Computer Architecture. Week 9

Alexander Tormasov

Innopolis University

a.tormasov@innopolis.ru

October 21, 2021

Topic of the lecture

- The Processor

Content of the class

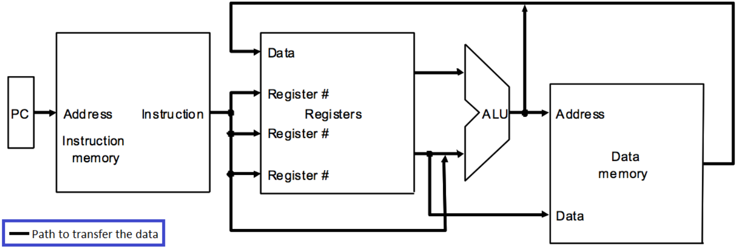
- Datapath and Control Signals
- Architecture Implementations
- The Building Blocks of Processor
- Steps in Designing the Processor
- Pipelining
- Stages of Pipeline
- Pipeline Datapath and Control Signal

Datapath and Control

- How to implement an architecture in hardware?
- **Processor**
  - **Datapath:** Datapath is the path that the input data follows in a processor to appear as an output. A datapath is a collection of functional units – such as arithmetic logic unit that performs data processing operations – registers, and buses.
  - **Control:** Control signals

Datapath

- Simplified view:



NOTE:

PC is Program Counter. It is also known as instruction pointer.

Architecture Implementations

- Multiple implementations for a single architecture
  - **Single-cycle:** All operations take the same amount of time – a single cycle
  - **Multicycle:** It allows faster operations to take less time than slower ones, so overall performance can be increased
  - **Pipelined:** Each instruction is broken up into series of steps multiple instructions execute at once

Recap: Processor Performance

- Program execution time

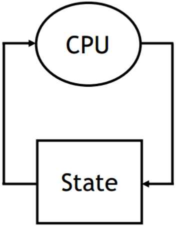
  `CPU Time = #instructions` $\times$ `CPI` $\times$ `Clock cycle time`

- CPU performance factors
  - Instruction count – Determined by Instruction Set Architecture (ISA) and compiler
  - CPI and cycle time – Determined by implementation of the processor

- Challenge is to satisfy constraints of
  - Cost
  - Power
  - Performance

Computers are state machines

- Computer is just a big fancy state machine.
- The processor keeps reading and updating the state, according to the instructions in some program.



Architectural State

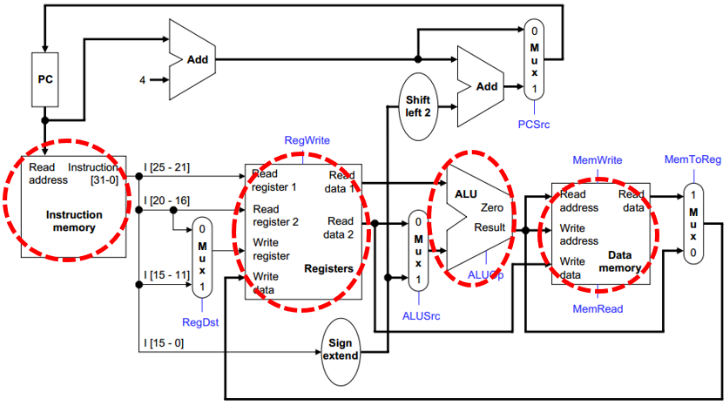
- Determines everything about a processor
  - PC (Program Counter)
  - CPU Registers
  - Memory

Note: Architectural state includes register files and memory, and microarchitectural state represents internal state that has not yet been exposed outside the processor (For example: instruction queue state).

