



# Computer Hardware Engineering (IS1200)

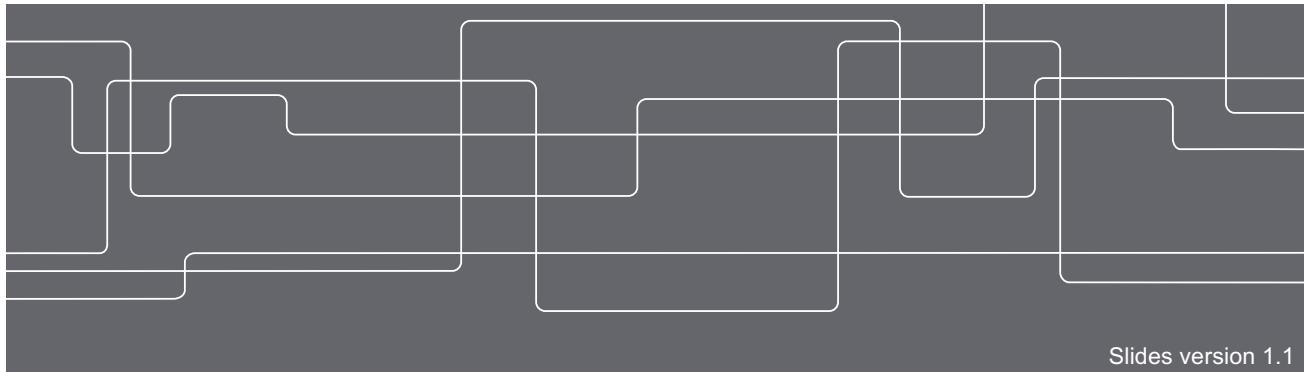
## Computer Organization and Components (IS1500)

Spring 2018

### Lecture 10: Pipelined Processors

David Broman

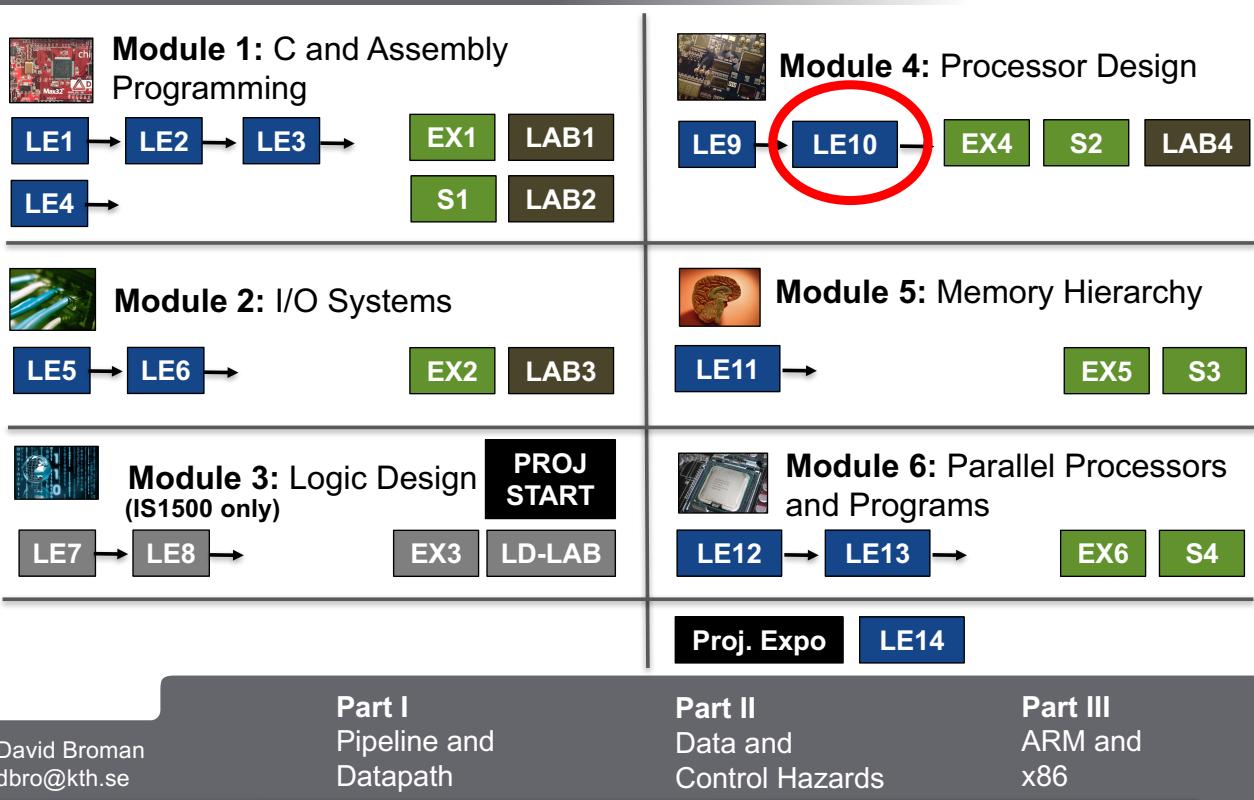
Associate Professor, KTH Royal Institute of Technology



