 ← Memory[R3+100]
SW R5, 100(R6) // Memory[R6+100] ← R5
ADDI R4, R5, 50 // R4 ← R5+signExtend(50);

Branch

31 – 26	25 – 21	20 – 16	15 – 0
Op	Rs1	Rs2/Opx	immediate

BEQ R5, R7, 25 // if R5=R7
// then PC ← PC+4+25*4
// else PC ← PC+4

Jump / Call

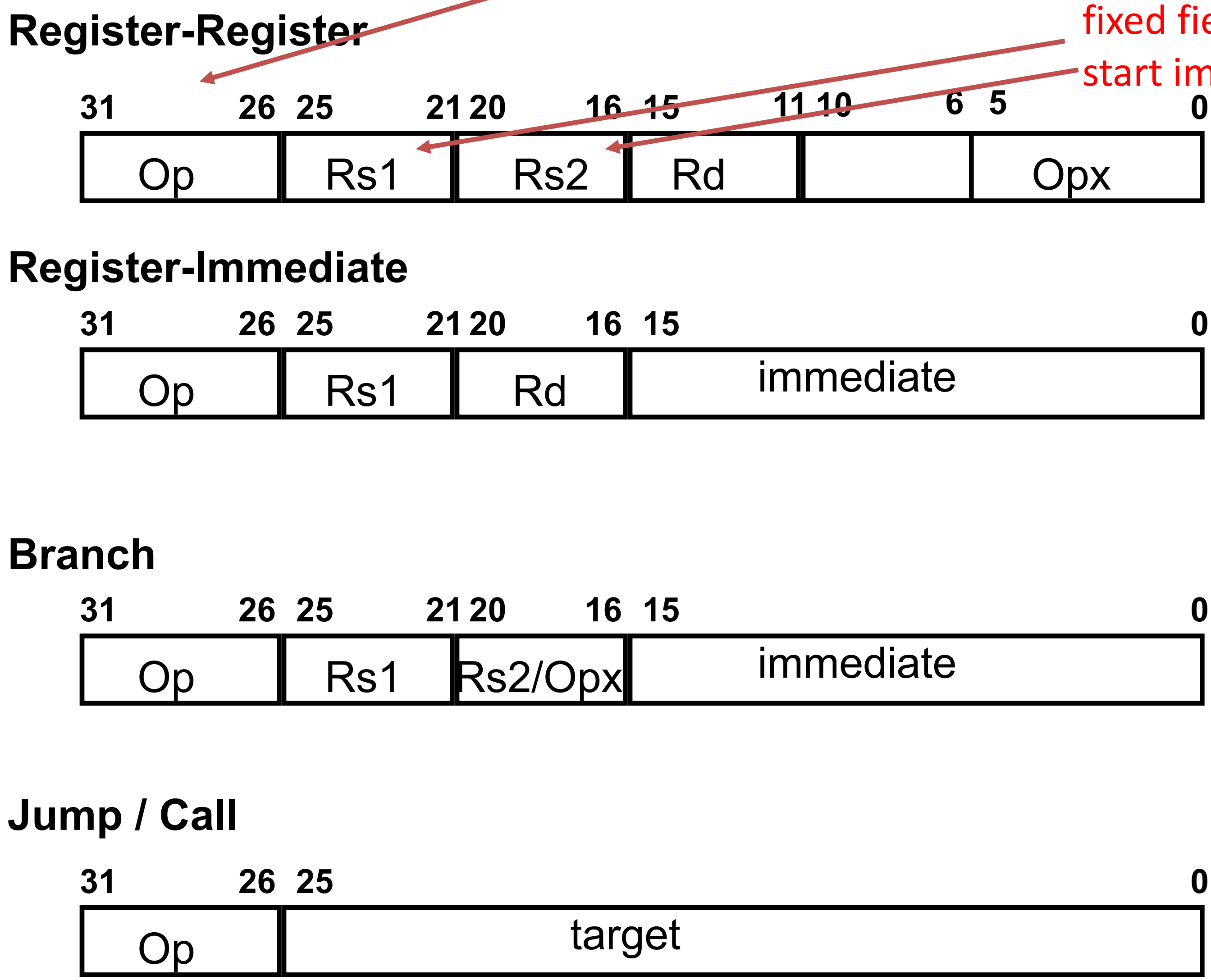
31 – 26	25 – 0
Op	target

J 200000 // PC ← 200000*4

Example: MIPS

Opcode specifies how other fields will be interpreted

5-bit register specifier at fixed field so access can start immediately



Q: How many registers can we address?

Q: What is the largest signed immediate operand for "ADD R1,R2,X"?

Q: What range of addresses can a conditional branch jump to?

A machine to execute these instructions

- To execute this instruction set we need a machine that fetches them and does what each instruction says
- A "universal" computing device – a simple digital circuit that, with the right code, can compute *anything*
- Something like:

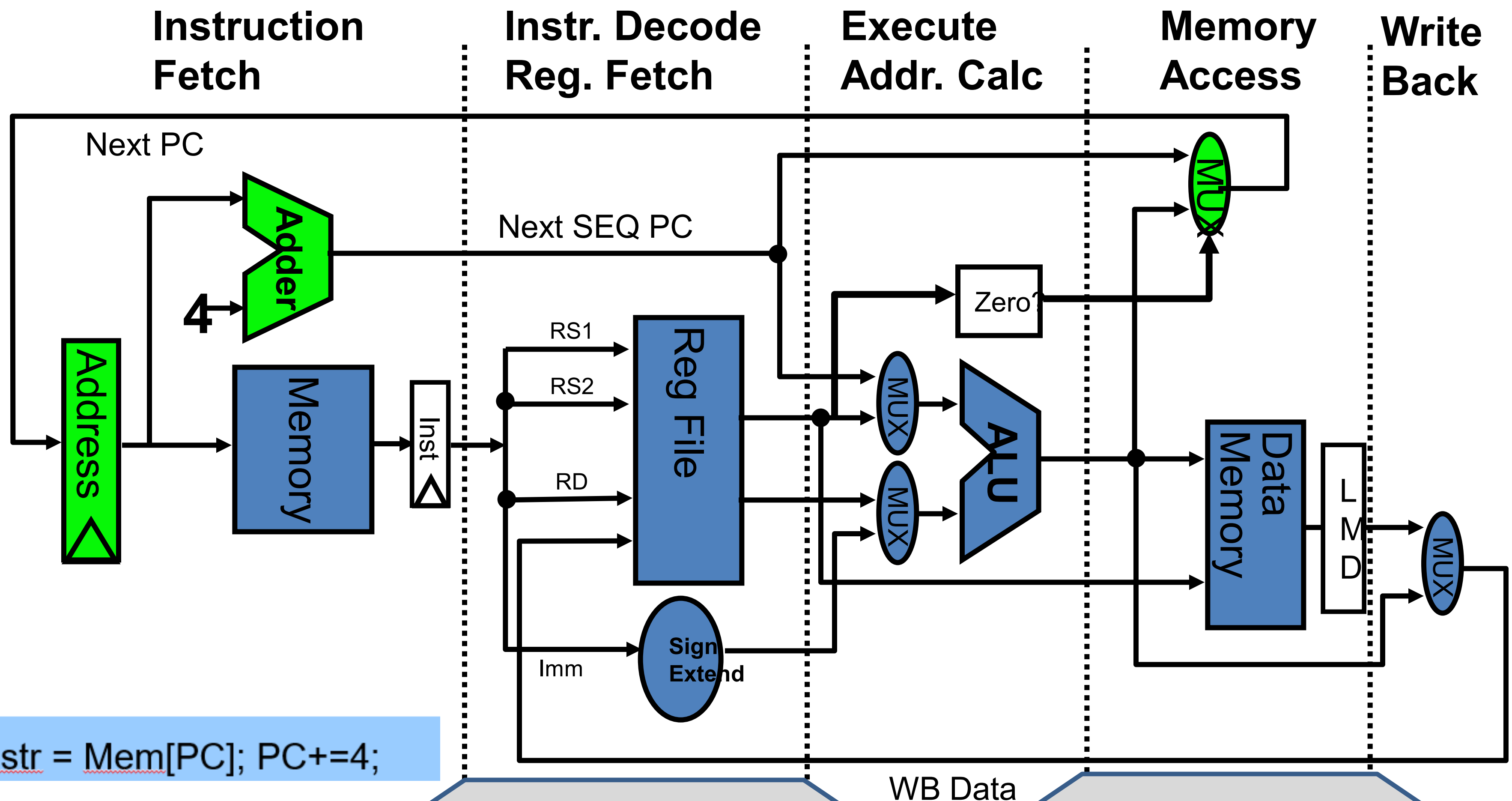
```
Instr = Mem[PC]; PC+=4;

rs1 = Reg[Instr.rs1];
rs2 = Reg[Instr.rs2];
imm = SignExtend(Instr.imm);

Operand1 = if(Instr.op==BRANCH) then PC else rs1;
Operand2 = if(immediateOperand(Instr.op)) then imm else rs2;
res = ALU(Instr.op, Operand1, Operand2);

switch(Instr.op) {
case BRANCH:
  if (rs1==0) then PC=PC+imm*4; continue;
case STORE:
  Mem[res] = rs1; continue;
case LOAD:
  lmd = Mem[res];
}
Reg[Instr.rd] = if (Instr.op==LOAD) then lmd else res;
```

5 Steps of MIPS Datapath



Instr = Mem[PC]; PC+=4;

rs1 = Reg[Instr.rs1];
rs2 = Reg[Instr.rs2];
imm = SignExtend(Instr.imm);

Operand1 = if(Instr.op==BRANCH) then PC else rs1;
Operand2 = if(immediateOperand(Instr.op)) then imm else rs2;
res = ALU(Instr.op, Operand1, Operand2);

```
switch(Instr.op) {
case BRANCH:
  if (rs1==0) then PC=PC+imm*4; continue;
case STORE:
  Mem[res] = rs1; continue;
case LOAD:
  lmd = Mem[res];
```

Reg[Instr.rd] = if (Instr.op==LOAD) then lmd else res;