- Consider subset of MIPS instructions
  - Memory-reference instructions: `lw`, `sw`
  - R-type instructions: `add`, `sub`, `and`, `or`, `slt`
  - Branch instructions: `beq`, `bne`

Datapath and Control Signals

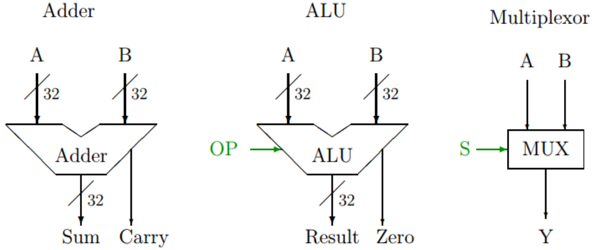
- Final view:



- NOTE: More details in tutorial session.

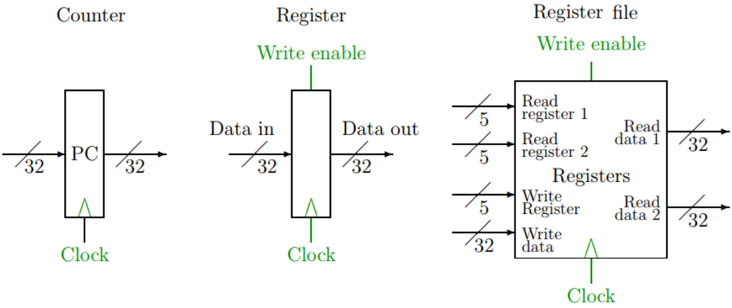
The Building Blocks of Processor (1/2)

- We have already designed many of the major components for the processor, or have at least identified how they could be implemented.



- Note: The diagram highlights the control signals in green color.

The Building Blocks of Processor (2/2)



Steps in Designing a Processor

- Register Transfer Language (RTL) is a kind of intermediate representation that is very close to assembly language.
- It is used to describe data flow at the register-transfer level of an architecture.
- From the RTL description of each instruction, determine
  - The required datapath components
  - The datapath interconnections
- Determine the control signals required to enable the datapath elements in the appropriate sequence for each instruction.
- Usually, it is not exposed to programmers.

Register Transfer Level (RTL)

- It is a design abstraction and used to describe instructions.
- It models the flow of data between registers.
- Any instruction cycle starts by fetching the instruction.
- Typically, every instruction involves operations and modification of the PC.

RTL Example

- The `ADD` instruction: `add rd, rs, rt`

  1. `mem[PC]` Fetch the instruction from memory

  2. `R[rd]` $\leftarrow$ `R[rs] + R[rt]` Set register `rd` to the value of the sum of the contents of registers `rs` and `rt`

  3. `PC` $\leftarrow$ `PC + 4` Calculate the address of the next instruction

- NOTE: More details in tutorial session.

Remarks! Datapath and Control

- A datapath contains all the functional units and necessary connections to implement an instruction set architecture.
- The control unit tells the datapath what to do, based on the instruction that's currently being executed.

John von Neumann (1/2)

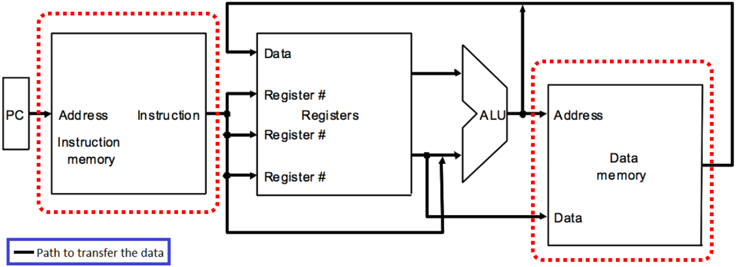
- In the old days, "programming" involved actually changing a machine's physical configuration by flipping switches or connecting wires.
  - A computer could run just one program at a time.
  - Memory only stored data that was being operated on.

John von Neumann (2/2)

- Then around 1944, John von Neumann and others got the idea to encode instructions in a format that could be stored in memory just like data.
  - The processor interprets and executes instructions from memory.
  - One machine could perform many different tasks, just by loading different programs into memory.
  - The "stored program" design is often called a Von Neumann machine
- In modern computers, programs are stored in the "text segment" of memory, which can be written to only by the operating system.