2

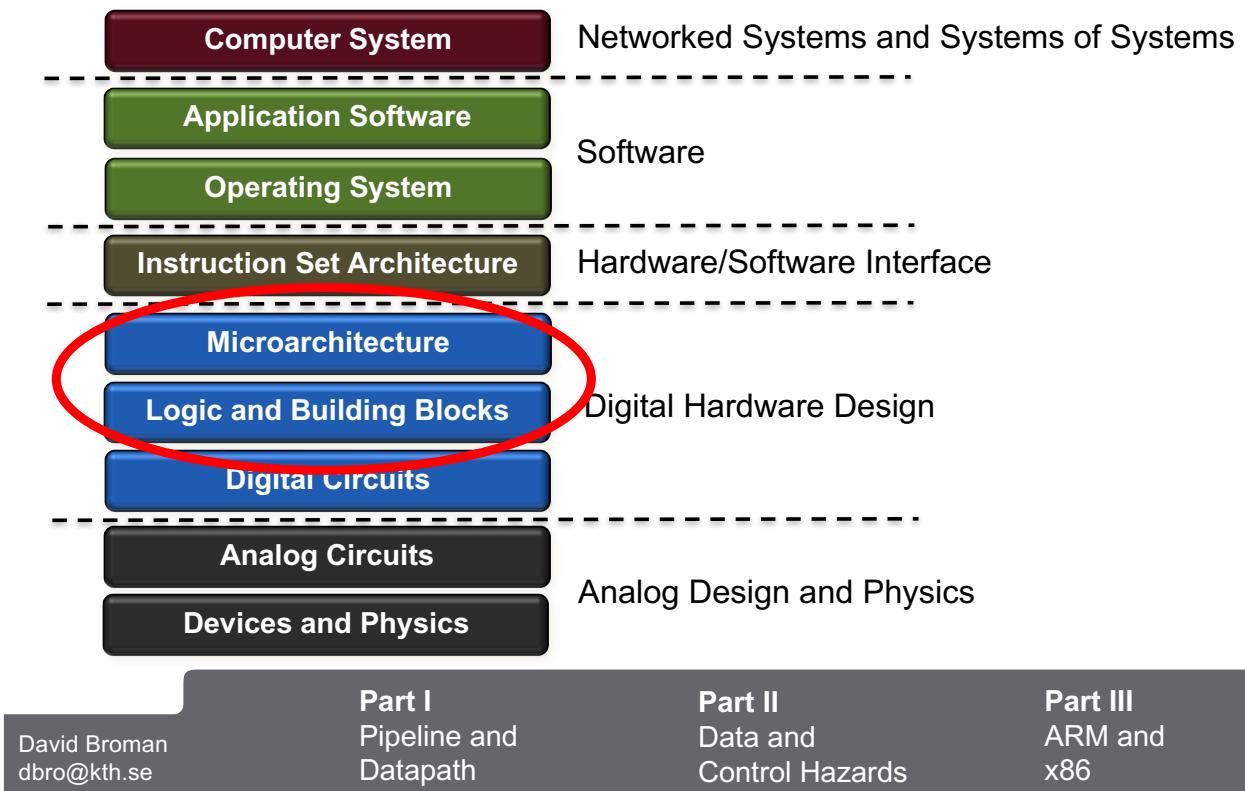


## Course Structure

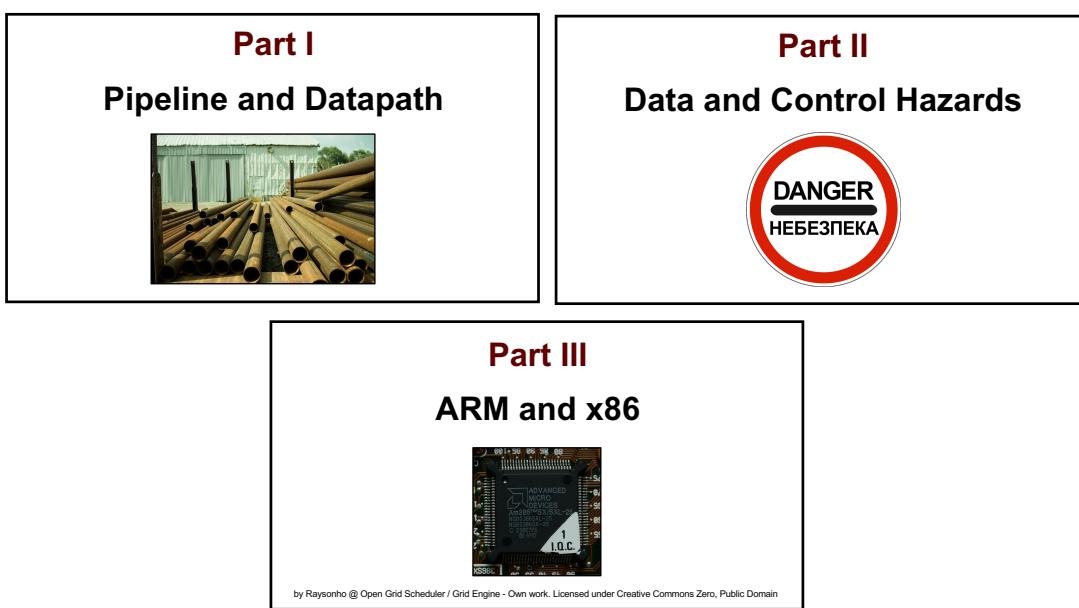




# Abstractions in Computer Systems



## Agenda



David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86



## Part I

# Pipeline and Datapath



Acknowledgement: The structure and several of the good examples are derived from the book "Digital Design and Computer Architecture" (2013) by D. M. Harris and S. L. Harris.

David Broman  
dbro@kth.se



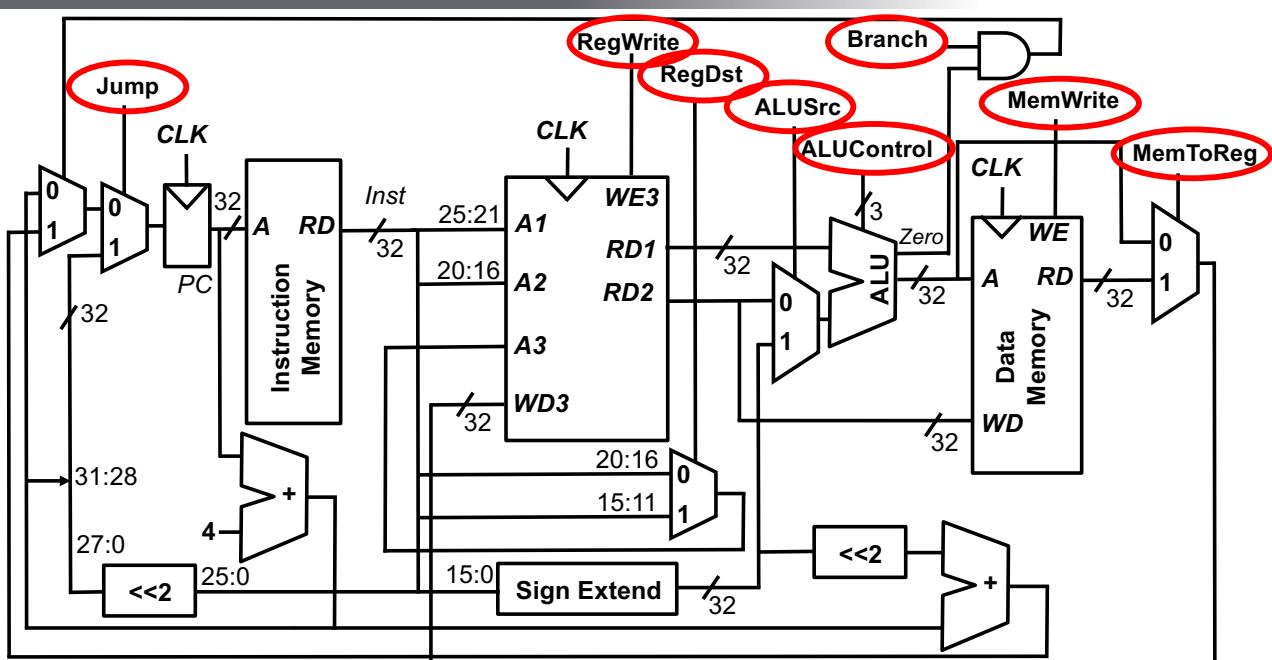
**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86



## Data Path (Revisited)



David Broman  
dbro@kth.se

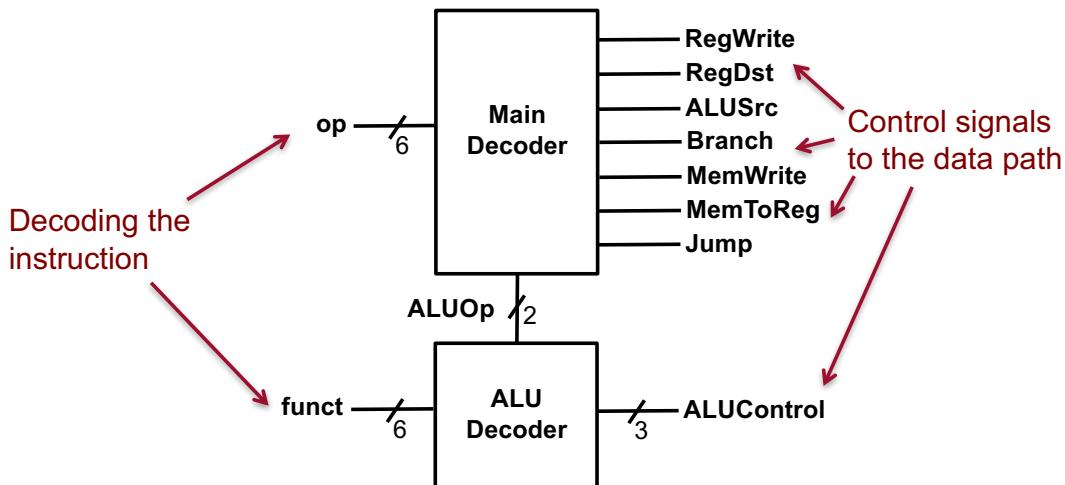


**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

## Control Unit (Revisited)



David Broman  
dbro@kth.se



**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

## Performance Analysis (Revisited)

$$\text{Execution time (in seconds)} = \# \text{ instructions} \times \frac{\text{clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

Number of instructions in a program (# = number of)

Determined by programmer or the compiler or both.

Average cycles per instruction (CPI)

Determined by the micro-architecture implementation.

Seconds per cycle = clock period  $T_c$

Determined by the critical path in the logic.

For the single-cycle processor, each instruction takes one clock cycle. That is, CPI = 1.

The main problem with the single-cycle processor design (last lecture) is the long critical path.

