COMPSYS 304: Computer Architecture

Lecture Notes

Nicholas Russell

Tuesday, September 3, 2024

Table of contents

Introduction

These notes are compiled from the lectures of COMPSYS 304. They are intended as a personal reference to help with assignments, exam preparation, and understanding key concepts in Computer Architecture.

The notes are organised by lecture and cover a wide range of topics from basic computer architecture principles to more advanced subjects like MIPS implementation and performance analysis. Feel free to add personal insights, additional readings, or questions as you review this material.

Organisation of the Notes

The notes are divided by lecture content, with each section corresponding to a specific set of topics. Here's how they're organized:

- **Lecture 1-3: Basics of Computer Architecture**
  Key topics include Instruction Set Architecture (ISA), memory hierarchy, and basic CPU organisation.
- **Lecture 4-6: MIPS Architecture**
  Covers the MIPS instruction set, control flow, and subroutine handling.
- **Lecture 7-9: CPU Implementation**
  Focuses on different methods for implementing CPUs and the trade-offs involved.
- **Lecture 10-12: Digital Circuits and Datapath Design**
  Reviews fundamental digital circuit concepts and discusses the design of a MIPS datapath.
- **Lecture 13-15: Performance Analysis**
  Provides an in-depth look at how different design choices impact CPU performance.

How I Use These Notes

These notes are a living document, and I intend to update them as I gain a deeper understanding of the material. Here's how I use them:

- **Quick Reference**: For quick lookups, the Table of Contents will help me jump directly to the relevant section.
- **In-Depth Study**: For exam preparation, I'll revisit each section, ensuring I understand each concept before moving on.
- **Personal Insights**: I'll be adding my own thoughts, additional notes from readings, and potential questions for further study.

I might also include exercises or practical examples that help solidify my understanding of the more complex topics.

Part I

Lectures 1 - 3

1 Introduction & Course Overview

This chapter provides an introduction to the course, some background, and an overview of the course itself.

1.1 Background

In the last six decades, computer technology has made incredible progress due to innovations in both **semiconductor technology** and computer **architecture**.

1.1.1 What do we mean from *Performance*?

The relative performance can be measured by standard benchmarks, which depend on the specific target applications.

Performance depends on clock frequency and other factors, as discussed more in later chapters.

1.1.2 Intel X86 Processor from 1978 to 2018:

- **Intel 8086** - (1978): 1 core, 1 W, 5 - 10 MHz
- **Core i7-8086K** - (2018): 6 cores, 95 W, 4 GHz

1.2 Course Details

1.2.1 Learning outcomes

The main learning outcomes of this course are:

- **To understand the basics of modern computer architectures and quantitative principles of computer design in order to develop a conceptual understanding of issues involved in designing a high performance computer system.**

- **To use and apply this knowledge to design computer systems or select computers for specific tasks. This course will give you an understanding of the effects of design decisions on system performance and makes you a well-informed consumer in addition to a processor designer.**

Recommended Textbook:

- David A. Patterson and John L. Hennessey, Computer Organization and Design: The Hardware/Software Interface, Fifth edition, 2013 by Elsevier/Morgan Kaufmann Publishers (or 3rd or 4th editions).
- Lecture notes provided on canvas (these will be summarised in these notes).

1.2.2 Course Overview (learning outcomes)

Part 1: *In this part you will learn to:*

- Design and evaluate the instruction set architectures (both RISC and CISC) and how it can be related to the hardware/software interface in a computer system with a quick review of assembly programming.
- Understand different processor implementation methods including the basic single-cycle implementation and how it can be extended to a multi-cycle, pipelined and superscalar implementations.
- Understand performance evaluation techniques and their relation to the target applications and the processor workload.

Part 2: *In this part you will learn to:*

- Understand the memory hierarchy in a modern computer system and its impact on the performance of the system. This includes physical and virtual memory systems and basics of cache memories.
- Understand some basic principles of parallel computing using special topics in this course (more advanced materials covered in some elective courses).