Memories: Harvard Architecture

- It's easier to use a Harvard architecture at first, with programs and data stored in separate memories.

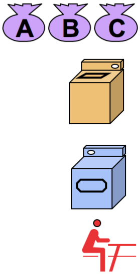


Multicycle Processor

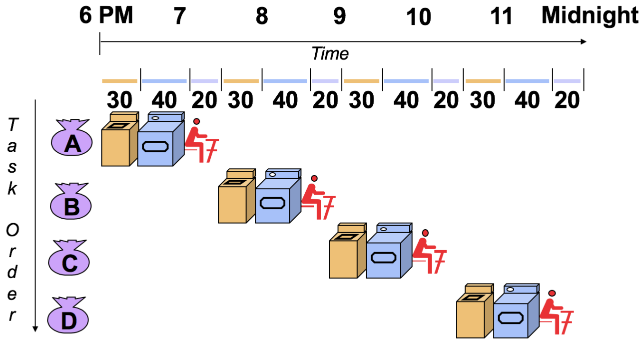
- A multicycle processor fixes some shortcomings
  in the single-cycle CPU.
  - Faster instructions are not held back by slower ones.
  - The clock cycle time can be decreased.
  - A multicycle processor requires a somewhat simpler datapath

Pipelining (1/4)

- Laundry Example
  - Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold.
  - Washer takes 30 minutes
  - Dryer takes 40 minutes
  - “Folder” takes 20 minutes

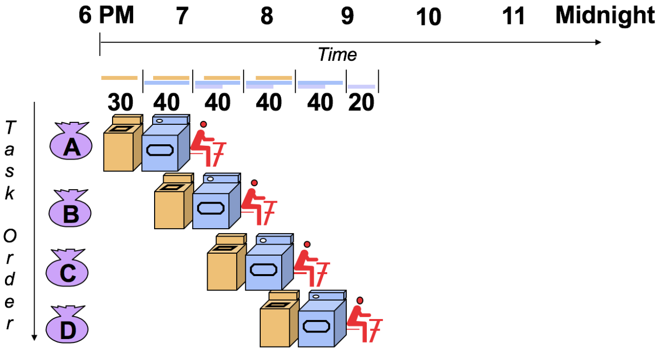


Pipelining (2/4)



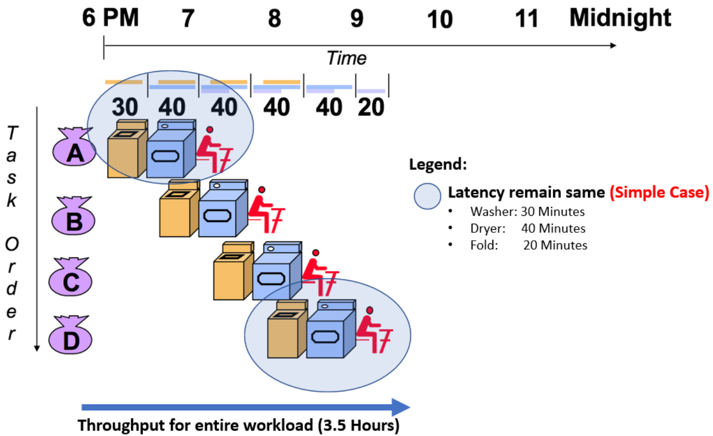
- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelining (3/4)



- Pipelined laundry takes 3.5 hours for 4 loads

- Pipelining doesn't help latency of single task, it helps throughput of entire workload



Pipeline Concept in Computer Architecture

- Break up instruction into tasks
- Balance the amount of work (time) between stages
- Allow each segment to complete and start next instruction

Properties of Pipelines

- **Latency:** The time it takes for a single instruction to execute. Pipelining makes latency ”slightly worse”.
- **Throughput:** The number of instructions executed per unit time. Pipelining improves throughput.

- **Pipelining Idea:** The idea behind pipelining is to maximize the usage of the available resources by overlapping the execution of several tasks.