**Solution: Pipelining**

David Broman  
dbro@kth.se



**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

**Processing System:** A system that takes input and produces outputs.



**Token:** An input that is processed by the processing system and results in an output.

**Latency:** The time it takes for the system to process one token.

**Throughput:** The number of tokens that can be processed per time unit.

David Broman  
dbro@kth.se



**Part I**  
Pipeline and  
Datapath

**Part II**

Data and  
Control Hazards

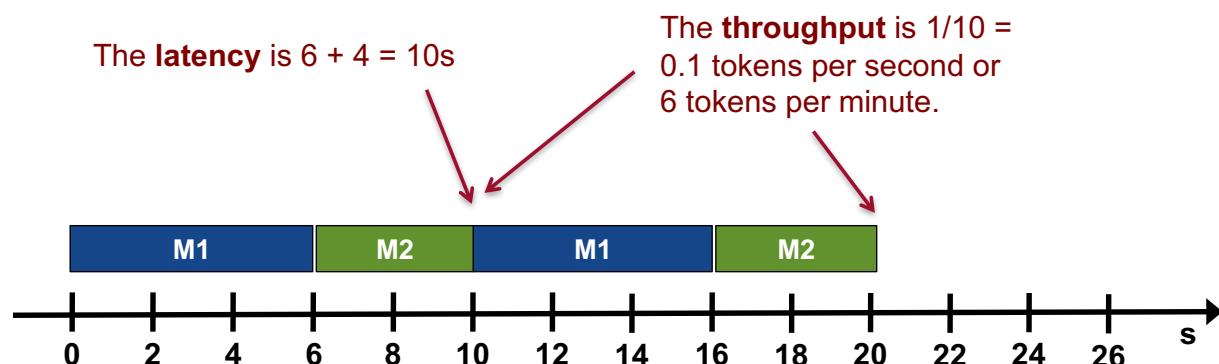
**Part III**

ARM and  
x86

**Example:** Assume we have a Christmas card factory with two machines (M1 and M2).

**M1:** Prints out the card (takes 6s)  
**M2:** Puts on a stamp (takes 4s)

**Approach 1.** Process tokens **sequentially**.  
In this case a token is a card.



David Broman  
dbro@kth.se



**Part I**  
Pipeline and  
Datapath

**Part II**

Data and  
Control Hazards

**Part III**

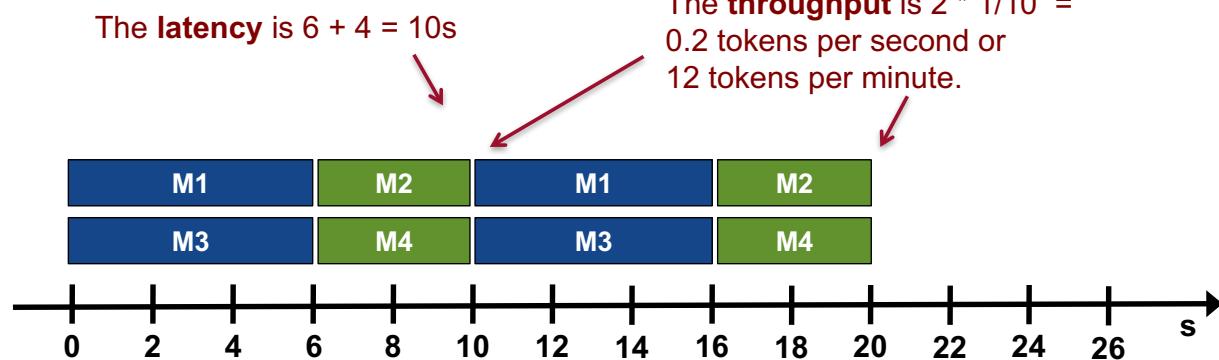
ARM and  
x86

## Parallelism and Pipelining (3/6)

### Parallel Processing (Spatial Parallelism)

**Example:** Assume we have a Christmas card factory with four machines.

**Approach 2.** Process tokens in parallel using more machines.



David Broman  
dbro@kth.se



Part I  
Pipeline and  
Datapath

Part II  
Data and  
Control Hazards

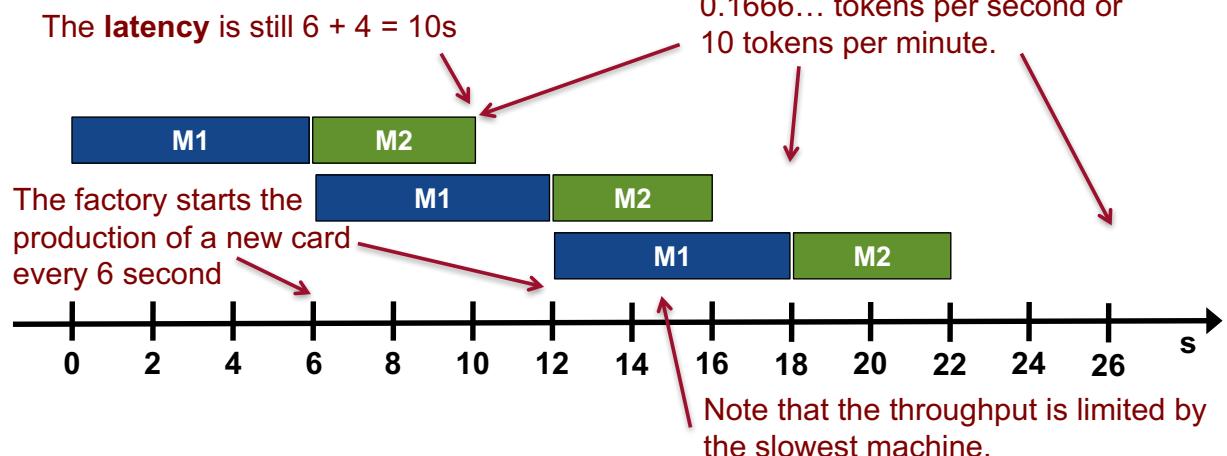
Part III  
ARM and  
x86

## Parallelism and Pipelining (4/6)

### Pipelining (Temporal Parallelism)

**Example:** Assume we have a Christmas card factory with two machines.

**Approach 3.** Process tokens by pipelining using only two machines.



David Broman  
dbro@kth.se



Part I  
Pipeline and  
Datapath

Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86

**Approach 1.** Process tokens sequentially using two machines

Latency: 10s  
Throughput: 6 tokens/min

**Approach 2.** Process tokens in parallel using four machines

Latency: 10s  
Throughput: 12 tokens/min

We improve throughput, but not latency

**Approach 3.** Process tokens by pipelining using only two machines.

Latency: 10s  
Throughput: 10 tokens/min

Parallelism in approach 2 adds extra machines, but pipelining in approach 3 does not require extra hardware.

Again: throughput improvements are limited by the slowest machine (in this case M1)



David Broman  
dbro@kth.se



**Part I**  
Pipeline and  
Datapath

**Part II**

Data and  
Control Hazards

**Part III**

ARM and  
x86

# Parallelism and Pipelining (6/6) Performance Analysis for Pipelining

**Idea:** We introduce a pipeline in the processor

How does this affect the execution time?

$$\text{Execution time (in seconds)} = \# \text{ instructions} \times \frac{\text{clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

Pipelining does not change the number of instructions

Pipelining will not improve the CPI (actually, make it slightly worse)

Pipelining will improve the cycle period (make the critical path shorter)

David Broman  
dbro@kth.se



**Part I**  
Pipeline and  
Datapath

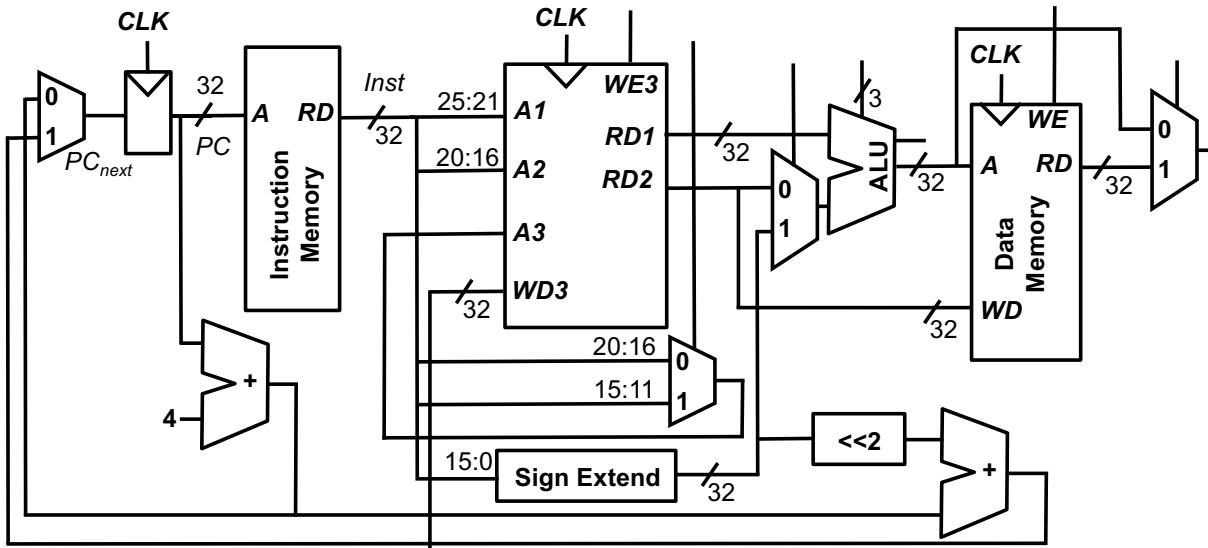
**Part II**

Data and  
Control Hazards

**Part III**

ARM and  
x86

Recall the single-cycle data path (the logic for the `j` and `beq` instructions is hidden)