1.2.3 Assessment

Three assignments and one test in addition to the final exam. The first assignment is mainly on the instruction set architecture design, hardware/software interfacing and review of assembly programming. The second assignment is related to processor implementation and performance issues. The third assignment is related to memory hierarchy system and multiprocessing.

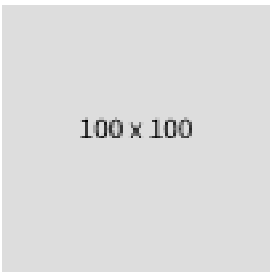
The test only covers the first part of the course. The final exam covers the whole course.

- **Assignment 1: 8% due Fri. 9 August.**
- **Assignment 2: 7% due Fri. 23 August.**
- **Assignment 3: 15% due Fri. 4 October.**
- **Test: 20% (in week 7, Wednesday 11 September)**
- **Final exam: 50%**

2 Basics of Computer Architecture

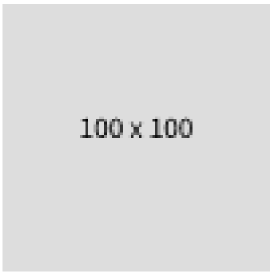
This chapter begins the journey into the content of this course.

2.1 What is Computer Architecture?



- **ISA:** The programmer-visible instructions — ISA serves as the boundary between the hardware and software.
- **Organization:** high-level aspects of computer design (e.g. the internal design of the CPU, the memory system, the interconnection structure, ...)
- **Hardware:** detailed logic design, circuit-level design, packaging technology, etc.

2.2 Review: Memory Organization

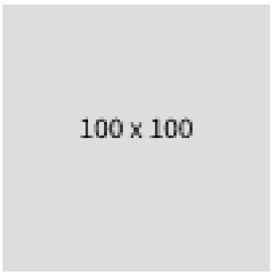


- Memory is viewed as a large, one-dimensional array, which has an address range.
- A memory address is an index into the array
- The index usually points to a byte of memory.
- To access a larger word (e.g., a 32-bit word or 4 bytes), four consecutive memory locations are accessed.

Memory access time is the amount of time required to read (usually one byte) or write data (usually one byte) from/to memory.

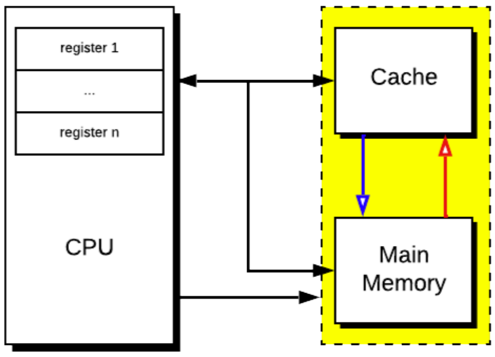
2.3 An Overview of Memory Speed vs. CPU Speed

The rate of memory speed increase has been less than processors speed increase



Fast memory is more expensive per bit than slow memory, so a **memory hierarchy** is often used to give a performance close to the fastest memory with a lower average cost per bit.

2.4 Processor and Memory Hierarchy



Processor Registers: The smallest and fastest memory for CPU.

Typically, 32 to 64 registers, each of them may be 32 or 64 bits with typical access time of *nanoseconds or less*

Cache Memory: Slower than registers.

Typically 8 to 256 Kbytes with an access time about a *few nanoseconds or less*

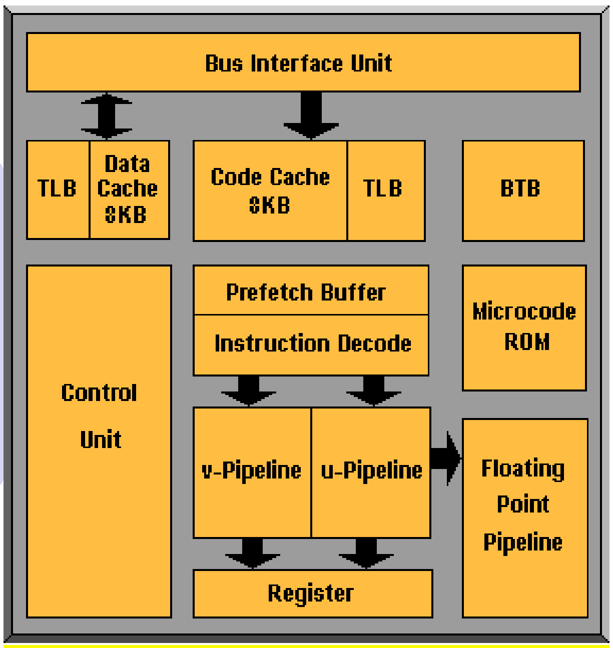
Main Memory: Slower than cache memory.