Figure 3.1, Page 130, CA:AQA 2e

5 Steps of MIPS Datapath

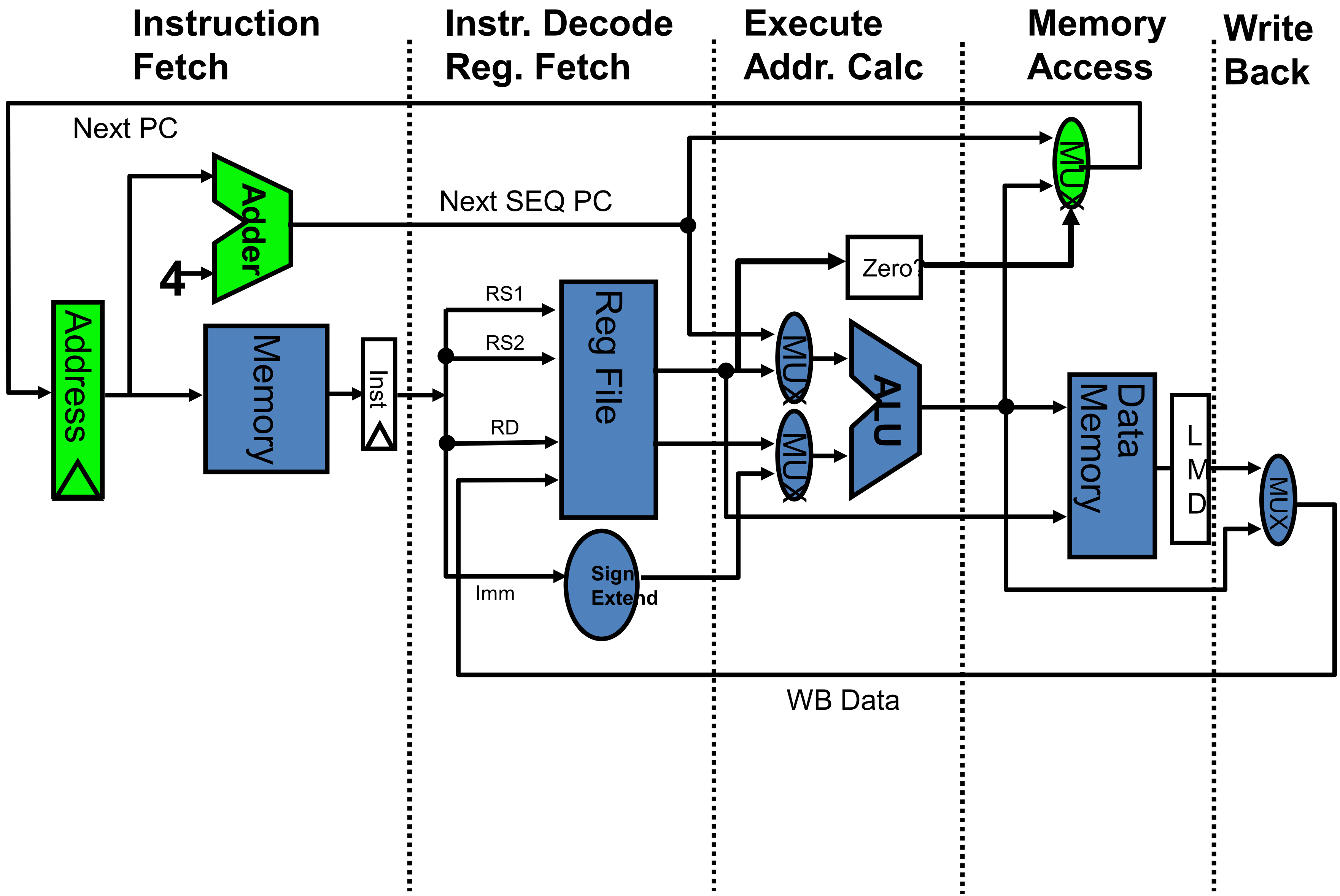


Figure 3.1, Page 130, CA:AQA 2e

5-stage MIPS pipeline with pipeline buffers

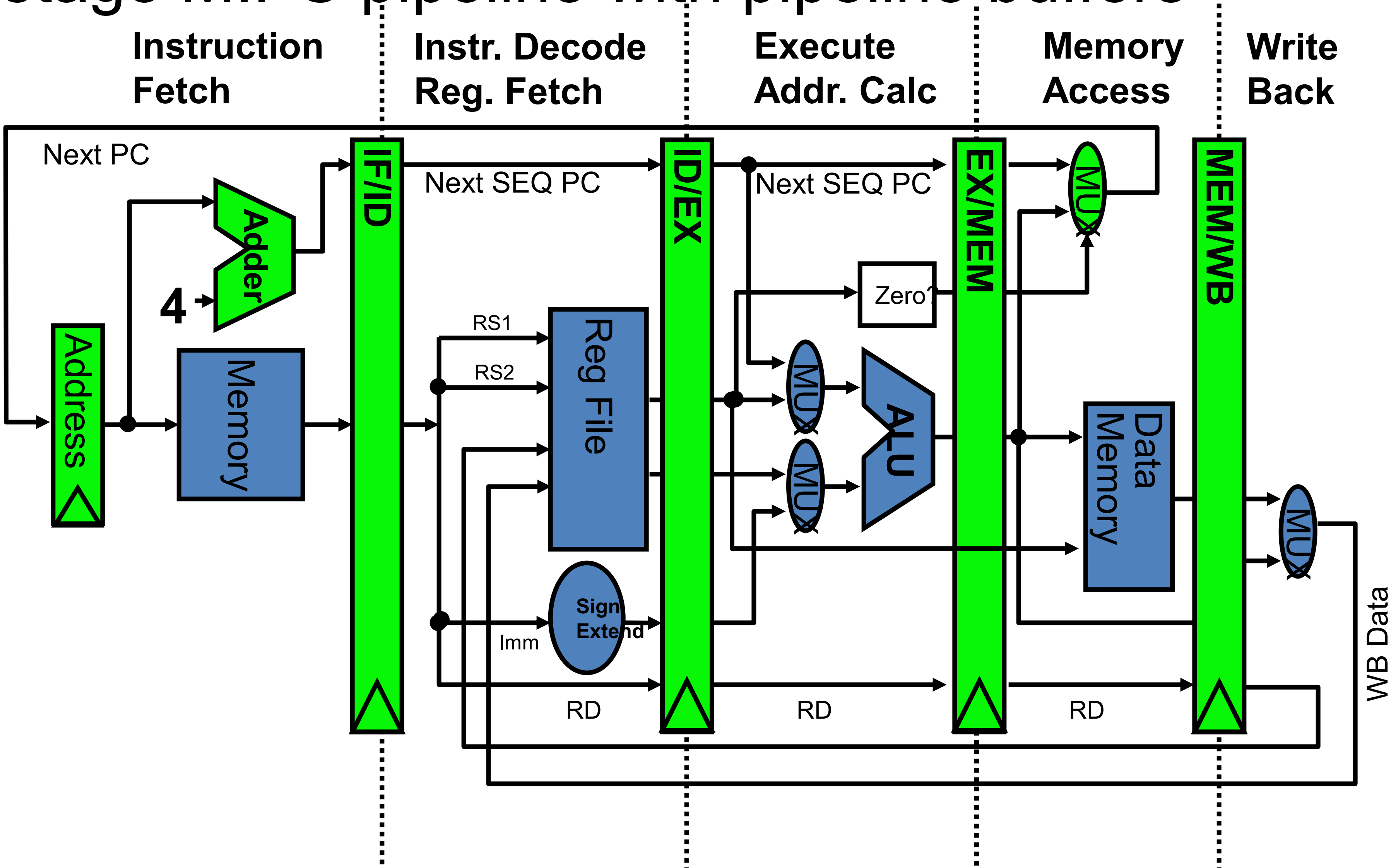
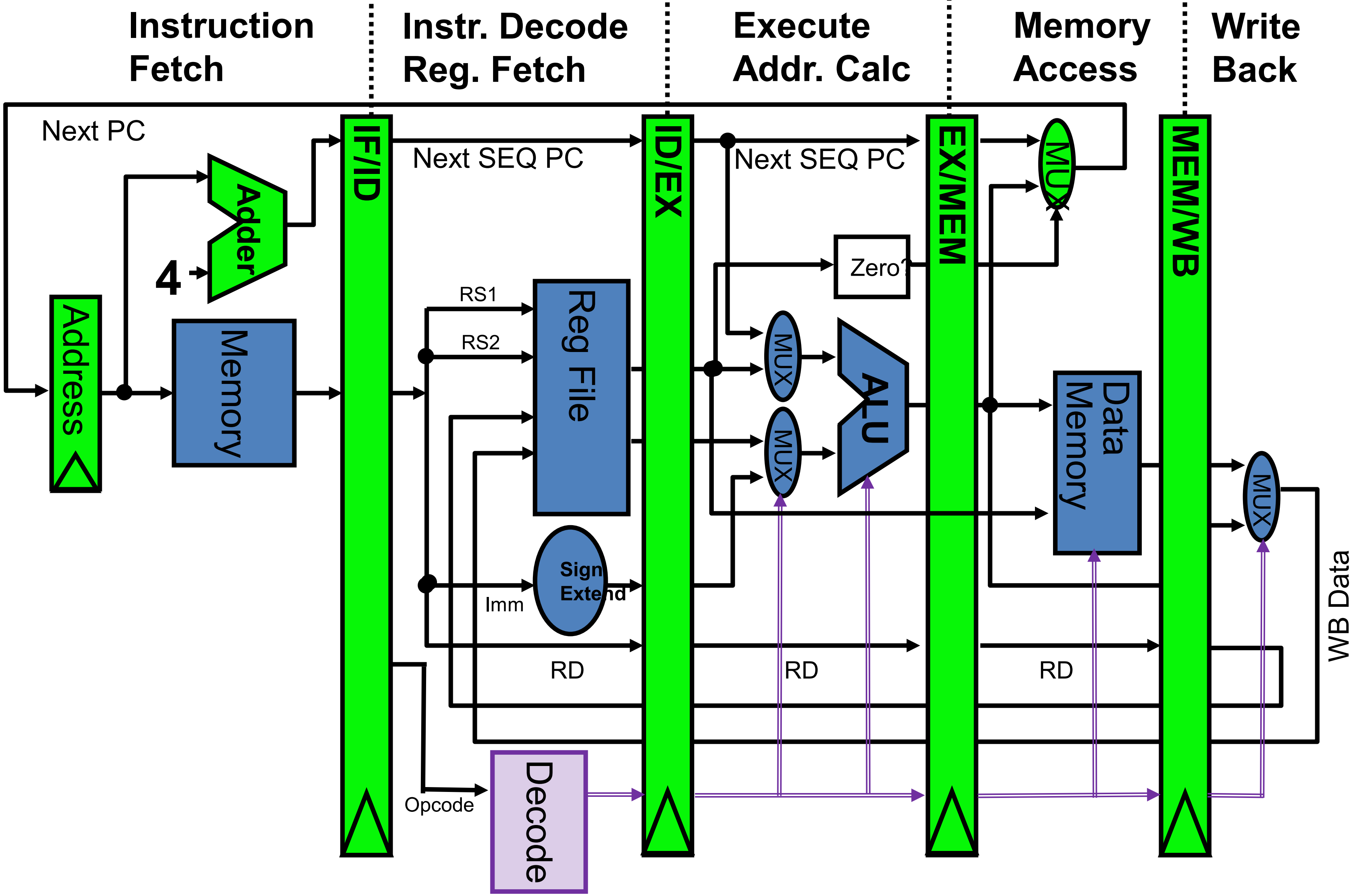


Figure 3.4, Page 134 , CA:AQA 2e

5-stage MIPS pipeline with pipeline buffers



- **Data stationary control**
  - **Control signals are needed to configure the MUXes, ALU, read/write**
  - **Carried with the corresponding instruction along the pipeline**