David Broman  
dbro@kth.se



**Part I**  
Pipeline and  
Datapath

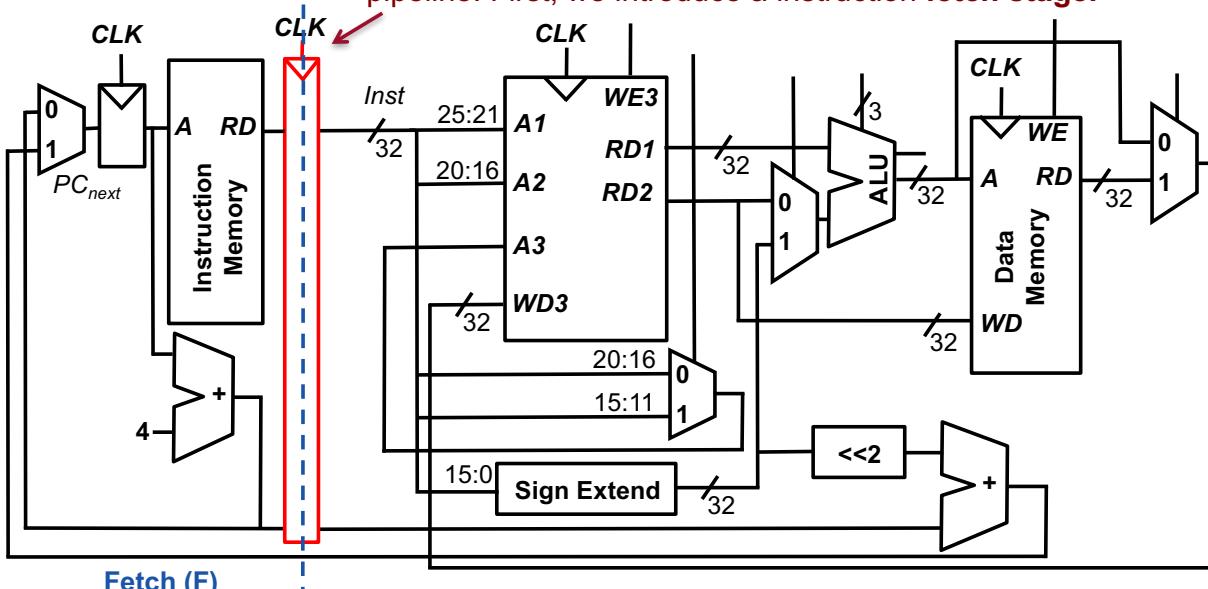
**Part II**

Data and  
Control Hazards

**Part III**  
ARM and  
x86

# Towards a Pipelined Datapath (2/8) Fetch Stage

A register splits the datapath into stages, forming a pipeline. First, we introduce a instruction **fetch stage**.