Hundreds of Mega bytes with an access time about *tens of nanoseconds*

2.5 Example of Intel Processors

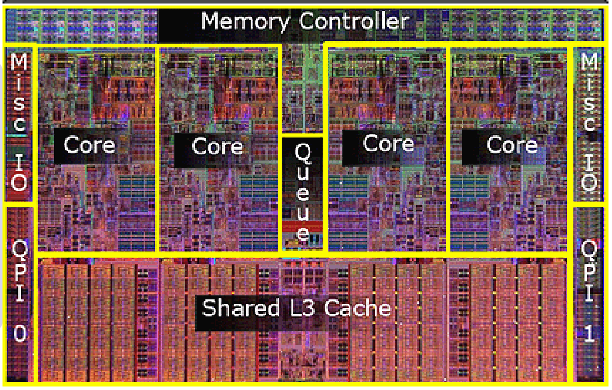
Pentium Processor Architecture (simplified)

An example of architectural innovations made by processor designers (to improve computer systems performance) (Source: Intel)



Intel Core i7

The Core i7 die and major components. (more than 700 million transistors in a die with area 263 mm^2 in 45 nm technology). (Source: Intel)



3 Instruction Set Architecture (ISA)

An ISA is the **interface between hardware and low-level software**

Some modern instruction set architectures:

80x86, Itanium, PowerPC, **MIPS**, SPARC, HP, **ARM**, **RISC-V**

Issues with a fixed ISA:

- **advantage:** different implementations of the same architecture

Example

- AMD and Intel chips use the same ISA, while their implementations are different.
- Pentium II and Celeron have different memory systems and different clock rates.
- Core i7, Core i5, Core i3, and Atom are different implementations of the same ISA (ignoring some ISA extensions).

- **disadvantage:** sometimes prevents using new innovations.

ISA design: What is needed to be included in each *machine instruction?*

1. What operation is to be performed by the CPU.

arithmetic/logic operations, data transfer, control transfer, etc.

2. How data for the operation can be provided.

through memory, through CPU registers, encoded in the instruction word

3. How the result is to be stored.

in memory or register

Therefore, the ISA must define:

- The operation to be performed (***opcode***)
- Data type and size of operands
- Location of operands and results
- Instruction format or Encoding

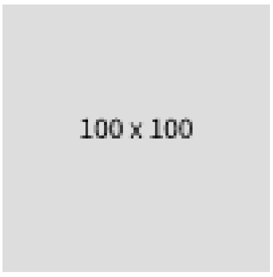
3.1 Classifying Instruction Set Architectures

Each instruction has (an) **opcode** which defines the type of operation and may have zero to three **operands** for input data and the result destination.

The type storage used for operands is the most basic differentiation among different types of Instruction Set Architectures (ISA).

- operands from/to memory
- operands from/to registers
- operands from memory/to register or vice versa

3.1.1 Stack-based Architecture

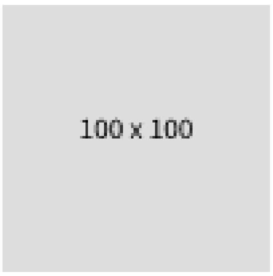


The pointer to the ***top of the stack*** will be adjusted properly after completing the ALU (**A**rithmetic / **L**ogic **U**nit) operation to point to the result.

Instruction:
ADD
ADD has three implicit operands, which are from/to the **stack**.