Question: Why pipelining leads to a ”slightly” worse latency?

Properties of Pipelines

- **Latency:** The time it takes for a single instruction to execute. Pipelining makes latency "slightly worse".
- **Throughput:** The number of instructions executed per unit time. Pipelining improves throughput.

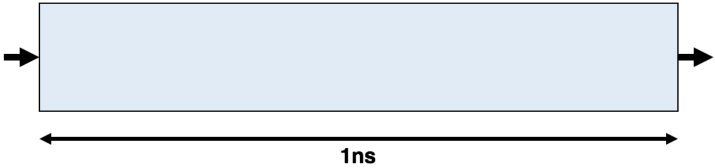
- **Pipelining Idea:** The idea behind pipelining is to maximize the usage of the available resources by overlapping the execution of several tasks.

Question: Why pipelining leads to a "slightly" worse latency?

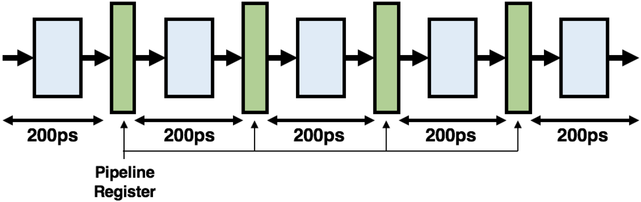
Answer: The length of a clock cycle must account the execution time of the longest pipeline stage.

Pipelining a Digital System (1/2)

- Key idea: Break big computation up into pieces



- Separate each piece with a pipeline register



Pipelining a Digital System (2/2)

- The idea behind pipelining is to maximize the usage of the hardware by overlapping the execution of several instructions.

Stages of Pipeline

- Five stages in classical pipeline
  - Stage 1: Instruction Fetch (IF)
  - Stage 2: Instruction Decode (ID)
  - Stage 3: Execute (EX)
  - Stage 4: Memory (MEM)
  - Stage 5: Write Back (WB)
- Clock is constrained by slowest stage of pipeline.
- Pipelining is not free: complexities in design and additional resources (more later).

Stage 1: Instruction Fetch (IF)

- Fetch an instruction from memory at every cycle
  - Use Program Counter (PC) to index memory
  - Increment PC (assume no branches for now)
- Write state to the pipeline register (IF/ID)
  - The next stage will read this pipeline register

Stage 2: Instruction Decode (ID)

- Decodes operation code (opcode) bits
  - Set up Control signals for later stages
- Read input operands from register file
  - Specified by decoded instruction bits
- Write state to the pipeline register `(ID/EX)`
  - `Opcode`
  - Register contents
  - `PC + 4`
  - Control signals

Stage 3: Execute (EX)

- Perform ALU operations
  - Calculate result of instruction
  - Control signals select operation
  - Contents of register 1 used as one input
  - Either register 2 or constant offset (from instruction) used as second input
- Calculate PC-relative branch target (if any)
  - `PC + 4 + (constant offset)`
- Write state to the pipeline register (EX/Mem)
  - ALU result, contents of register 2, and `PC + 4 + offset`
  - Control signals

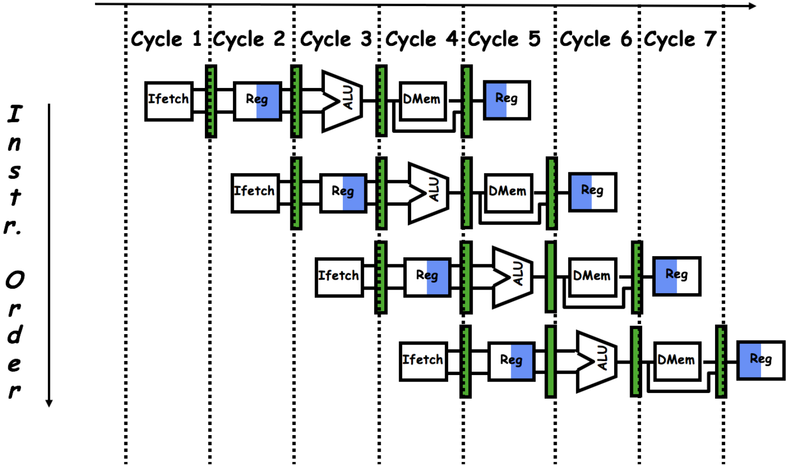
Stage 4: Memory (MEM)