David Broman  
dbro@kth.se



**Part I**  
Pipeline and  
Datapath

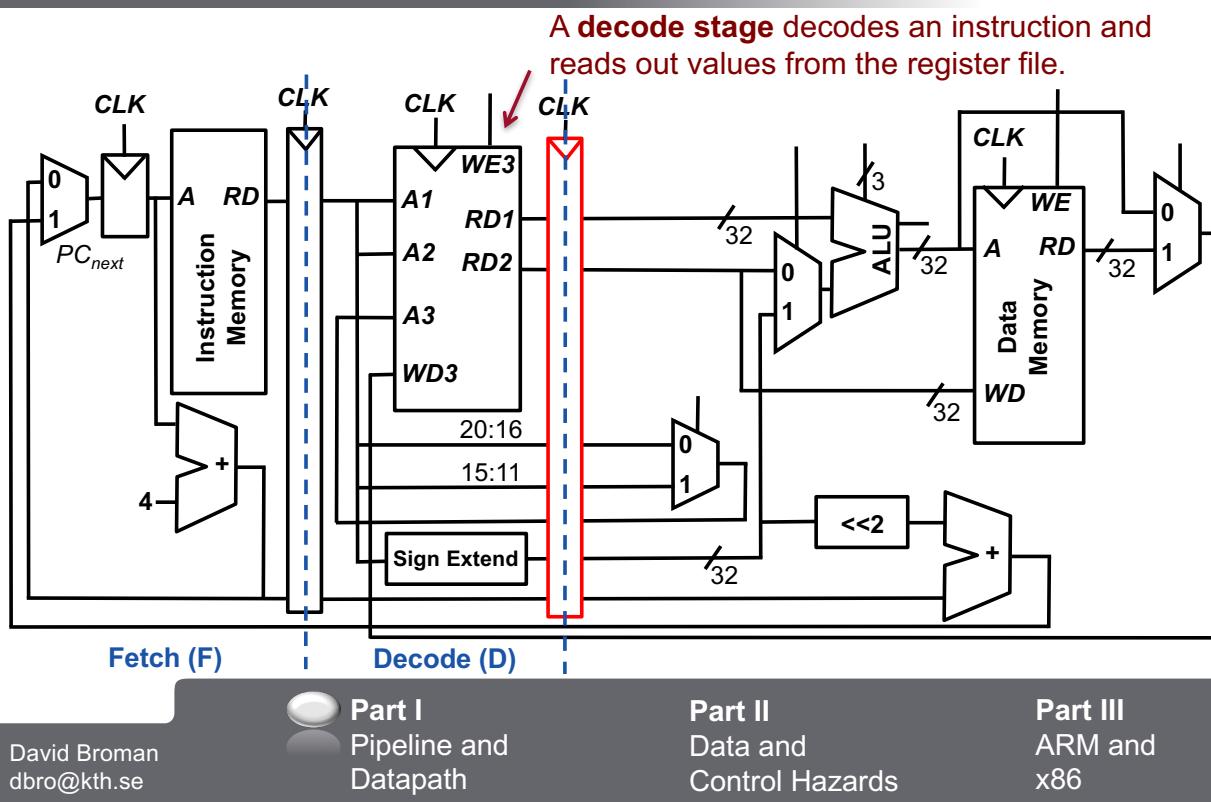
**Part II**

Data and  
Control Hazards

**Part III**  
ARM and  
x86

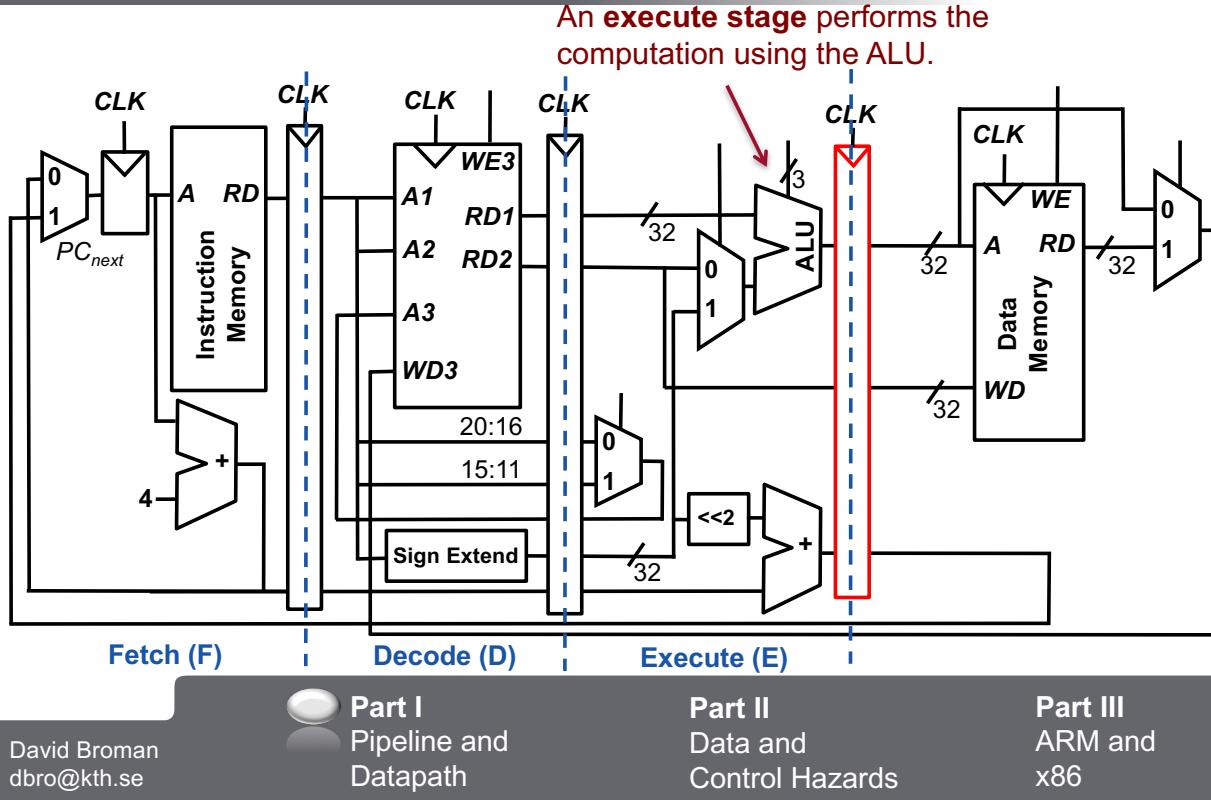
## Towards a Pipelined Datapath (3/8) Decode Stage

17



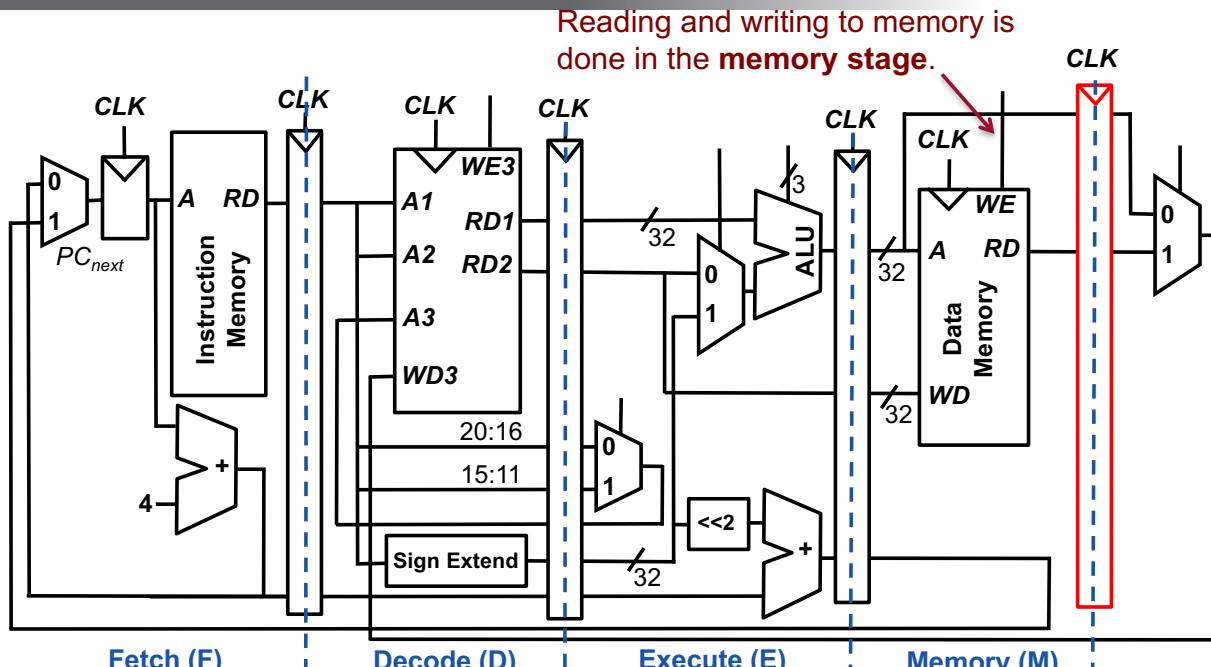
## Towards a Pipelined Datapath (4/8) Execute Stage

18



## Towards a Pipelined Datapath (5/8) Memory Stage

19



David Broman  
dbro@kth.se

Part I  
Pipeline and  
Datapath

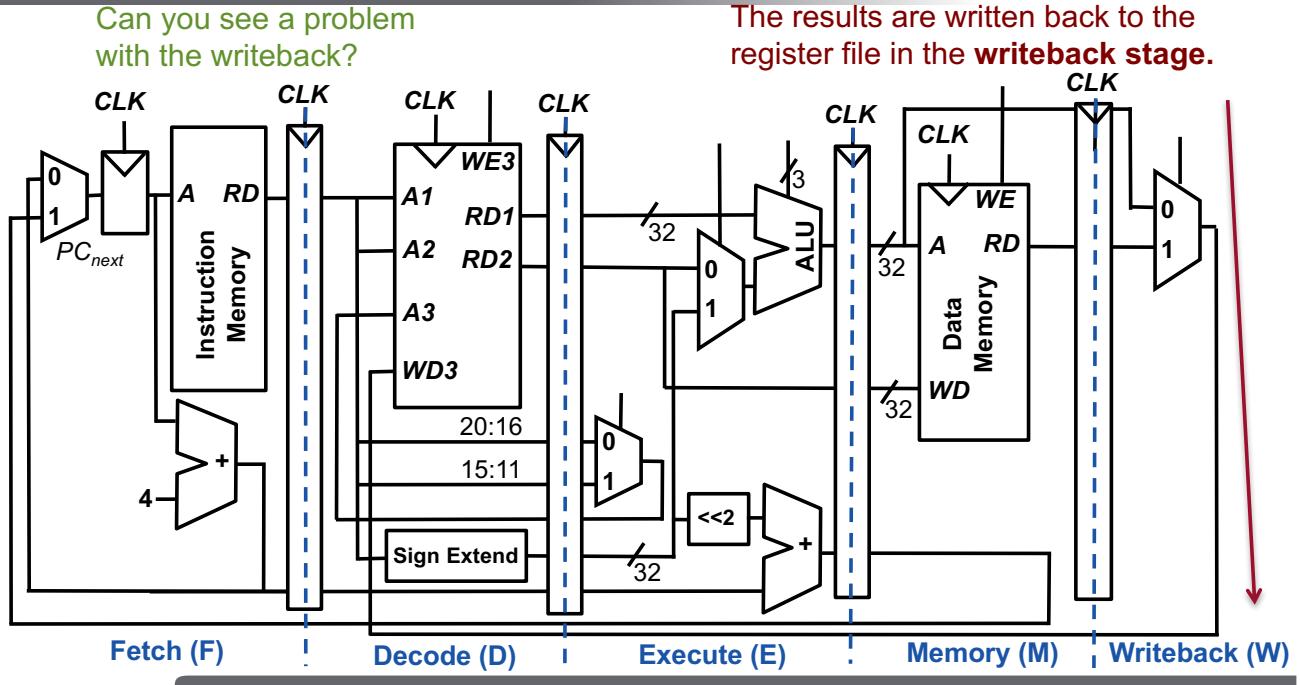
Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86