Figure 3.4, Page 134 , CA:AQA 2e

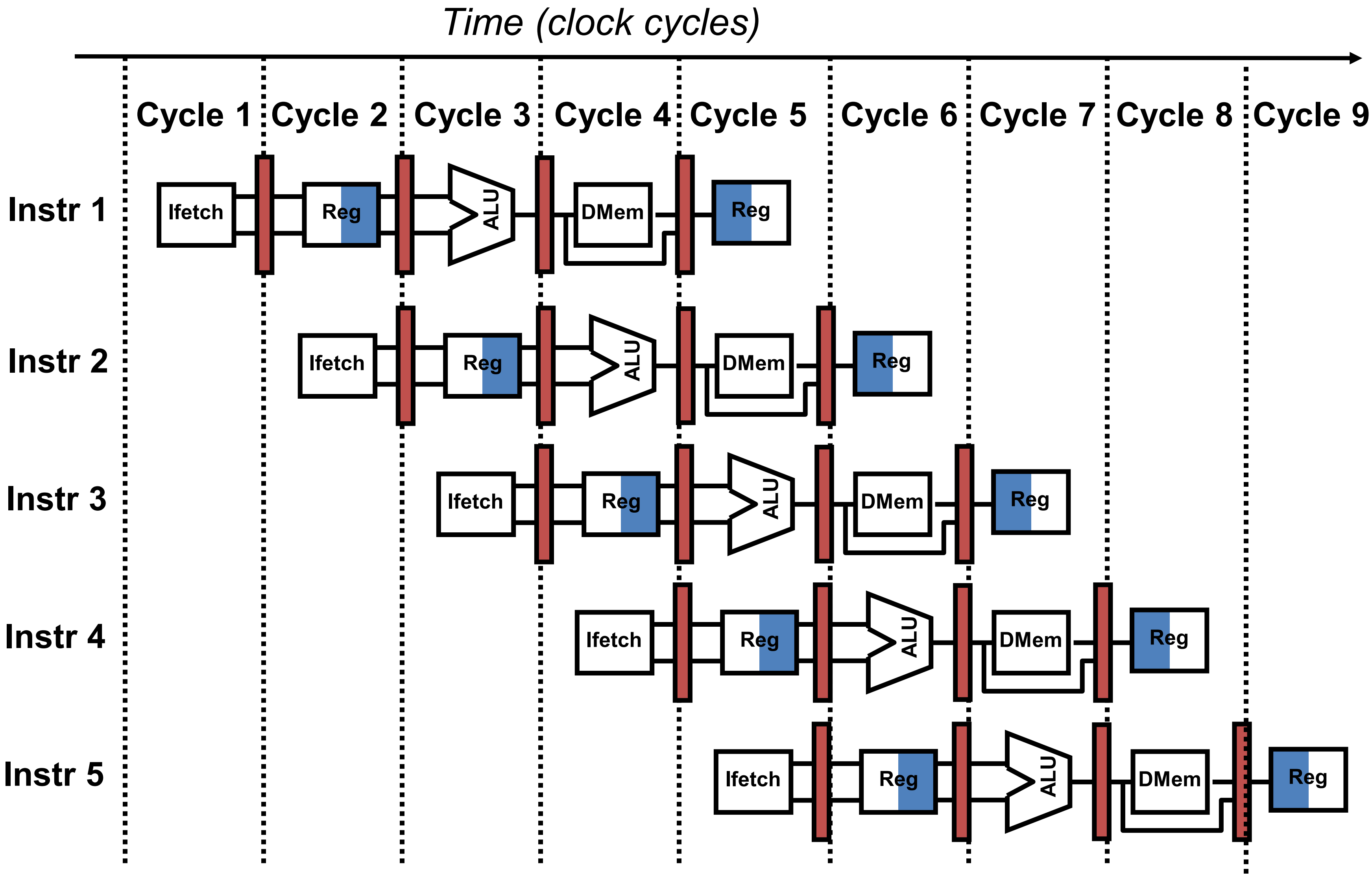
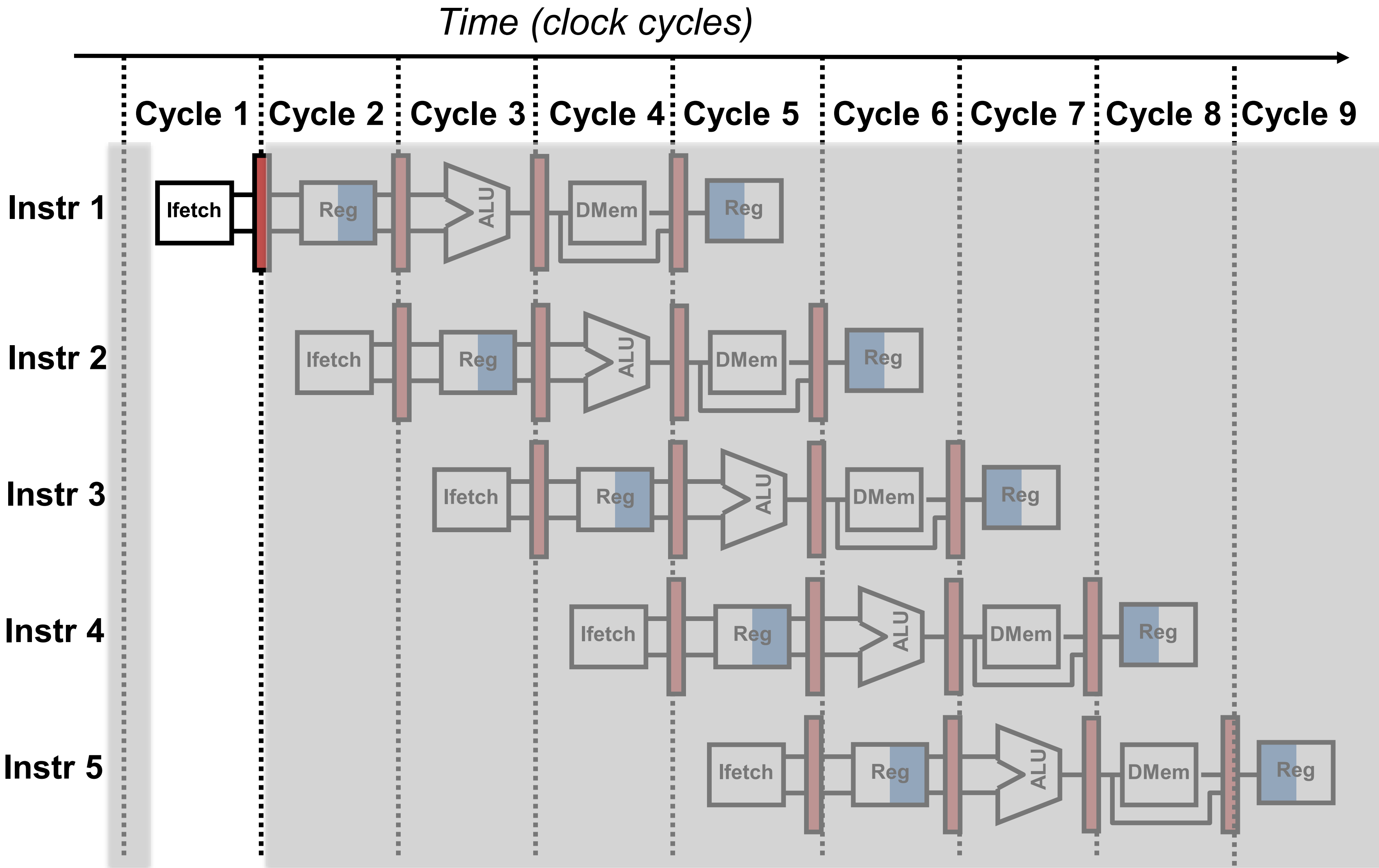
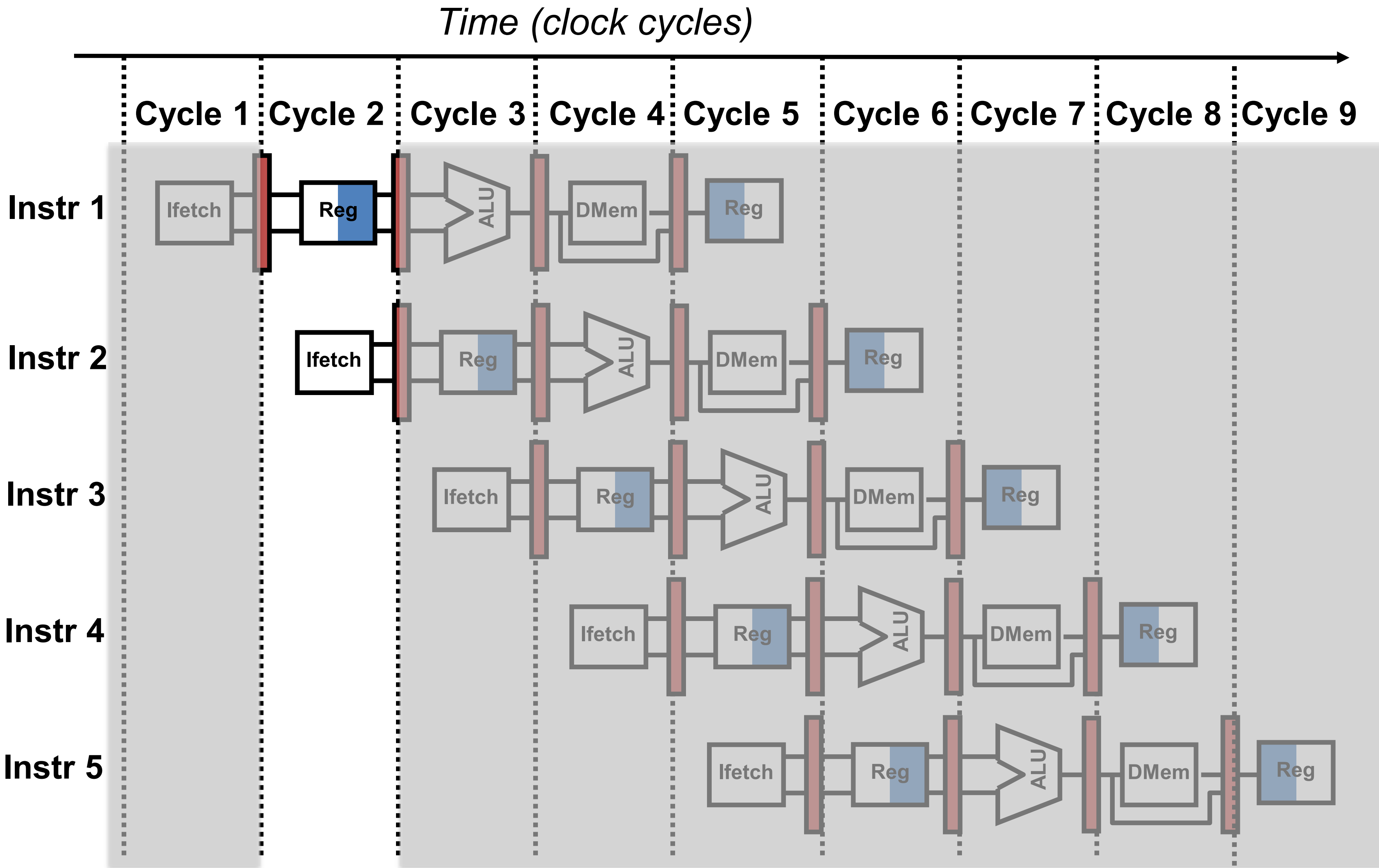


Figure 3.3, Page 133 , CA:AQA 2e

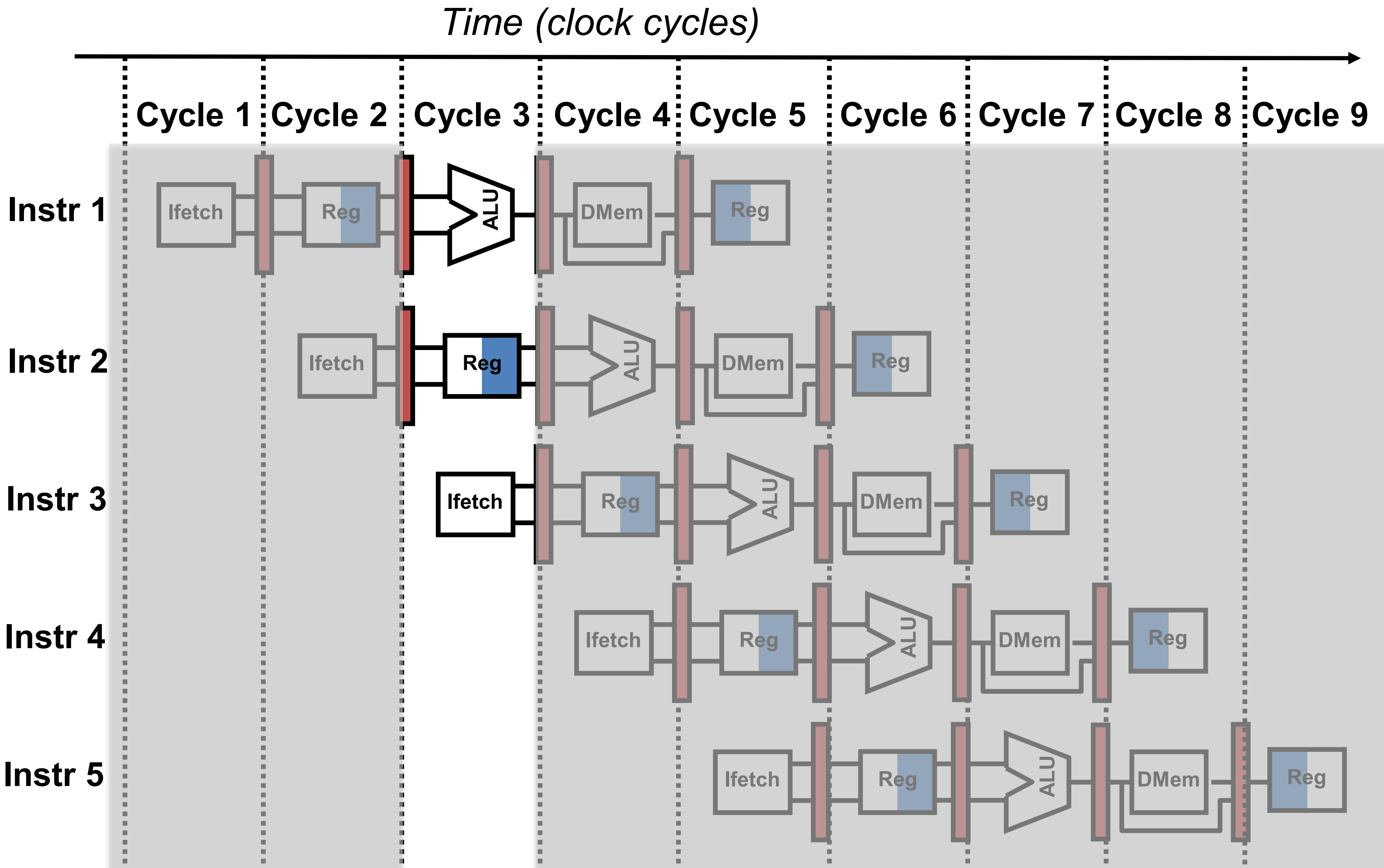


- At each cycle we fetch a new instruction
- And pass the preceding instruction to the next stage of the pipeline