Two special instructions **PUSH** and **POP** are used to write into the top of the stack or read from the top of the stack respectively. The ***stack pointer*** will be adjusted properly after completing these operations.

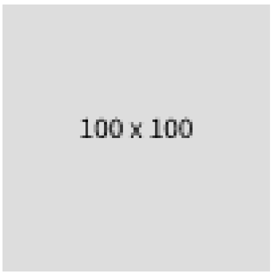
3.1.2 Accumulator-based Architecture



Instruction:
ADD ***mem_address***
ADD has two implicit operands (Accumulator register) and one explicit operand (memory).

Two special instructions **LOAD** and **STORE** are used to read from the memory (i.e. copy the contents of the given memory location into accumulator register), and write into memory (i.e. copy the contents of the accumulator into given memory location) respectively.

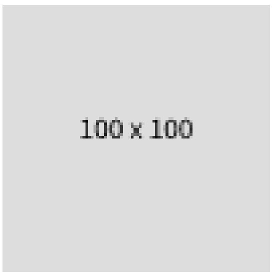
3.1.3 Register-Memory Architecture



Instruction:
ADD Rs, Rd *mem_address*
ADD has three operands. Rd and Rs represent the destination and source registers respectively. The other source operand is provided through memory.

Two special instructions **LOAD** and **STORE** are used to read from the memory (i.e. copy the contents of the given memory location into a given register), and write into memory (i.e. copy the contents of the specified register into given memory location) respectively.

3.1.4 Register-Register (load/store) Architecture



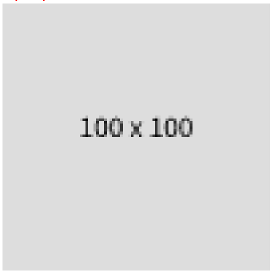
Instruction:
ADD Rs, Rd, Rt
ADD has three operands. Rd, Rs, and Rt represent the destination and source registers respectively.

Two special instructions **LOAD** and **STORE** are used to read from the memory (i.e. copy the contents of the given memory location into a given register), and write into memory (i.e. copy the contents of the specified register into given memory location) respectively.

Example

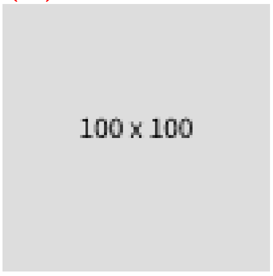
Write the instructions for the expression **C = A + B** for the four classes of instruction sets. It is assumed that A, B and C are located in memory addresses 1000, 2000 and 3000 respectively.

(a) Stack-based architecture



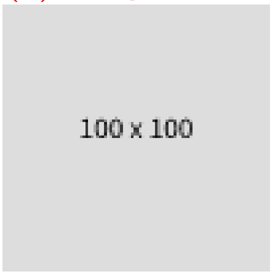
```
PUSH 1000
PUSH 2000
ADD
POP 3000
```

(b) Accumulator-based architecture



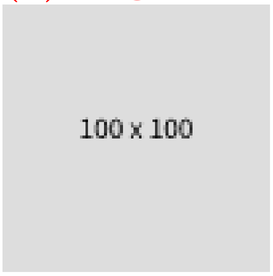
```
LOAD 1000
ADD 2000
STORE 3000
```

(c) Register-Memory architecture



```
LOAD R1, 1000
ADD R2, R1, 2000
STORE R2, 3000
```

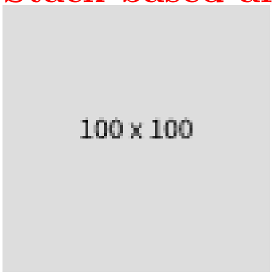
(d) Register-Register (load/store) architecture



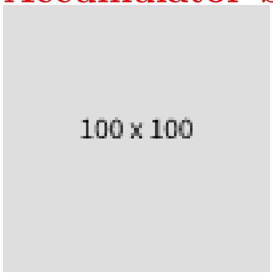
```
LOAD R1, 1000
LOAD R2, 2000
ADD R3, R1, R2
STORE R3, 3000
```