## Towards a Pipelined Datapath (6/8) Writeback Stage

20



David Broman  
dbro@kth.se

Part I  
Pipeline and  
Datapath

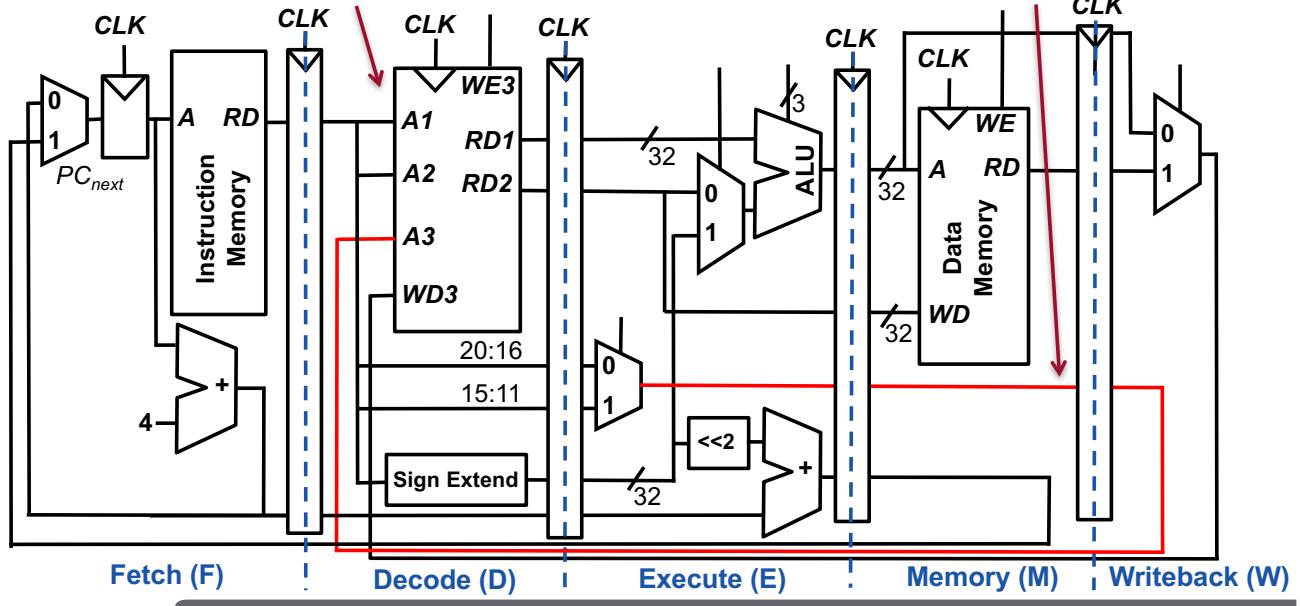
Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86

## Towards a Pipelined Datapath (7/8) Writeback Stage

Note that the register file is read in the decode stage, but written to in the writeback stage

The address must be forwarded to the correct stage!



David Broman  
dbro@kth.se

Part I  
Pipeline and  
Datapath

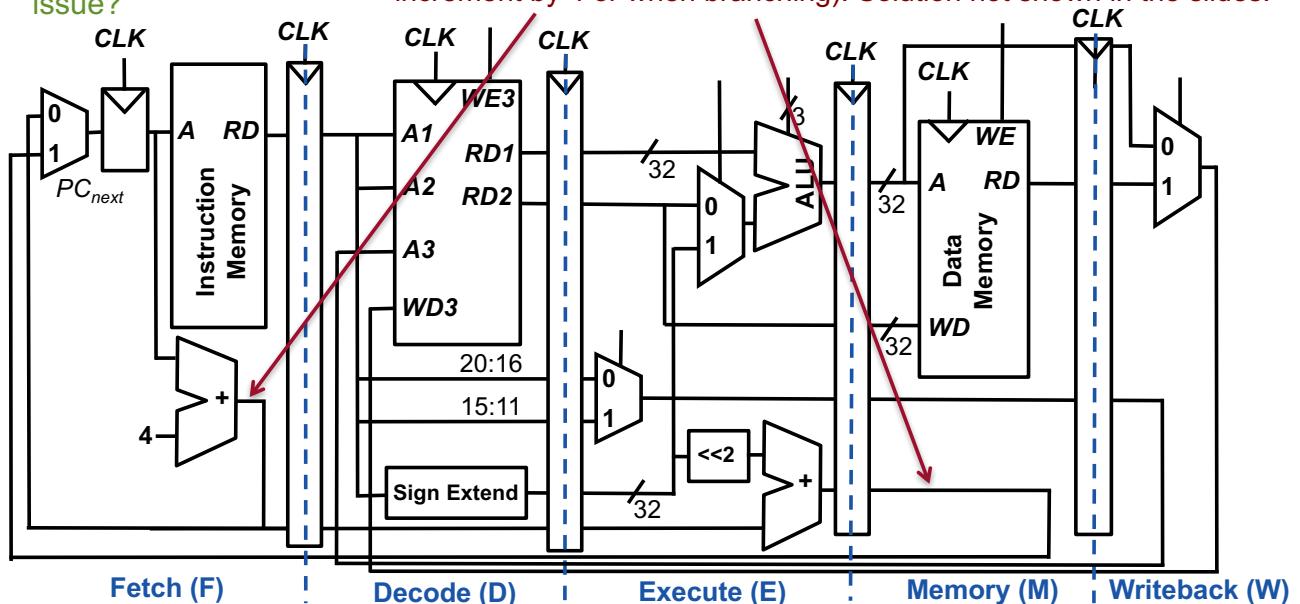
Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86

## Towards a Pipelined Datapath (8/8) Another issue

Can you see another issue?

The program counter can be updated in the wrong stage (PC increment by 4 or when branching). Solution not shown in the slides.



David Broman  
dbro@kth.se

Part I  
Pipeline and  
Datapath

Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86

## Part II

# Data and Control Hazards



Acknowledgement: The structure and several of the good examples are derived from the book "Digital Design and Computer Architecture" (2013) by D. M. Harris and S. L. Harris.

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86



## A Five-Stage Pipeline

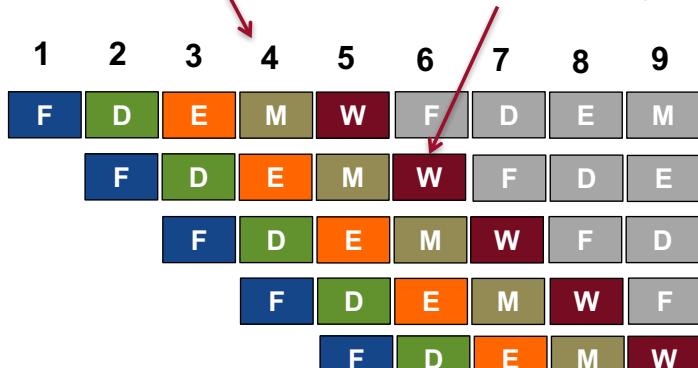
24  
E

In each cycle, a new instruction is fetched, but it takes 5 cycles to complete the instruction.

add \$s0, \$s1, \$s2  
sub \$t0, \$t1, \$t2  
addi \$t3, \$0, 55  
xori \$t4, \$t5, 100  
and \$t6, \$s3, \$s4

In each cycle all stages are handling different instructions in parallel.

**Example.** In cycle 6, the result of the **sub** instruction is written back to register **\$t0**.



We can fill the pipeline because there are no dependencies between instructions

**Exercise:** What is the ALU doing in cycle 5?

**Answer:** Adding together values 0 and 55

David Broman  
dbro@kth.se

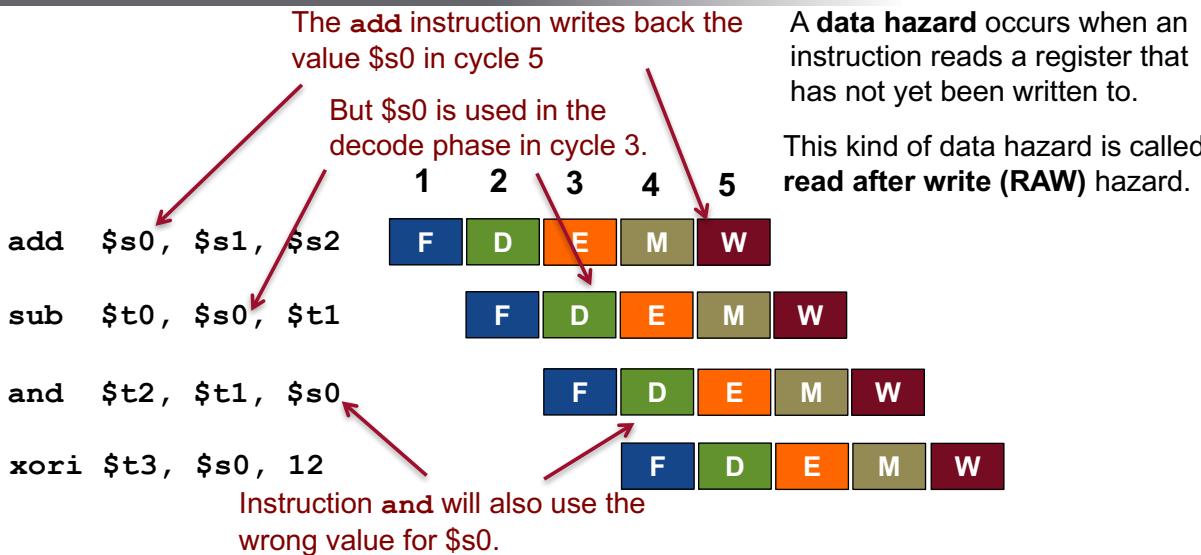
**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