- Perform data operations
  - ALU result contains address for `ld` or `sw`
  - Operation code bits control read/write and enable signals
- Write state to the pipeline register (Mem/WB)
  - ALU result and loaded data
  - Control signals

Stage 5: Writeback (WB)

- Writing result to register file (if required)
  - Write loaded data to destination register for `lw`
  - Write ALU result to destination register for arithmetic instruction
  - Operation code bits control register write enable signal

The Basic Pipeline For MIPS



A Pipeline Diagram

	1	2	3	4	5	6	7	8	9
`lw $t0, 4($sp)`	**IF**	**ID**	**EX**	**MEM**	**WB**				
`sub $v0, $a0, $a1`		**IF**	**ID**	**EX**	**MEM**	**WB**			
`and $t1, $t2, $t3`			**IF**	**ID**	**EX**	**MEM**	**WB**		
`or $s0, $s1, $s2`				**IF**	**ID**	**EX**	**MEM**	**WB**	
`addi $sp, $sp, -4`					**IF**	**ID**	**EX**	**MEM**	**WB**

- A pipeline diagram shows the execution of a series of instructions.
  - The instruction sequence is shown vertically, from top to bottom.
  - Clock cycles are shown horizontally, from left to right.
  - Each instruction is divided into its component stages.

- This clearly indicates the overlapping of instructions. For example, there are three instructions active in the third cycle above.
  - `lw` instruction is in its execute stage.
  - `sub` is in its Instruction decode stage.
  - `and` instruction is just being fetched.

Note: Not all instructions require MEM stage (e.g. R-type instructions). Still, for simplicity, we depict all 5 stages for every instr.

Pipeline Terminology

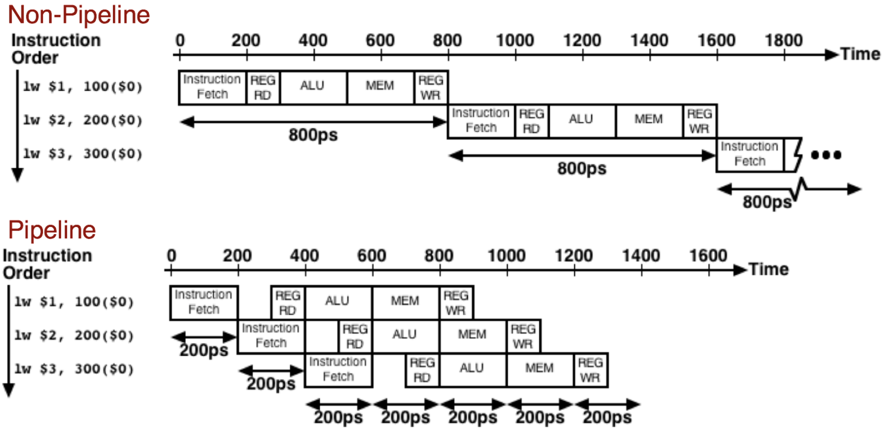
	1	2	3	4	5	6	7	8	9
`lw $t0, 4($sp)`	**IF**	**ID**	**EX**	**MEM**	**WB**				
`sub $v0, $a0, $a1`		**IF**	**ID**	**EX**	**MEM**	**WB**			
`and $t1, $t2, $t3`			**IF**	**ID**	**EX**	**MEM**	**WB**		
`or $s0, $s1, $s2`				**IF**	**ID**	**EX**	**MEM**	**WB**	
`addi $sp, $sp, -4`					**IF**	**ID**	**EX**	**MEM**	**WB**