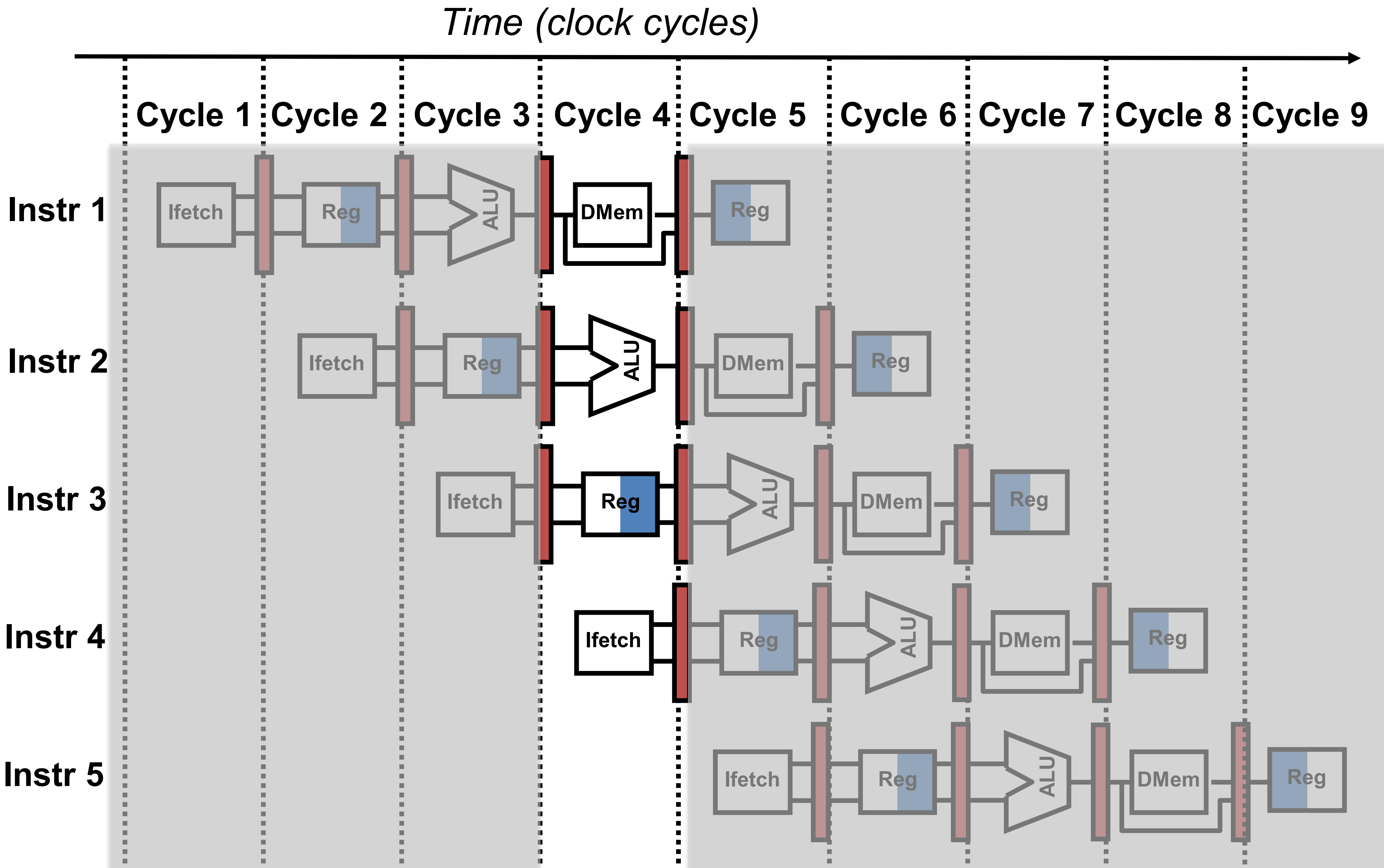
- At each cycle we fetch a new instruction
- And pass the preceding instruction to the next stage of the pipeline

Figure 3.3, Page 133 , CA:AQA 2e



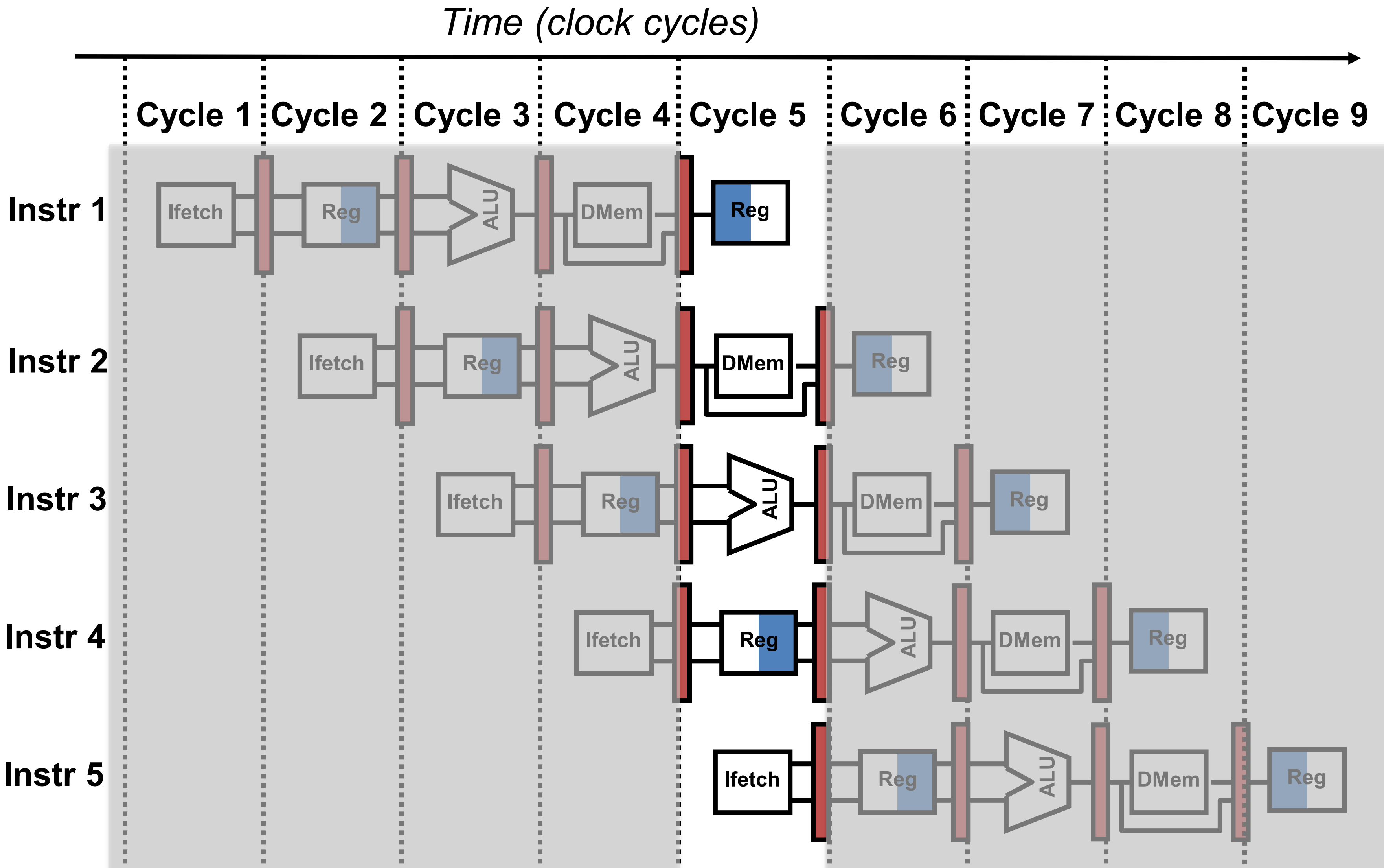
- **At each cycle we fetch a new instruction**
- **And pass the preceding instruction to the next stage of the pipeline**

Figure 3.3, Page 133 , CA:AQA 2e



- At each cycle we fetch a new instruction
- And pass the preceding instruction to the next stage of the pipeline

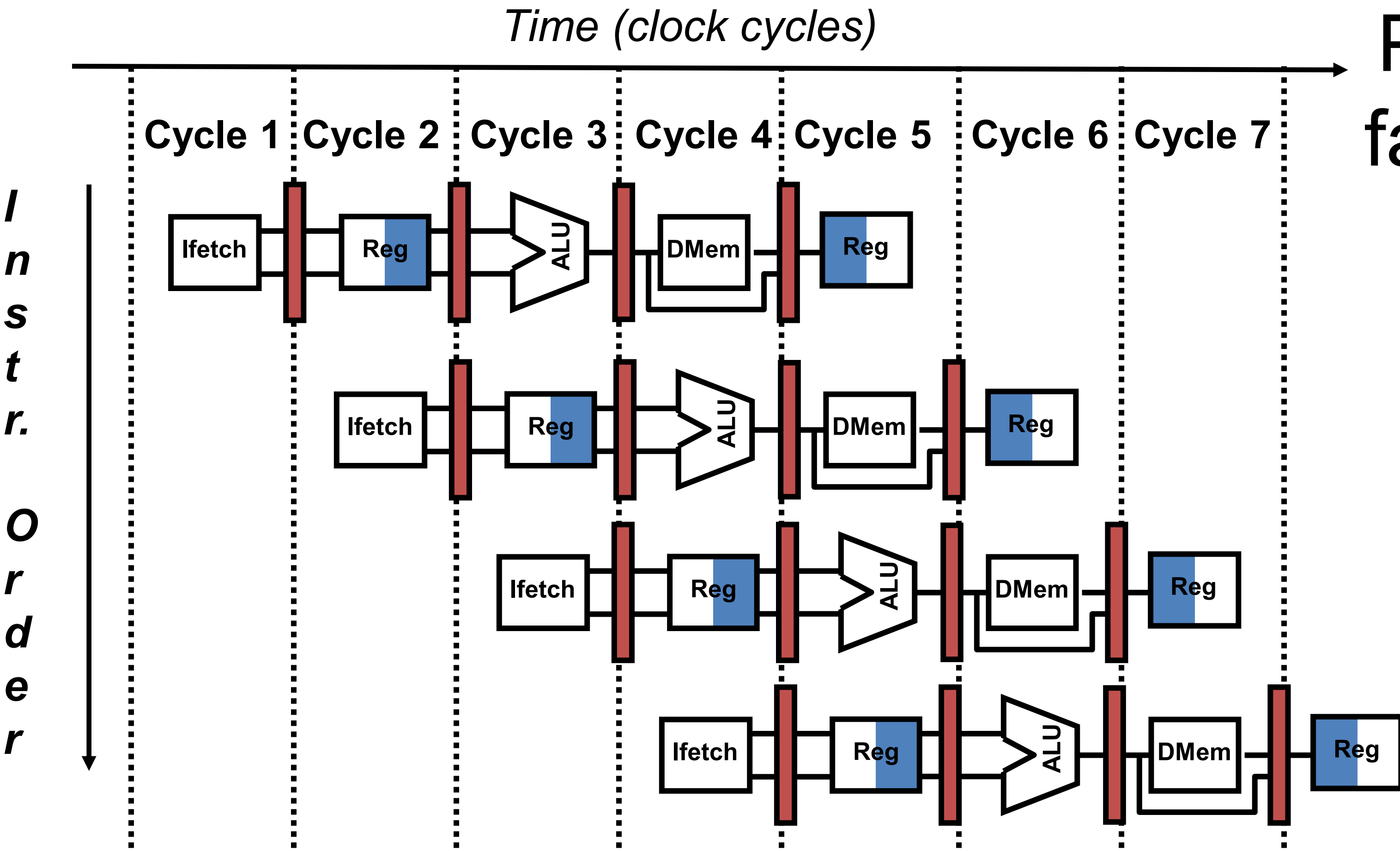
Figure 3.3, Page 133 , CA:AQA 2e



- **At cycle 5 the pipeline is fully-occupied – all stages are busy**
  - **IF is fetching Instr 5**
  - **ID is decoding Instr 4**
  - **EX is executing Instr 3**
  - **MEM is handling Instr 2 if it is a load or store**
  - **WB is writing the register results from Instr 1 back**