# Data Hazards (1/4)

## Read after Write (RAW)



**Exercise:** For MIPS, will instruction **xori** result in a hazard? Stand for yes, sleep for no.

A **data hazard** occurs when an instruction reads a register that has not yet been written to.

This kind of data hazard is called **read after write (RAW)** hazard.

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

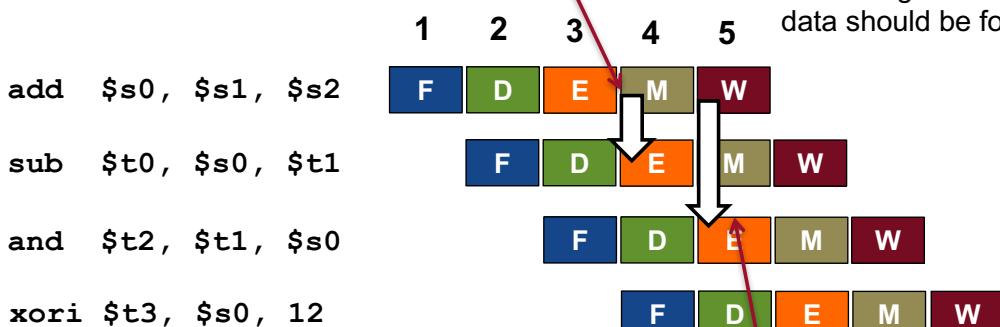


# Data Hazards (2/4)

## Solution 1: Forwarding

The result from the execute stage for **add** can be **forwarded** (also called bypassing) to the execute stage for **sub**.

Hazard detection is implemented using a **hazard detection unit** that gives control signals to the datapath if data should be forwarded.



Can all data hazards be solved using forwarding?

The **and** instruction's hazard is solved by forwarding as well.

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

# Data Hazards (3/4)

## Solution 1: Forwarding (partially)

**Exercise:** Which of the instructions `sub`, `and`, and `xori` have data hazards? Which can be solved using forwarding?

**Answer:**

Hazards: `sub` and `and`

Can use forwarding: `and`

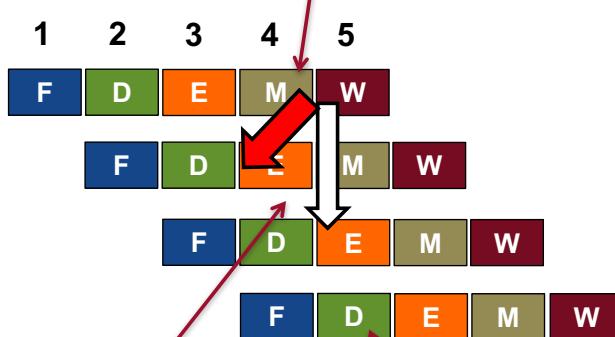
`lw $s0, 20($s2)`

`sub $t0, $s0, $t1`

`and $t2, $t1, $s0`

`xori $t3, $s0, 12`

The `sub` instruction cannot be solved using forwarding because the memory access is available at the end of cycle 4, but is needed in the beginning of cycle 4.



The `and` instruction memory result can be forwarded after the memory stage to execution.

`xori` can read the data from the write stage (writes in first part of cycle, reads in second part)

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

# Data Hazards (4/4)

## Solution 2: Stalling

Solution when forwarding does not work: **stalling**

After stalling, the result can be forwarded to the execute stage.

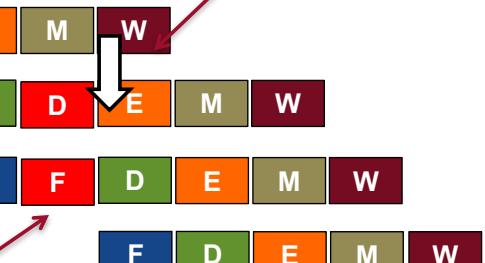
`lw $s0, 20($s2)`

`sub $t0, $s0, $t1`

`and $t2, $t1, $s0`

`xori $t3, $s0, 12`

We need to stall the pipeline.  
Stages are repeated and the fetch of `xori` is delayed.



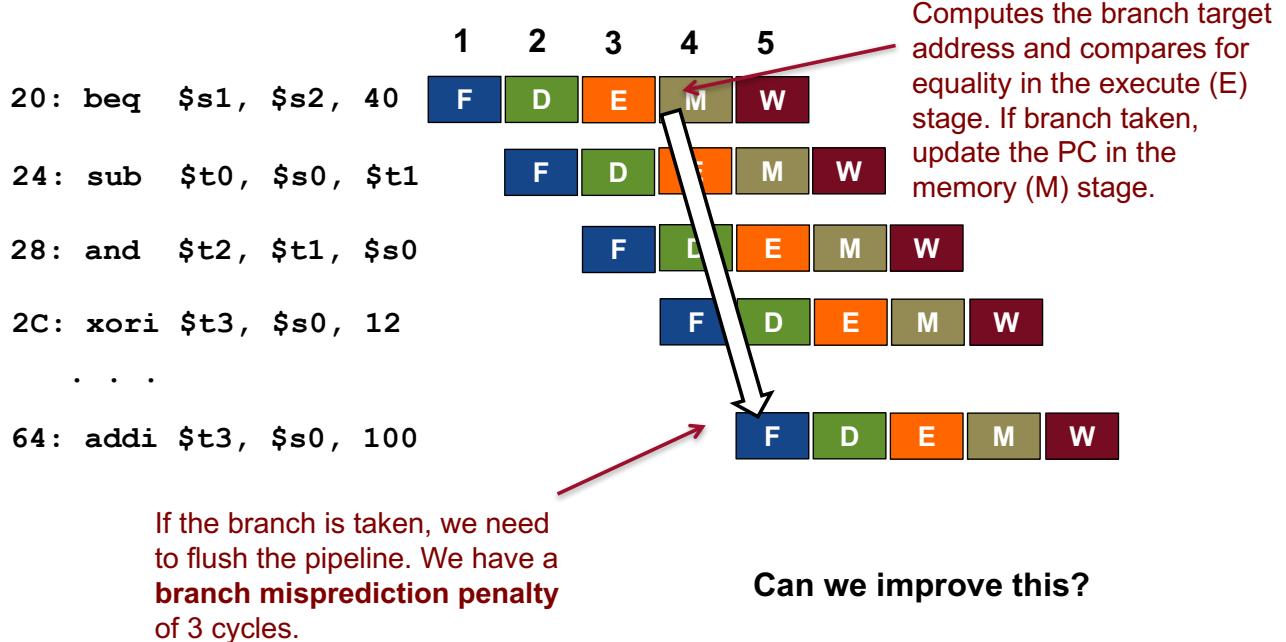
Stalling results in more than one cycle per instruction. The unused stage is called a **bubble**.

