Chapter 06: Instruction Pipelining and Parallel Processing

Lesson 03:

Instruction Hazards and Data hazards

Objective

- Understand instruction-pipeline hazards—data hazard (dependencies)

Instruction throughput idle case

Instruction throughput

- Pipelining increases processor performance by increasing the instruction throughput
- Several instructions overlapped in the pipeline, cycle time can be reduced, increasing the rate at which instructions execute

Ideal case throughput

- In the ideal case, the throughput of a pipeline is simply 1/cycle time
- Assume— 5-stage pipeline with a 6-ns cycle time and an unpipelined cycle time of 25 ns
- Ideal throughput of = 1/6ns = 1.67×10^8 instructions/s (= 167 MIPS)
- MIPS stands for million instructions per second

Improvement over the unpipelined processor

- A more than 4 × improvement over the unpipelined processor's throughput of 4×10^7 = 40 MIPS instructions/s

Limiting factor for Instruction throughput with respect to idle case

Limiting Factor

- Number of factors limit a pipeline's ability to execute instructions at its peak rate

1. Dependencies between instructions operands (called data dependency hazards)
2. Branches (called program flow control hazards), and the time required to access memory

Instruction Pipeline Data Dependency Hazards

Instruction Pipeline Hazards

- RAR
- RAW
- WAR and WAW

Hazard

- A risk in which pipeline operations *stall* (stop) for one or more clock cycles

Stall (Bubble)

- The instruction at the later stage waits for the front one to complete and thus clock cycle(s) is not utilized during that period
- Stall─ a bubble in the pipeline

Data Dependency Hazard

- Can occur among the operands in the instruction at the pipeline stages
- Instruction hazards (data dependencies) occur when instructions read or write registers that are used by other instructions

RAR

Read-after-read (RAR) hazards

- Occurs when two instructions both read from the same register
- Don't cause a problem for the processor because reading a register doesn't change the register's value

Detection condition for RAR hazards

- I_n — an *n*-th instruction at an advanced stage of processing
- I_{n+1} — an *n+1* th instruction at previous stage of processing
- I_n and I_{n+1} contain at least one common operand
- Input operands be same in I_{n+1} as in I_n

RAR Example

Instructions

ADD r1, r2, r3

SUB r4, r5, r3

Both instructions read r3, creating a RAR hazard

RAW

Read-after-write (RAW)

- The detection condition of RAW hazards is that O_n and I_{n+1} contain at least one common operand
- Output operand(s)— O_n of I_n
- Input operands— I_{n+1} of $n+1$ instruction at the back