- The pipeline depth is the number of stages – in this case, five.
- In the first four cycles here, the pipeline is filling, since there are unused functional units.
- In cycle 5, the pipeline is full. Five instructions are being executed simultaneously, so all hardware units are in use.
- In cycles 6-9, the pipeline is emptying

Exceptions in Pipelining

- Pipeline provides ability for processor to handle exception, save state, and restart without affecting program execution.
- Pipeline is restartable
- All processors support this feature now, as it is needed to implement virtual memory

Single-Cycle vs. Pipeline Execution



What about Control Signals?

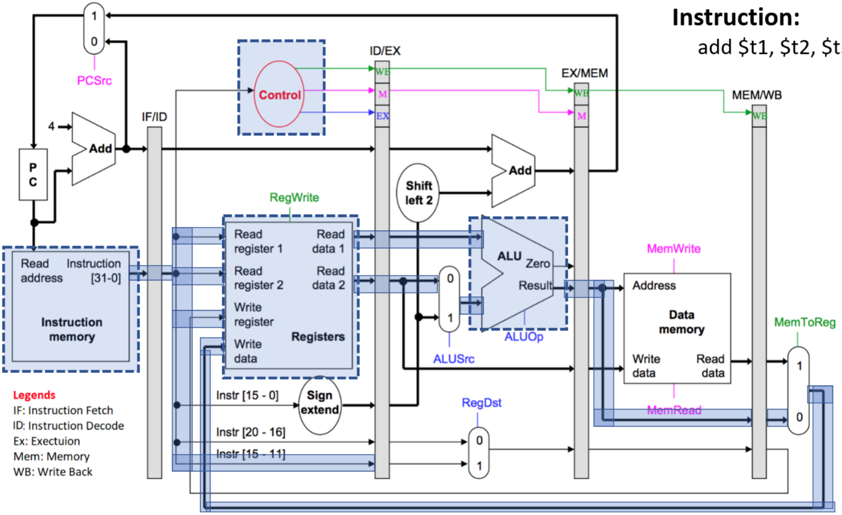
- The control signals are generated in the same way as in the single-cycle processor – after an instruction is fetched, the processor decodes it and produces the appropriate control values.
- But just like before, some of the control signals will not be needed until some later stage and clock cycle.
- These signals must be propagated through the pipeline until they reach the appropriate stage. We can just pass them in the pipeline registers, along with the other data.

What about Control Signals?

- Control signals can be categorized by the pipeline stage

Stage	**Control signal needed**		
EX	ALUSrc	ALUOp	RegDst
MEM	MemRead	MemWrite	PCSrc
WB	RegWrite	MemToReg	

Note: The details will be discussed in tutorial.



Instruction:
add \$t1, \$t2, \$t3

Legends
- IF: Instruction Fetch
- ID: Instruction Decode
- Ex: Exectuion
- Mem: Memory
- WB: Write Back

Remarks on Pipelining (1/2)

- Pipelining attempts to maximize hardware usage by overlapping the execution stages of several different instructions.
- Pipelining offers amazing speedup.
  - The CPU throughput is dramatically improved, because several instructions can be executing concurrently.
  - In the best case, one instruction finishes on every cycle, and the speedup is equal to the pipeline depth.

Remarks on Pipelining (2/2)

- The bad news
  - Instructions can interfere with each other **hazards**
  - It may increases the latency
  - Different instructions may need the same piece of hardware (e.g., memory) in same clock cycle
  - **For Example:** Instruction may require a result produced by an earlier instruction that is not yet complete
- All these details are in next lecture!!

Summary

- Datapath and Control Signals
- Architecture Implementations
- The Building Blocks of Processor
- Steps in Designing the Processor
- Memories
- Pipelining and its stages
- Pipeline Datapath and Control Signal

Acknowledgements

- This lecture was created and maintained by Muhammad Fahim, Giancarlo Succi and Alexander Tormasov