



# Politechnika Wrocławska

## Computer Architecture and Organization

### Lecture 3

Dr. Radosław Michalski

Department of Computational Intelligence, Faculty of Computer Science  
and Management, Wrocław University of Science and Technology

Version 1.0, spring 2017



# Source and licensing

The most current version of this lecture is here:  
<https://github.com/rmhere/lecture-comp-arch-org>

This material is licensed by Creative Commons Attribution  
NonCommercial ShareAlike license 4.0 ([CC BY-NC-SA 4.0](#)).



# Overview of this lecture

**How to access the assembly code?**

**RISC architecture**

**MIPS architecture**



# How to access the assembly code?

## Exemplary source code

Assume that you write your code in C++. How can you find out how the assembly code will look like?

hello.cc:

```
#include <iostream>
int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

```
$ g++ -O2 -S hello.cc -o hello.asm
```



# RISC architecture

## Introduction

- ▶ CISC architecture
  - ▶ more complex instructions
  - ▶ varying instruction lengths
  - ▶ more CPU cycles
- ▶ RISC architecture
  - ▶ simpler and fixed length instructions
  - ▶ faster execution
- ▶ nowadays, the distinction is not that easy



# RISC architecture

## History

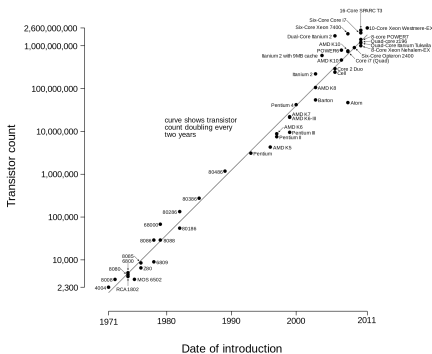
- ▶ early attempts to simplify:
  - ▶ Seymour Cray's CDC 6600 (1964)
  - ▶ John Cocke's IBM 801 (1975 - 1980)
- ▶ DARPA's VLSI Program
  - ▶ 100,000 transistors in CPU in 1970's
  - ▶ goal: automating the design process
  - ▶ outcomes: CAD workstation (Stanford University Network), standardized Unix implementation (Berkeley Software Distribution), Berkley RISC, Stanford MIPS

## RISC architecture

## Transistor count

Transistor count - see this Wikipedia page

## Microprocessor Transistor Counts 1971-2011 & Moore's Law



Wgsimon, CC BY-SA 3.0



# RISC architecture

## RISC architectures (ISAs)

- ▶ **Low end and mobile systems**
  - ▶ ARM®
  - ▶ MIPS™
  - ▶ Hitachi™ SuperH
  - ▶ Atmel® AVR®
  - ▶ RISC-V
  - ▶ Tensilica Xtensa®
- ▶ **High end RISC and supercomputing**
  - ▶ MIPS™
  - ▶ IBM® Power®
  - ▶ Oracle® SPARC®
  - ▶ HP® PA-RISC
  - ▶ DEC™ Alpha
  - ▶ RISC-V





# RISC architecture

## RISC-V - free and open RISC ISA



- ▶ project started in 2010
- ▶ originated in University of California, Berkeley, US
- ▶ goal: small, fast, low-power architecture
- ▶ BSD license



# RISC architecture

## Popular ISAs

### Video

Engineering8 - Top 10 Popular CPU Instruction Set Today



# RISC architecture

RISC vs. CISC considerations

## Video

Microchip Makes - Ask Alf: RISC vs. CISC Architecture



# MIPS architecture

## History

- ▶ **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages - **MIPS™**
- ▶ originated in Stanford University - John L. Hennessy (1981)
- ▶ commercialized by MIPS Technologies
- ▶ first chip: R2000 (1985)
- ▶ in 2013 Imagination Technologies acquired MIPS Technologies



# MIPS architecture

## Goals

- ▶ simple load-store instruction set
- ▶ fixed instruction set encoding
- ▶ pipelining



# MIPS architecture

## Where to find it?

- ▶ embedded systems:
  - ▶ Nintendo® 64 - NEC VR4300 (based on MIPS R4300i)
  - ▶ Sony® PlayStation® 2 - MIPS R5900-based "Emotion Engine"
  - ▶ Sony® PSP® - MIPS R4000
- ▶ supercomputing systems:
  - ▶ SGI® Challenge (R4400, R8000, R10000)



# MIPS architecture

## Specification - MIPS64

- ▶ registers
  - ▶ 32 64 bits general-purpose registers
  - ▶ 32 floating-points registers
  - ▶ special registers, e.g., floating-point status register
- ▶ data types:
  - ▶ 8-,16-,32-,64-bits for integer data
  - ▶ 32-bit single precision for floats
  - ▶ 64-bit double precision for floats
- ▶ addressing:
  - ▶ immediate
  - ▶ displacement
  - ▶ memory is byte addressable with a 64-bit address



# MIPS architecture

## Instructions

- ▶ load and store
  - ▶ Any of the general-purpose or floating-point register may be loaded or stored