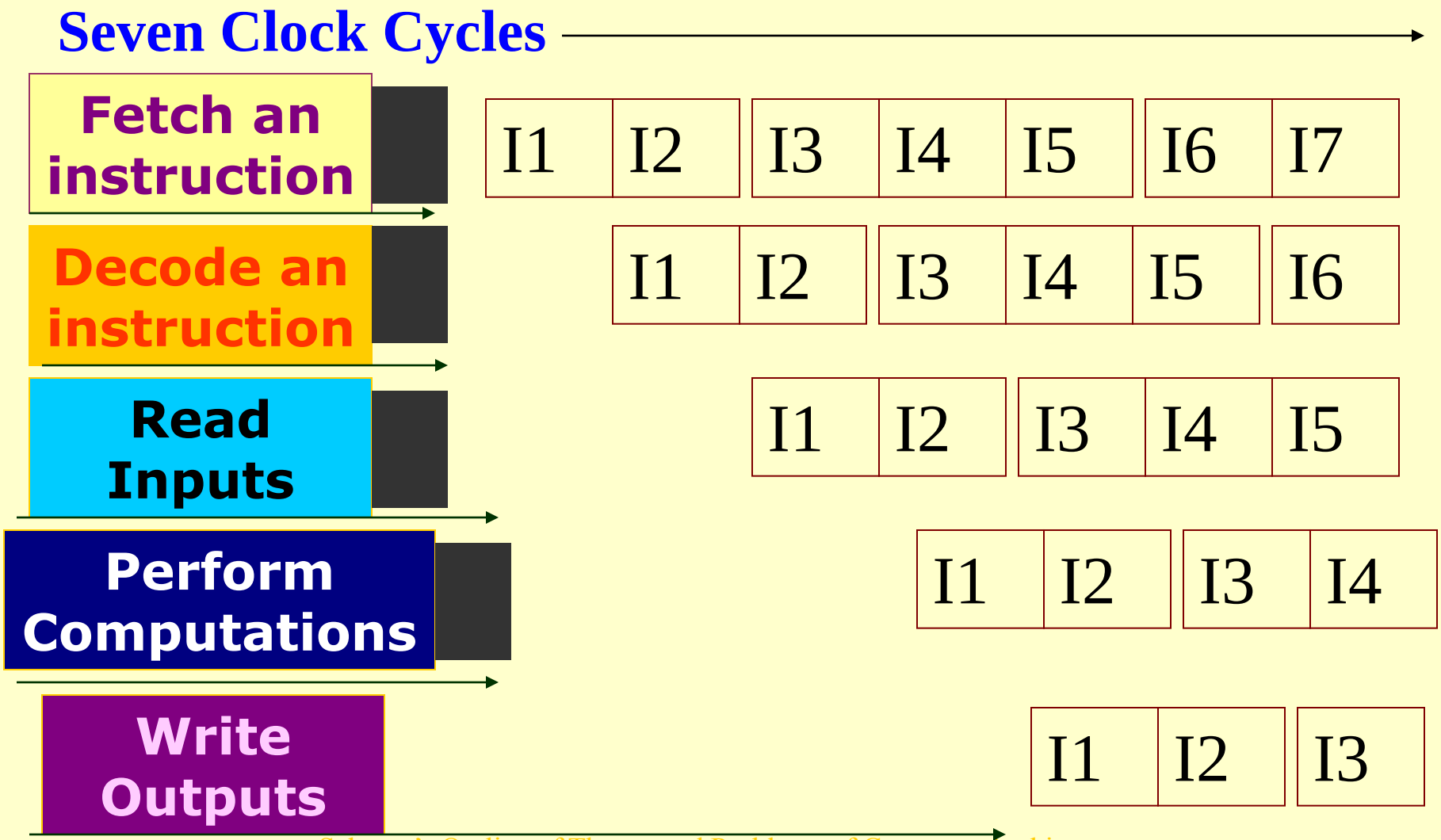
RAW Example

Instructions
ADD r1, r2, r3

SUB r4, r5, r1

Subtract reads output of the addition, creating a RAW hazard

Seven Clock Cycles



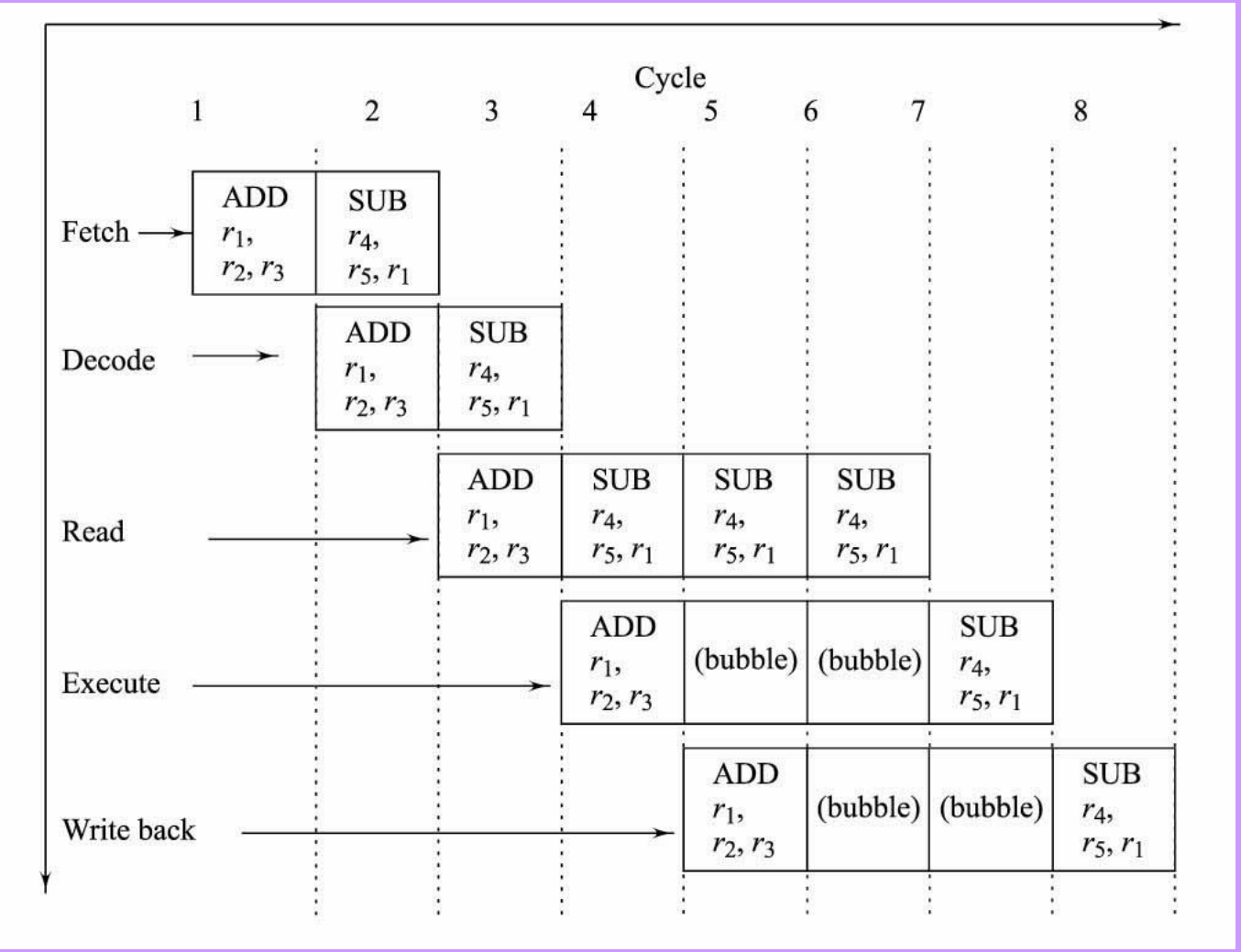
RAW and pipeline stall or bubble

- Reading instruction can proceed through the instruction fetch and instruction decode stages of the pipeline before the writing instruction completes, because the reading instruction does not need the value produced by the writing instruction until it reaches the register read stage

RAW and pipeline stall or bubble

- Reading instruction cannot proceed past the register read stage of the pipeline until the writing instruction has passed through the write-back stage
- The data that the reading instruction needs is not available until then
- Wait of a cycle pipeline stall or bubble

Pipeline stall



Inserting NOPs

- In cycle 5, the subtract would normally enter the execute stage, but it is prevented from doing so because it was not able to read r1 on cycle 4
- Instead, the hardware inserts a special no-operation (NOP) instruction, known as a bubble, into the execute stage of the pipeline, and the subtract tries to read its input registers again on cycle 5

Effect of pipeline stall or bubble

- The result of the ADD still unavailable on cycle 5 because the ADD has not completed the write back stage yet,
- so the subtract cannot enter the execute stage on cycle 6
- On cycle 6, the SUB is able to read r1, and it proceeds into the execute stage on cycle 7

WAR and WAW

Write-After-Read (WAR)

- Hazards occur when the output register of an instruction is used for write after read by a previous instruction

Write-After-Write (WAW)

- Hazard occur when the output register of an instruction is used for write after written by a previous instruction

Effects of WAR and WAW

- These hazards are sometimes called *name dependencies*, as they occur because the processor has a finite number of registers
- If the processor had an infinite number of registers, it could use a different register for the output of each instruction, and WAW and WAR hazards would never occur

WAR Detection