Figure 3.3, Page 133 , CA:AQA 2e

Pipelining: facts of life

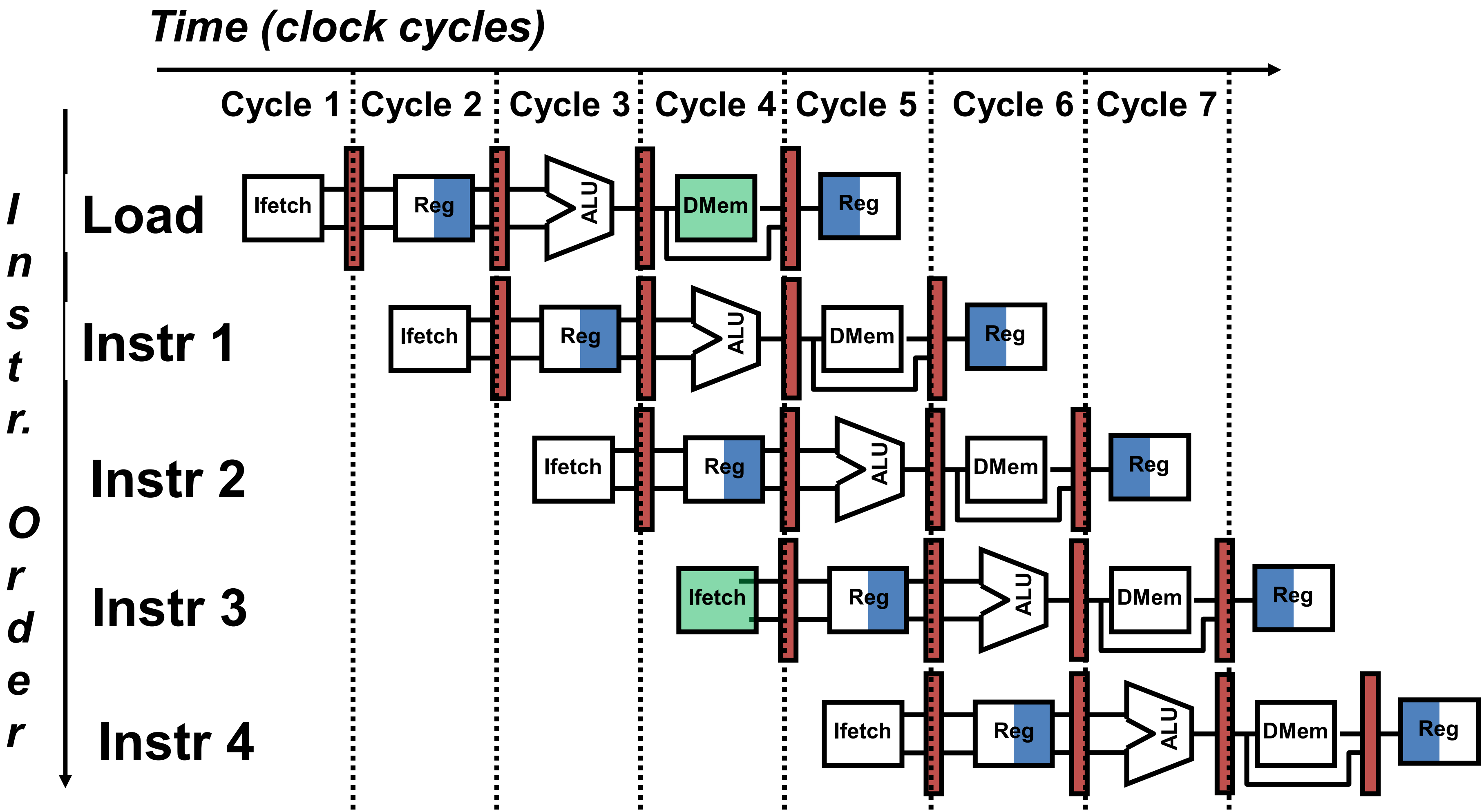


- Pipelining doesn't help latency of single instruction
  - it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup
- Speedup comes from parallelism - for free – no new hardware
- Many pipelines are more complex - Pentium 4 "Netburst" has 31 stages.

It's Not That Easy

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: the hardware cannot support this combination of instructions
  - Data hazards: Instruction depends on result of a prior instruction still in the pipeline
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

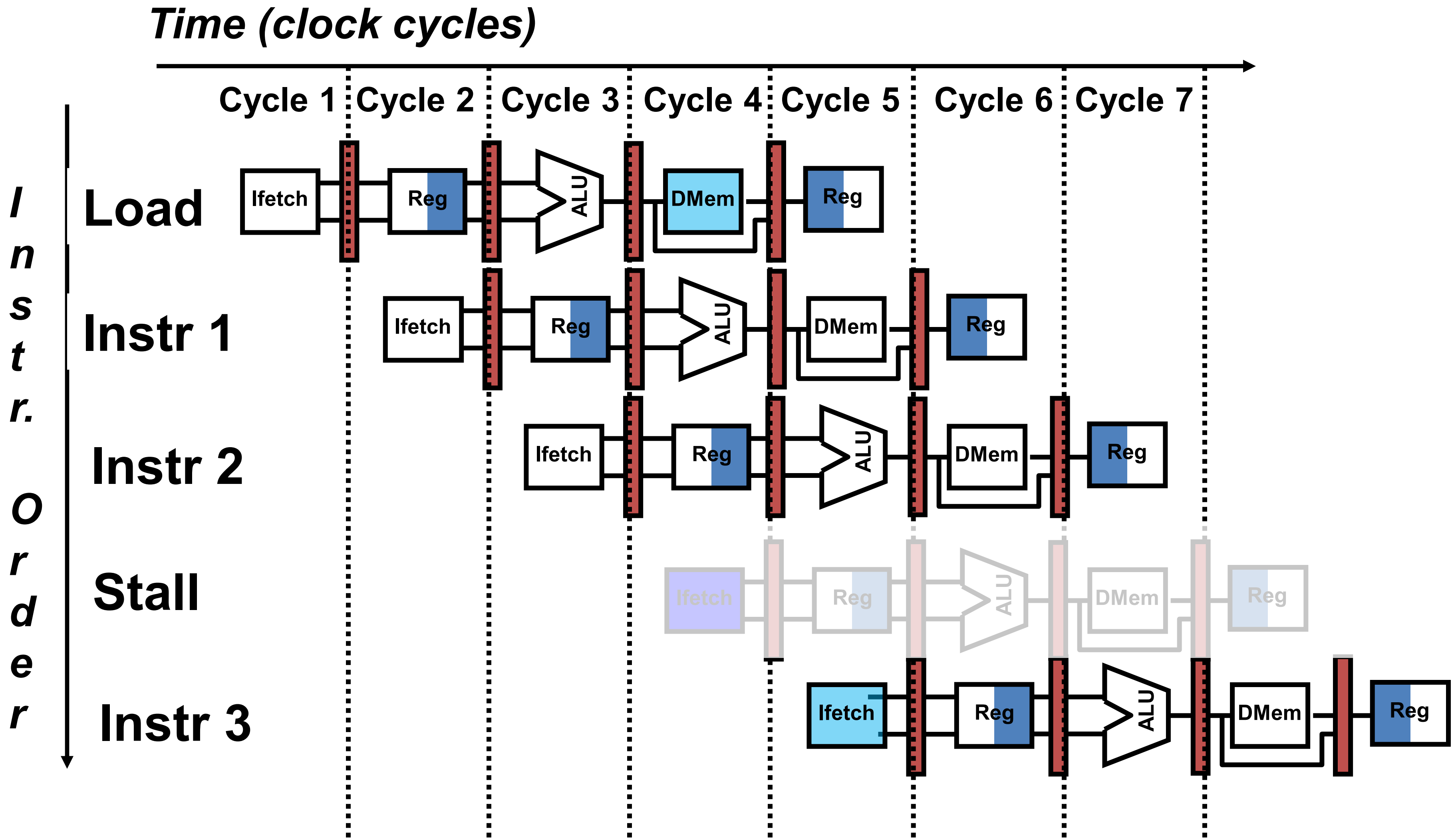
Structural Hazard: example – one RAM port



- Eg if there is only one memory for both instructions and data
- Two different stages may need access at same time
- Example: IBM/Sony/Toshiba Cell processor's "SPE" cores
  - The microarchitecture of the synergistic processor for a cell processor (IEEE J. Solid-State Circuits (V41(1) , Jan 2006)

Figure 3.6, Page 142 , CA:AQA 2e

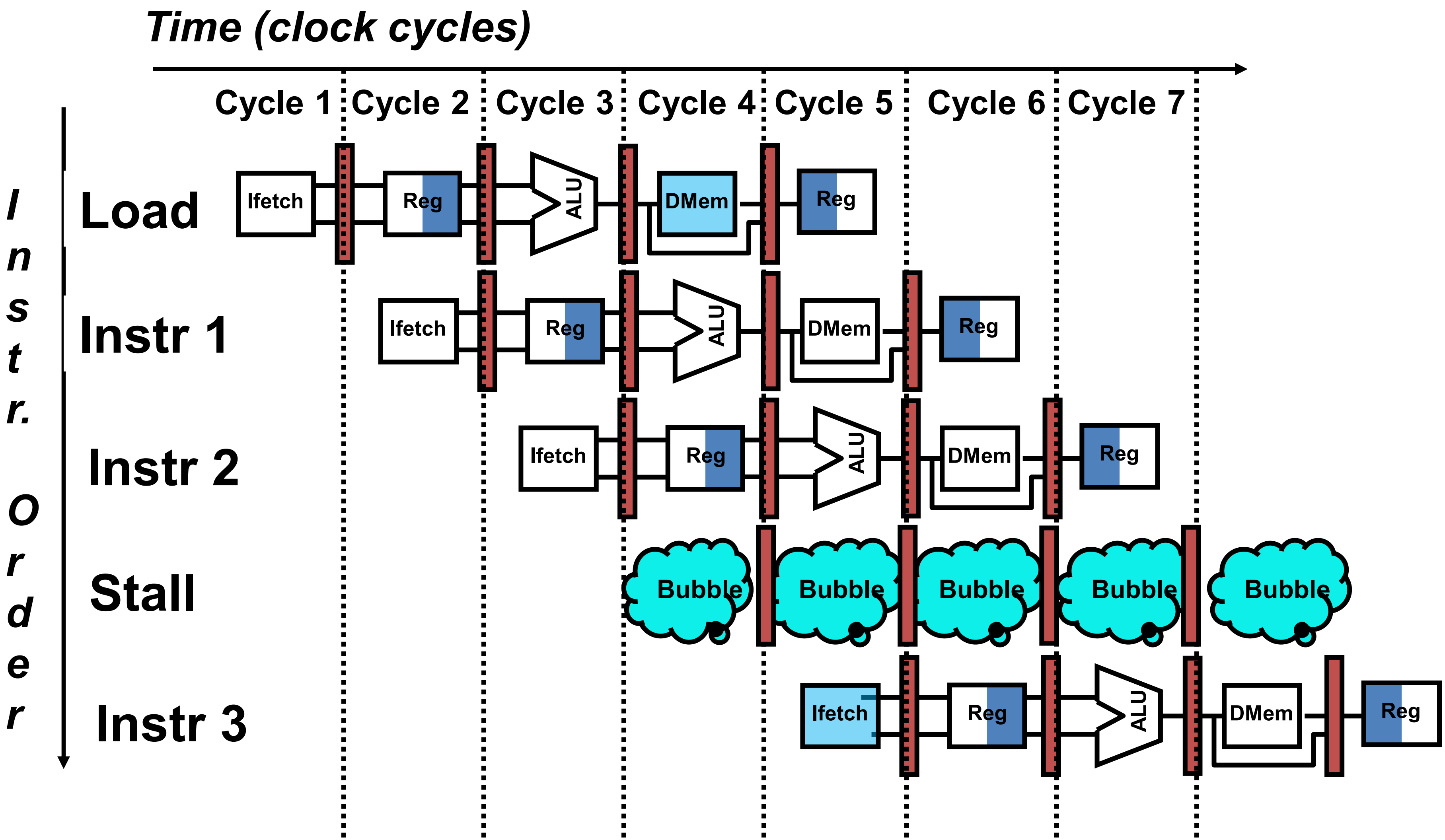
Structural Hazard: example – one RAM port



- Instr 3 cannot be loaded in cycle 4
- ID stage has nothing to do in cycle 5
- EX stage has nothing to do in cycle 6, etc. “Bubble” propagates

Figure 3.7, Page 143 , CA:AQA 2e

Structural Hazard: example – one RAM port



- Instr 3 cannot be loaded in cycle 4
- ID stage has nothing to do in cycle 5
- EX stage has nothing to do in cycle 6, etc. "Bubble" propagates

Figure 3.7, Page 143 , CA:AQA 2e

Data Hazard on R1

Time (clock cycles)