- ▶ ALU Operations
  - ▶ register – register instructions
- ▶ branch and jumps
  - ▶ all branches are conditionals (compare instructions, which compare two registers)
- ▶ floating point
  - ▶ IEEE 754 format





# MIPS architecture

## Pipelining

- ▶ key implementation technique used to make fast CPUs
- ▶ multiple instructions are overlapped in execution
- ▶ takes advantage of parallelism that exists among the actions needed to execute an instruction
- ▶ each step in the pipeline completes a part of an instruction – these steps are called pipe stages or pipe segments
- ▶ time required between moving an instruction one step down the pipeline is a processor cycle
- ▶ assuming ideal conditions, the time per instruction on the pipelined processor is equal to: time per instruction on unpipelined machine divided by the number of pipe stages



# MIPS architecture

## RISC pipeline

### Classic RISC pipeline:

- ▶ instruction fetch (IF)
- ▶ instruction decode and register fetch (ID)
- ▶ execute (EX)
- ▶ memory access (MEM)
- ▶ register write back (WB)



# MIPS architecture

## RISC pipeline - schema

Instr No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7



# MIPS architecture

## MIPS64 - studying the details

### MIPS64 architecture details

- ▶ Introduction
- ▶ Volume I-A: Introduction to the MIPS64 Architecture
- ▶ Volume II-A: The MIPS64® Instruction Set Reference Manual



# MIPS architecture

## Instruction set

### Instructions

- ▶ length: fixed, 32 bits
- ▶ types: three types of instructions: R, I, J
- ▶ direct hardware implementation vs. pseudoinstructions



# MIPS architecture

## Instruction types

- ▶ **R-type** - register type instruction
- ▶ **I-type** - immediate type instruction
- ▶ **J-type** - jump type instruction



# MIPS architecture

## R-type - register

- ▶ the most complex type
- ▶ works only on registers (we provide their addresses)

$B_{31-26}$	$B_{25-21}$	$B_{20-16}$	$B_{15-11}$	$B_{10-6}$	$B_{5-0}$
opcode	register s	register t	register d	shift amount	function

**Table 1:** R-type instruction

- ▶ `add $rd, $rs, $rt`
- ▶  $R[d] = R[s] + R[t]$
- ▶ 6-bit opcode?



# MIPS architecture

## I-type - immediate

$B_{31-26}$	$B_{25-21}$	$B_{20-16}$	$B_{15-0}$
opcode	register s	register t	immediate

**Table 2:** I-type instruction

- ▶ `addi $rd, $rs, immed`
- ▶  $R[t] = R[s] + (IR_{15})^{16} IR_{15-0}$
- ▶ IR - instruction register, the register where the current instruction is stored
- ▶  $(IR_{15})^{16}$  means that bit  $B_{15}$  of the instruction register (which is the sign bit of the immediate value) is repeated 16 times
- ▶  $IR_{15-0}$  - the 16 bits of the immediate value





# MIPS architecture

## J-type - jump

$B_{31-26}$	$B_{25-0}$
opcode	target

**Table 3:** J-type instruction

- ▶ j target
- ▶  $PC \leftarrow PC_{31-28} \text{ IR}_{25-0} \text{ } 00$
- ▶ PC - the program counter, which stores the current address of the instruction being executed.
- ▶ update the PC by using the upper 4 bits of the program counter, followed by the 26 bits of the target (which is the lower 26 bits of the instruction register), followed by two 0's, which creates a 32-bit address



# MIPS architecture

## Recommended videos

- ▶ EngMicroLectures - History of ISAs
- ▶ MR Trick - How a CPU is made
- ▶ Computer History Museum - MIPS: Risking It All on RISC



# RISC and MIPS architectures

## Sources

- ▶ Imagination Technologies Limited, *MIPS64 Architecture*, Hertfordshire, UK (technical documentation)
- ▶ M. Esponda and R. Rojas, *"The RISC Concept - A Survey of Implementations"*, Freie Universitat Berlin, Berlin, Germany (technical report)
- ▶ K. Keville, *"Introduction to RISC-V"*, R&D Labs at MIT, 2016 Stanford HPC Conference (video)
- ▶ J. Kwiatkowski, *"Computer Architecture and Organization"*, Wrocław University of Science and Technology (course materials)