- The detection condition of WAR hazards is that I_n and O_{n+1} contain at least one common operand

WAR Example

ADD r1, r2, r3

SUB r2, r5, r6

Subtract writes r2, which is read by the addition, creating a WAR hazard

WAW Detection

- The detection condition for WAW hazards is that O_n and O_{n+1} contain at least one common operand

WAW Example

Instructions

ADD r1, r2, r3

SUB r1, r5, r6

Subtract writes the same register as the addition, creating a WAW hazard

In Order Execution

- If a processor executes instructions in the order that they appear in the program and uses the same pipeline for all instructions, WAR and WAW hazards do not cause the delays because of the way instructions flow through the pipeline

WAW

- The output register of an instruction written in the write back stage of the pipeline
- Instructions with WAW hazards will enter the write back stage in the order in which they appear in the program
- Write their results into the register in the right order

WAR

- Even less of a problem, because the register read stage of the pipeline occurs before the write back stage

WAR

- By the time an instruction enters the write back stage of the pipeline, all previous instructions in the program have already passed through the register read stage and read their input values
- Writing instruction can go ahead and write its destination register without causing any problems

Different Latencies Effects

- WAW and WAR hazards can cause problems, because it is possible for a low-latency instruction to complete before a longer-latency instruction that appeared earlier in the program

Different Latencies Effects

- These processors must keep track of name dependencies between instructions and stall the pipeline as necessary to resolve these hazards

WAR and WAW hazards

- An issue in out-of-order processor
- Processor allows instructions to execute in different orders than the original program to improve performance

Summary

We learnt

- Data Dependent Hazards
- RAR
- RAW causes bubble
- WAR and WAW results in name dependencies
- WAR and WAW create the issue in out-of-order executing processor pipeline

End of Lesson 3 on

Instruction Hazards and Data hazards