Instr. Order

add r1,r2,r3

sub r4,r1,r3

and r6,r1,r7

or r8,r1,r9

xor r10,r1,r11

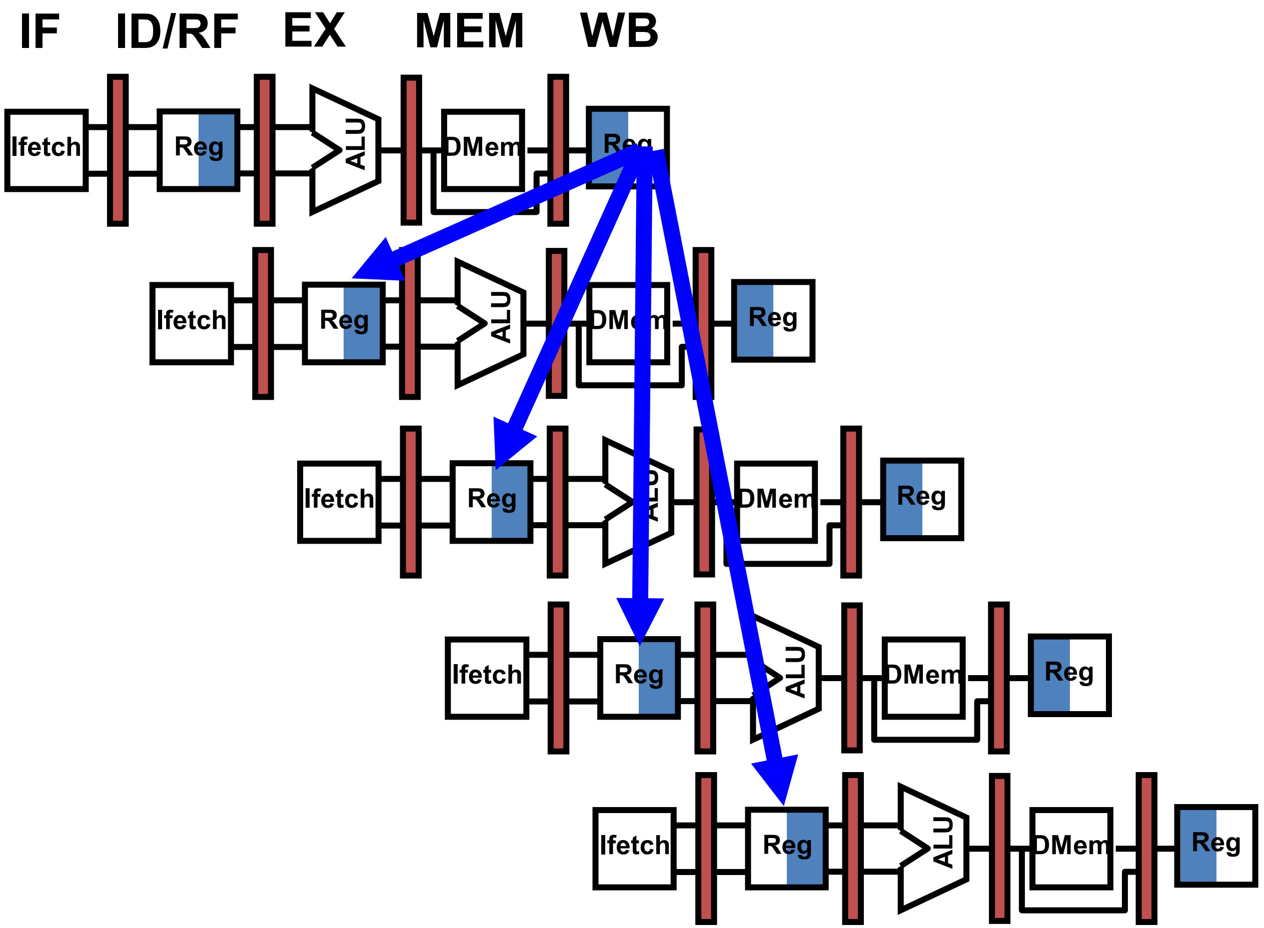
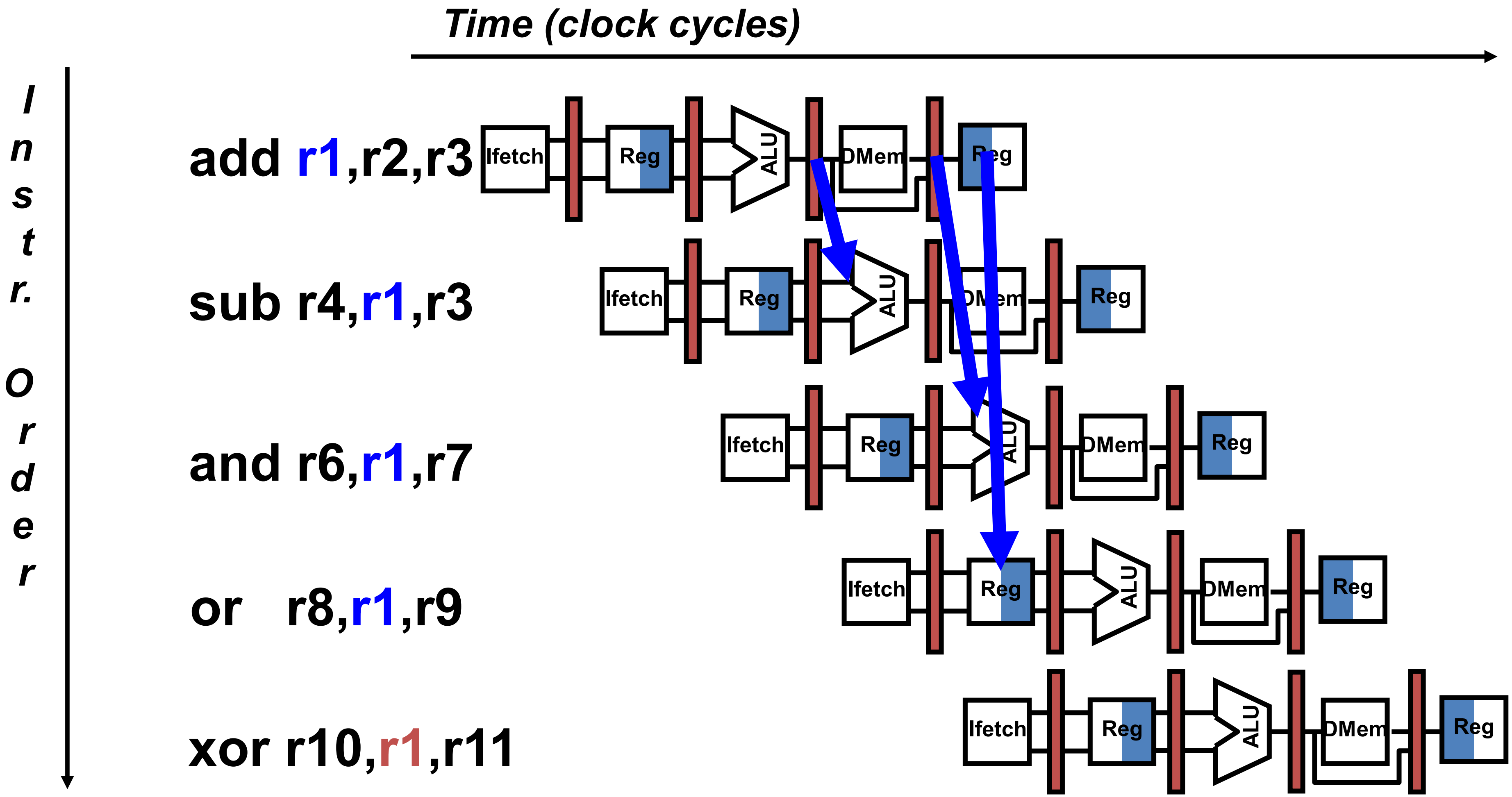


Figure 3.9, page 147 , CA:AQA 2e

Forwarding to Avoid Data Hazard

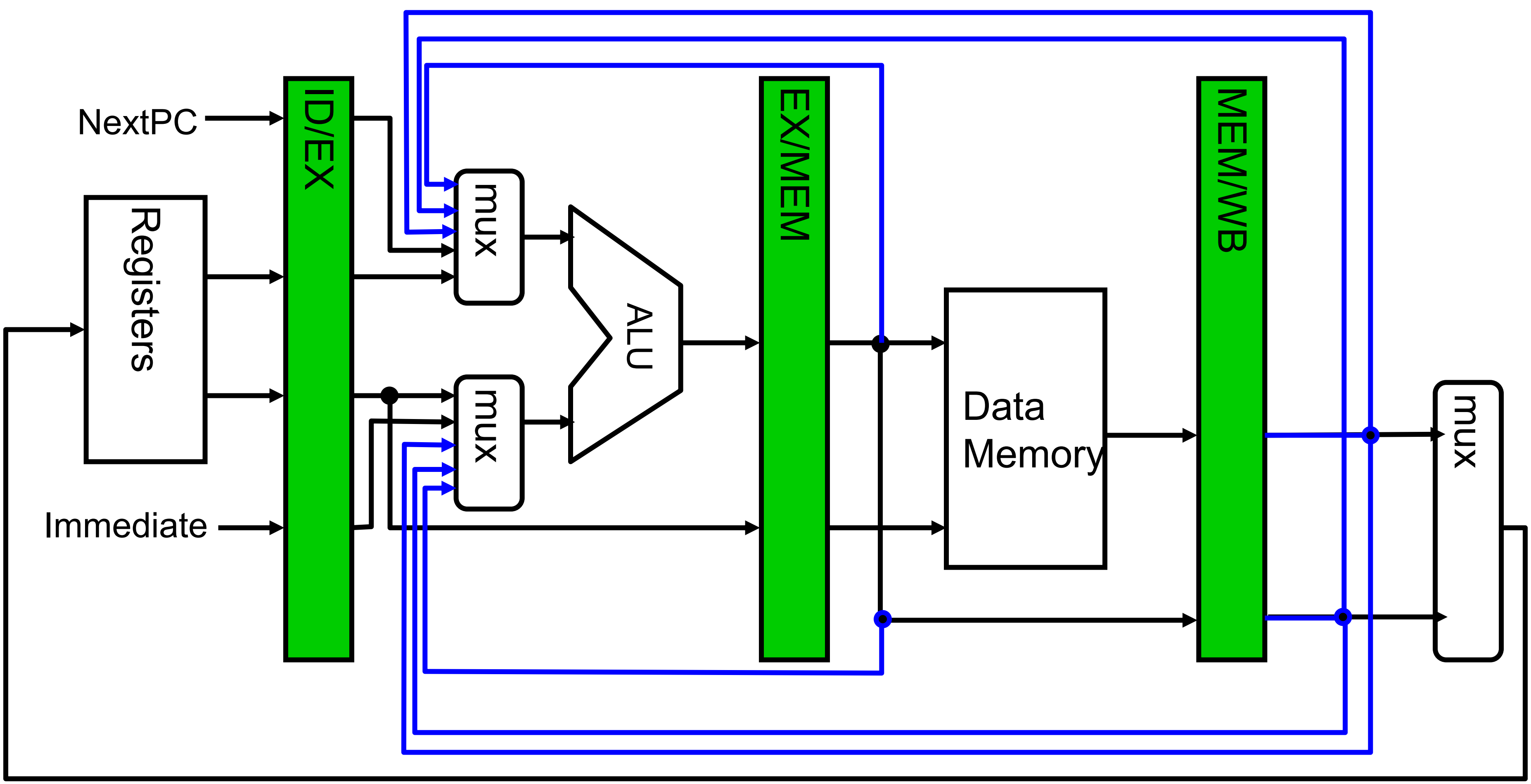
Figure 3.10, Page 149 , CA:AQA 2e



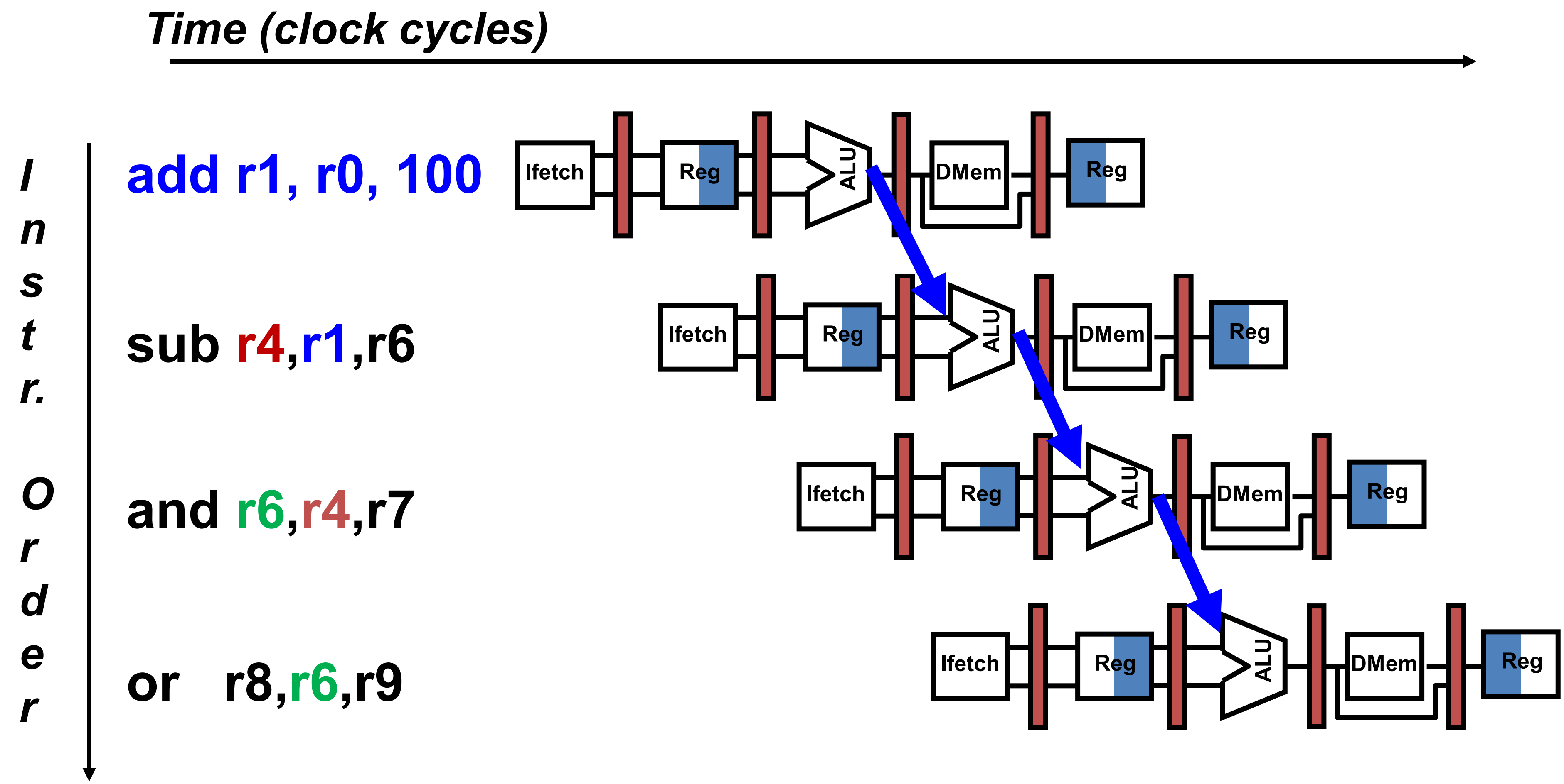
HW Change for Forwarding

Figure 3.20, Page 161, CA:AQA 2e

- Add forwarding ("bypass") paths
- Add multiplexors to select where ALU operand should come from
- Determine mux control in ID stage
- If source register is the target of an instrn that will not WB in time