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86



David Broman  
dbro@kth.se

Part I  
Pipeline and  
Datapath

Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86



# Control Hazards (2/5)

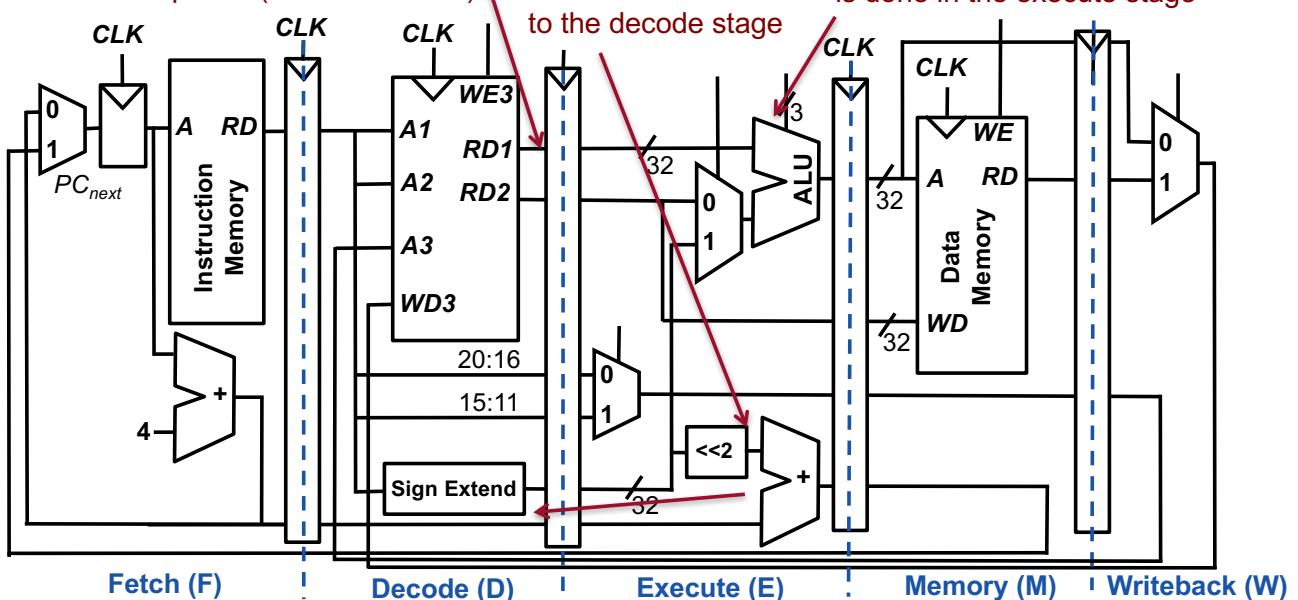
## Improving the Pipeline

30  
E  
E

Add an equality comparison for `beq` in the decode phase (not shown here)

Move the branch address calculation to the decode stage

Right now, branch comparison is done in the execute stage

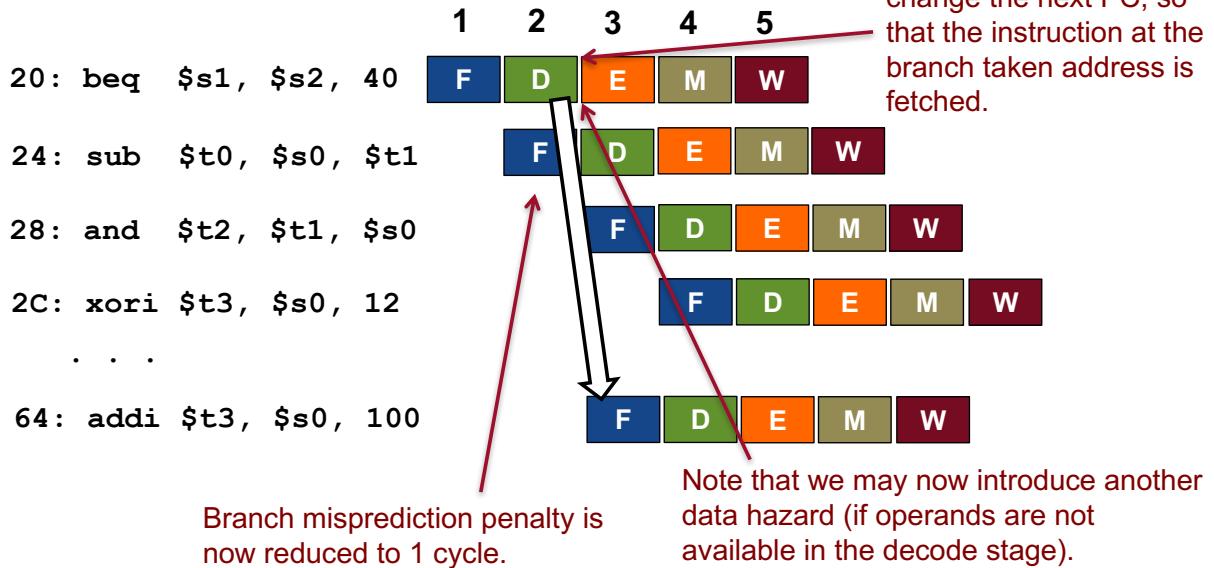


David Broman  
dbro@kth.se

Part I  
Pipeline and  
Datapath

Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86



David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86



# Control Hazards (4/5)

## Deeper Pipelines

32  
E

Why do we sometimes want more stages than 5?

The critical path can be shorter with less logic in the slowest stage.

The processor can have higher clock frequency.

For instance, Intel's Core 2 duo has more than 10 pipeline stages.

Why not always have more pipeline stages?

Adds hardware (registers)

The branch misprediction penalty increases!

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

How can we handle deep pipelines,  
and minimize misprediction?



### Static Branch Predictors

- Statically (at compile time) determine if a branch is taken or not. For instance, predict branch not taken.

### Dynamic Branch Predictors

- Dynamically (at runtime) predict if a branch will be taken or not.
- Operates in the fetch state.
- Maintains a table, called the **branch target buffer**, that contains hundreds or thousands of executed branch instructions, their destinations, and information if the branches were taken or not.

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

 **Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86



## Part III

### ARM and x86



by Raysonho @ Open Grid Scheduler / Grid Engine - Own work. Licensed under Creative Commons Zero, Public Domain

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

 **Part II**  
Data and  
Control Hazards

 **Part III**  
ARM and  
x86



## ARMv7

The most popular ISA for **embedded devices**  
(9 billion devices in 2011, growth 2 billion a year).

More complex addressing modes than MIPS (can do shift and add of addresses in registers in one instruction)

Condition results are saved in special flags: negative, zero, carry, overflow.

ARMv7

16 registers, each 32-bit (integers)

Instruction size 32-bit (Thumb-mode, 16-bits encoding).

MIPS

Conditional execution of instructions, depending on condition code.

Example 1: ARM Cortex-A8, a processor at 1GHz, 14-stage pipeline, with branch predictor.

Example 2: A6 by Apple, manufactured by Samsung, used in iPhone 5.

David Broman  
dbro@kth.se

Part I  
Pipeline and  
Datapath

Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86



## x86

Standard in laptops, PCs, and in the cloud

CISC –instructions are more powerful than for ARM and MIPS, but requires more complex hardware

x86 architecture has evolved over the last 35 years,

There are 16, 32, and 64 bits variants.

8 general purpose registers (eax, ebx, ecx, edx, esp, ebp, esi, edi).

Variable length of instruction encoding  
(between 1 and 15 bytes)

Arithmetic operations allow destination operand to be in memory.

Major manufacturers are Intel and AMD.

David Broman  
dbro@kth.se

Part I  
Pipeline and  
Datapath

Part II  
Data and  
Control Hazards

Part III  
ARM and  
x86

## You know the drill... ...keep calm



David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86

## Reading Guidelines



### Module 4: Processor Design

Lecture 9: ALU and Single-Cycled Processors  
• H&H Chapters 5.2.4, 7.1-7.3.

Lecture 10: Pipelined processors  
• H&H Chapters 7.5, 7.8.1-7.8.2, 7.9

**Reading Guidelines**  
See the course webpage  
for more information.

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86



# Summary

## Some key take away points:

- Pipelining is a **temporal** way of achieving parallelism
- **Pipelining processors** improve performance by reducing the clock period (shorter critical path)
- Pipelining introduces pipeline hazards. There are two main kind of hazards: **data hazards** and **control hazards**.
- Data hazards are solved by forwarding or stalling
- Control hazards are solved by flushing the pipeline and improved by **branch prediction**.



**Thanks for listening!**

David Broman  
dbro@kth.se

**Part I**  
Pipeline and  
Datapath

**Part II**  
Data and  
Control Hazards

**Part III**  
ARM and  
x86