Forwarding builds the data flow graph



- Values are passed directly from the output of the ALU to the input
- Via forwarding wires
- That are dynamically configured by the instruction decoder/control
- (This gets much more exciting when we have multiple ALUs and multiple-issue)

Imagine a machine with more ALUs

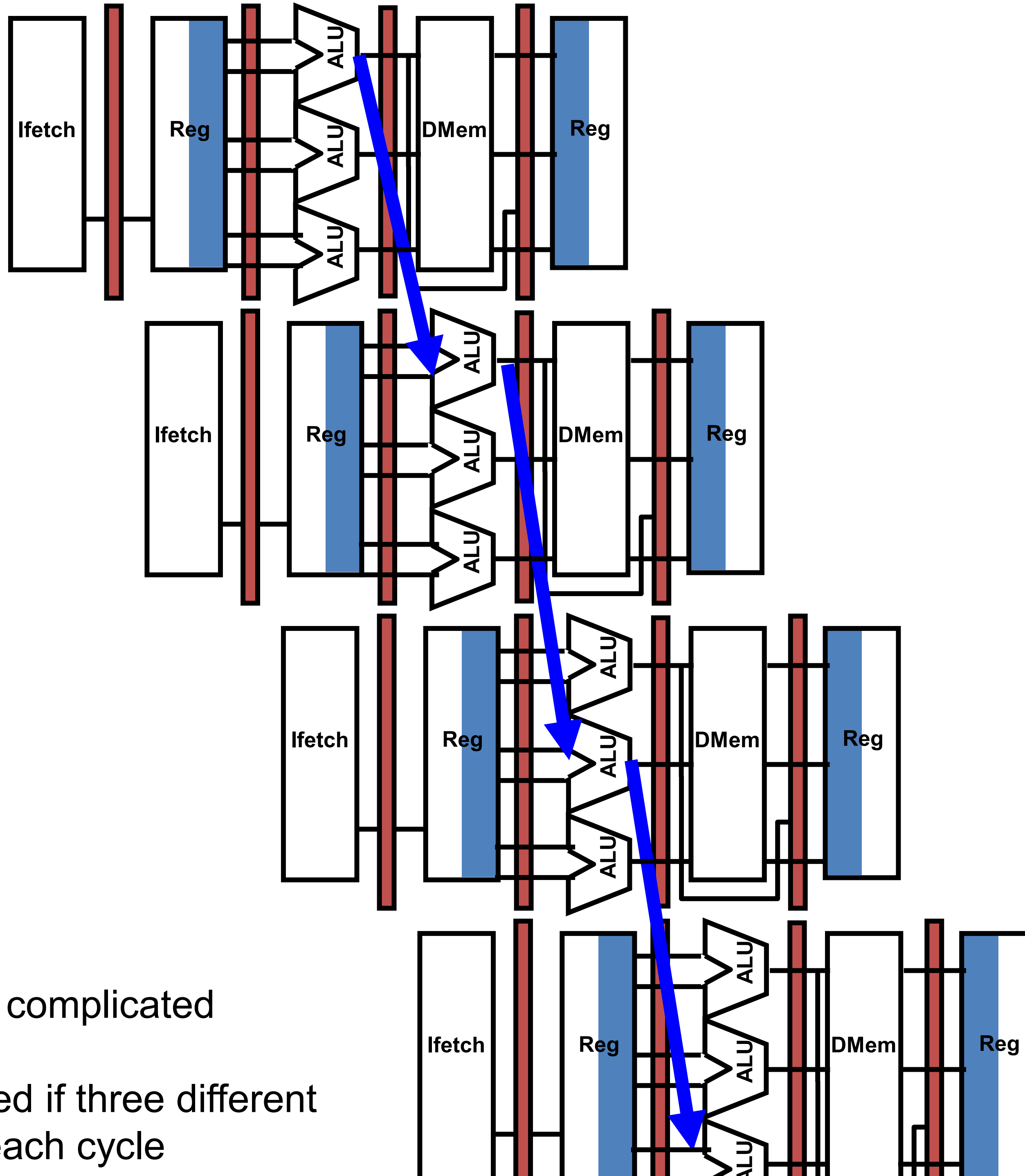
Instr. Order

add r1, r0, 100

sub r4,r1,r6

and r6,r4,r7

or   r8,r6,r9



- We would need a rather complicated forwarding network
- It's a bit more complicated if three different instructions are issued each cycle

Data Hazard Even with Forwarding

Figure 3.12, Page 153 , CA:AQA 2e

Time (clock cycles)

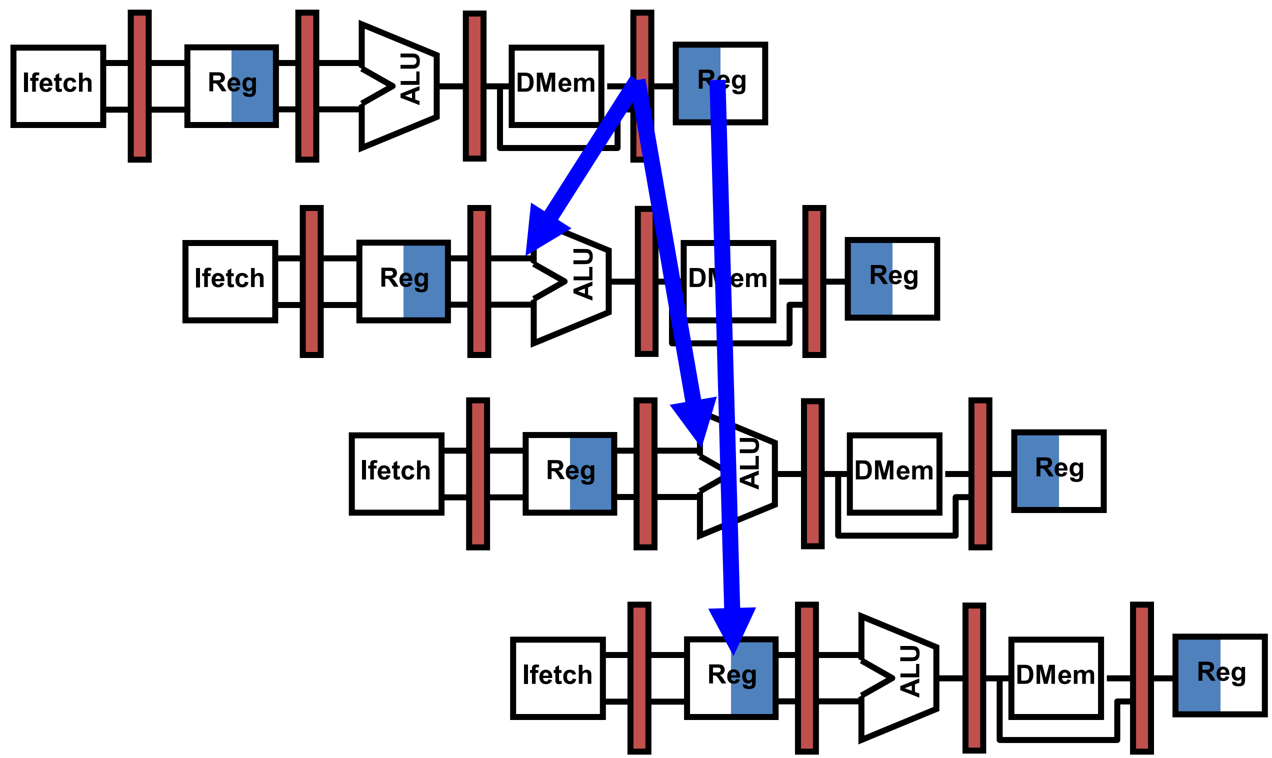
Instr. Order

lw r1, 0(r2)

sub r4,r1,r6

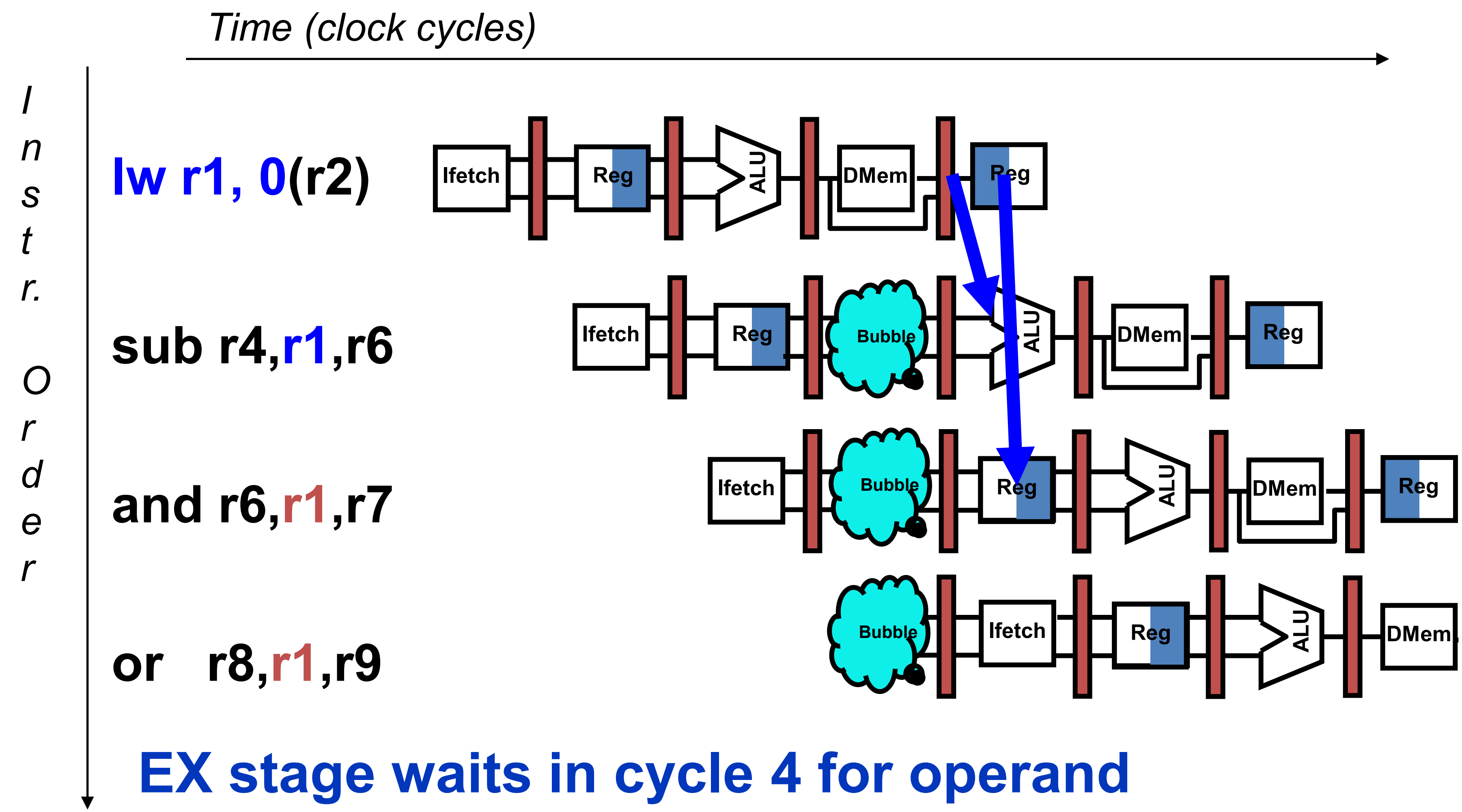
and r6,r1,r7

or r8,r1,r9



Data Hazard Even with Forwarding

Figure 3.12, Page 153 , CA:AQA 2e



EX stage waits in cycle 4 for operand
Following instruction ("and") waits in ID stage
Missed instruction issue opportunity…

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:		Fast code:	
LW	Rb,b	LW	Rb,b
LW	Rc,c	LW	Rc,c
STALL		LW	Re,e
ADD	Ra,Rb,Rc	ADD	Ra,Rb,Rc
SW	a,Ra		
LW	Re,e		
LW	Rf,f	LW	Rf,f
STALL		SW	a,Ra
SUB	Rd,Re,Rf	SUB	Rd,Re,Rf
SW	d,Rd	SW	d,Rd
10 cycles (2 stalls)		8 cycles (0 stalls)	

Show the stalls explicitly

Control Hazard on Branches

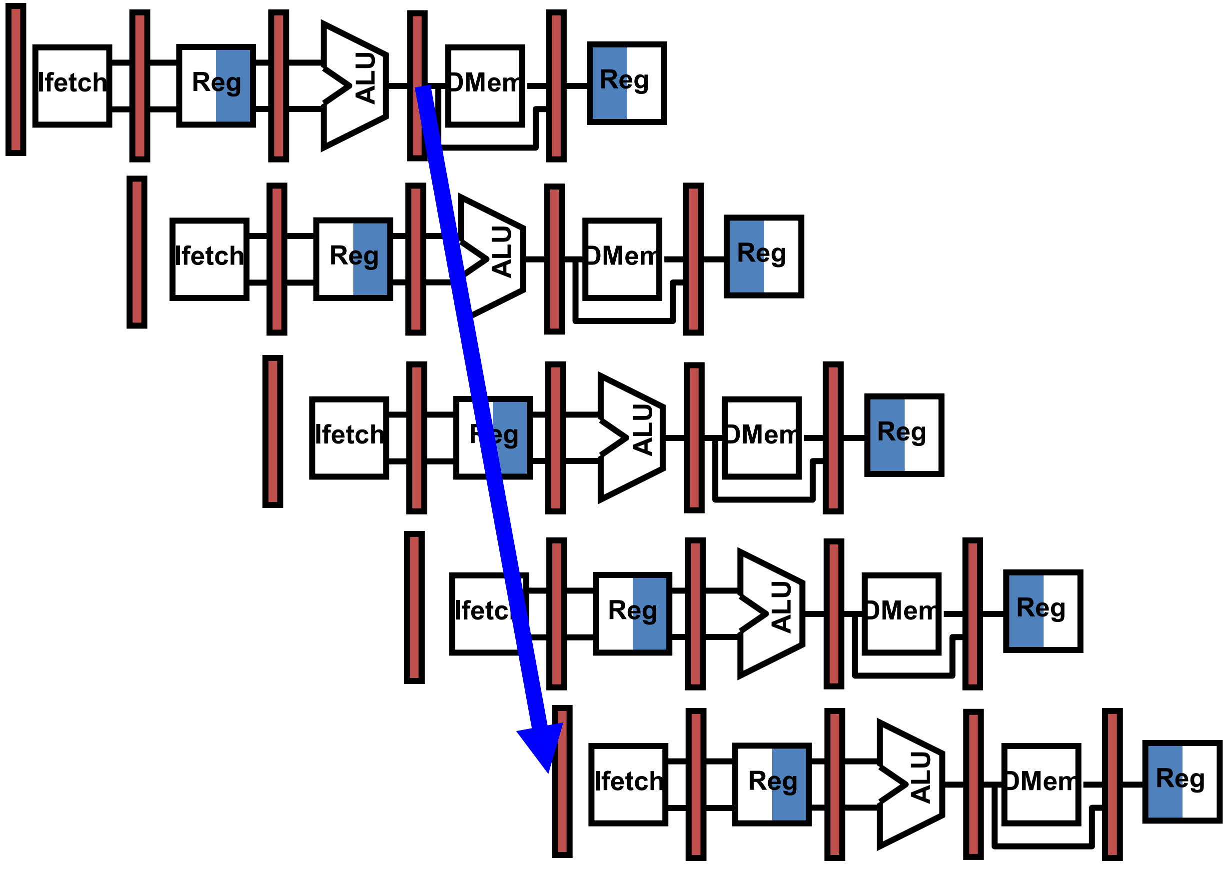
10: beq r1,r3,36

14: and r2,r3,r5

18: or r6,r1,r7

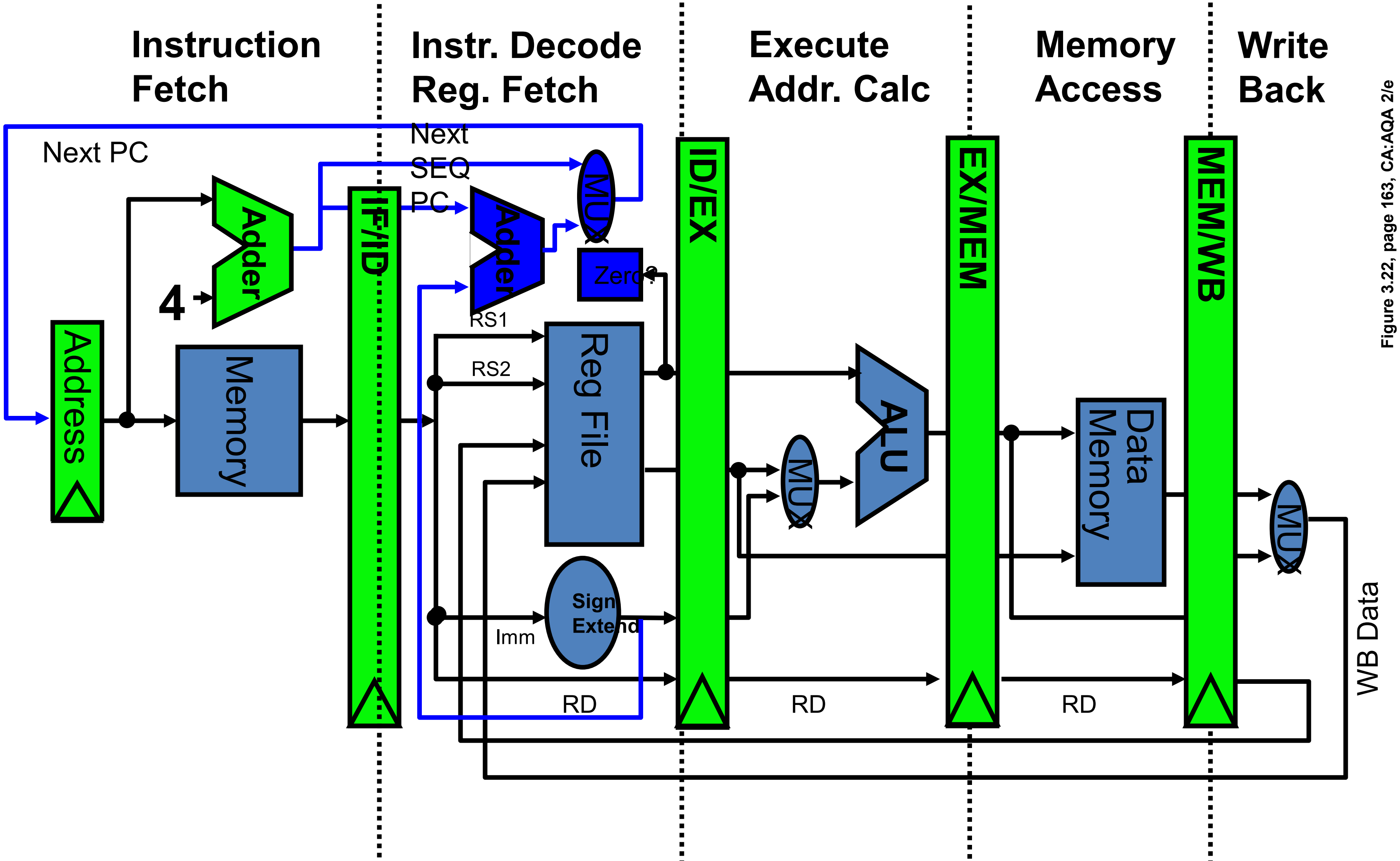
22: add r8,r1,r9

36: xor r10,r1,r11



If we're not smart we risk a three-cycle stall

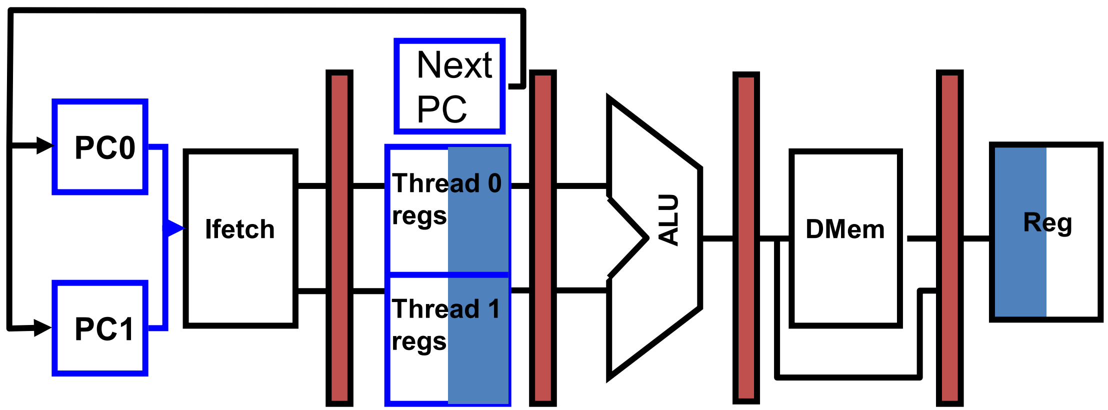
Pipelined MIPS Datapath with early branch determination



- Add extra hardware to the decode stage, to determine branch direction and target earlier
- We still have a one-cycle delay – we just have to fetch and start executing the next instruction
- If the branch is actually taken, block the MEM and WB stages and fetch the right instruction

Eliminating hazards with multi-threading

- If we had no stalls we could finish one instruction every cycle
- If we had no hazards we could do without forwarding – and decode/control would be simpler too



Example: PowerPC processing element (PPE) in the Cell Broadband Engine (Sony PlayStation 3)

- **IF maintains two Program Counters**
- **Even cycle – fetch from PC0**
- **Odd cycle – fetch from PC1**
- **Thread 0 reads and writes thread-0 registers**
- **No register-to-register hazards between adjacent pipeline stages**

(cf "C-Slowing"......)

So – how fast can this design go?

- A simple 5-stage pipeline can run at 5-9GHz
- Limited by critical path through slowest pipeline stage logic
- Tradeoff: do more per cycle? Or increase clock rate?
  - Or do more per cycle, in parallel…
- At 3GHz, clock period is 330 picoseconds.
  - The time light takes to go about four inches
  - About 10 gate delays
    - for example, the Cell BE is designed for 11 FO4 (“fan-out=4”) gates per cycle: www.fe.infn.it/~belletti/articles/ISSCC2005-cell.pdf
    - Pipeline latches etc account for 3-5 FO4 delays leaving only 5-8 for actual **work**

- **How can we build a RAM that can implement our MEM stage in 5-8 FO4 delays?**

Summary

- **This course is about fetch-execute machines!**
- **The fetch-decode-execute sequence is naturally pipelinable**
- **Pipelining would be wonderful… but:**
  - **Control hazards**
  - **Data hazards**
  - **Structural hazards**

  We will see how all of these can be tackled with dynamically-scheduled “out-of-order” microarchitectures
- **Hazards can sometimes be handled by *forwarding***
- **Hazards sometimes cause *stalls***
- **Control hazards are just trouble! But there are things we can do!**
- **Pipeline design affects the maximum clock rate**

Next: tutorial exercise 1 on the connection between the instruction set and the pipeline architecture

And then we'll look at caches and the memory system

For next week

Watch Ch01-part3 on caches

- Make sure you understand it -
- come with questions

Have a think about the Turing Tax discussion exercise – watch the video!

- Come prepared to talk about it!

Watch Ch02-part1 on dynamic instruction scheduling

- Come with questions!

Feeding curiosity

- **Do you really need pipeline latches?**
  - **Perhaps we could compute with just the wavefront of the signal as it propagates through the combinational logic?**
  - **But what if the wires are not precisely matched in length?**
  - **See** *Wave-Pipelining: A Tutorial and Research Survey*, **Burleson et al 1998 https://ieeexplore.ieee.org/abstract/document/711317**
- **Do we really need a global clock?**
  - **Look up asynchronous circuit design**
- **What's the optimal number of pipeline stages?**
  - **Eg see** *Optimizing Pipelines for Power and Performance*, **Srinivasan et al MICRO 2002 https://vlsiarch.eecs.harvard.edu/sites/hwpi.harvard.edu/files/vlsiarch/files/micro2002-optpipeline.pdf?m=1651843040**