3.1.5 Summary: Different types of ISA

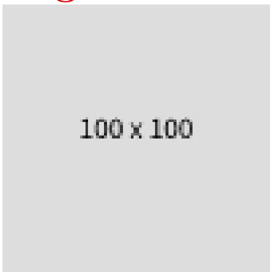
Stack-based architecture



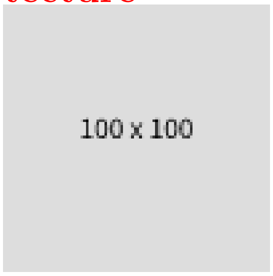
Accumulator-based architecture



Register-Memory architecture



Register-Register (load/store) architecture



4 Instruction Set Architecture Design

4.1 Comparing different Instruction Set Architectures.

Which one is the best? (for our example: C = A + B)

Stack	**Accumulator**	**Register-Memory**	**Register-Register**
Push 1000	Load 1000	Load R1, 1000	Load R1, 1000
Push 2000	Add 2000	Add R2, R1, 2000	Load R2, 2000
Add	Store 3000	Store R2, 3000	Add R3, R1, R2
Pop 3000			Store R3, 3000

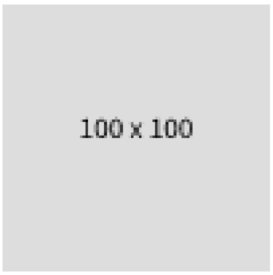
Basic ISA Classes

Instruction set architectures (ISA) are generally classified based on: - the instruction word size (how many bytes for encoding each instruction) - the number of different instructions in the ISA - the number of clock cycles required to complete each instruction (which can be implementation dependent)

Based on that, the following have been specified as different ISA classes:

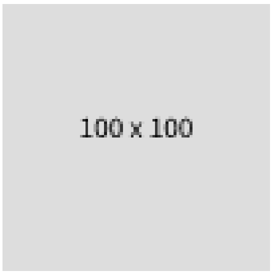
- **RISC** (Reduced Instruction Set Computers): where the size of the instruction word for all instructions are the same. This may lead to simpler decoding hardware. For example, **MIPS** (or RISC-V) processors use this type of ISA.
- **CISC** (Complex Instruction Set Computers): where the size of the instruction word may be different for different instructions. This results in more complex decoding hardware but the code footprint (binary code size of the program) can be less. for example, **Intel X86** based processors are based on this type of ISA.
- **EPIC** (Explicitly Parallel Instruction Computers): In this case, parallel operations are explicitly encoded in the instruction. The compiler plays a more important role in EPIC architectures. For example, **Intel Itanium** is based on this type of ISA.

4.2 Different levels of abstraction (Software)



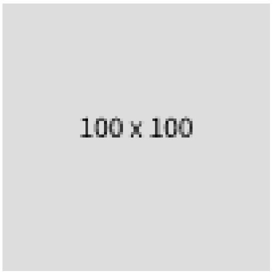
An abstraction removes unnecessary detail, helps us cope with the complexity. More hardware details are considered in programming through moving from **High-Level Language** to **Machine Language**.

4.3 Basic steps in a CPU for Instruction Processing



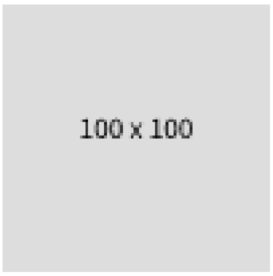
Fetching

Access memory to get the next instruction: Activate memory read signal, place the right memory address on address lines, read the content of memory pointed by the address.



Decoding

Interpret the bits of the instruction word: to identify the operation and its data (which might be taken from memory or registers)



Execution

Perform that specific operation: Use the processor resources to perform the operation and write the result to memory or register if necessary

- What type of operations are needed?

- How operands (data) are provided in each instruction?

through registers except for *load* and *store* instructions which will access memory too.

- Size of Instruction word and Data word?

32 bits

4.4 Summary of Our Design Decisions:

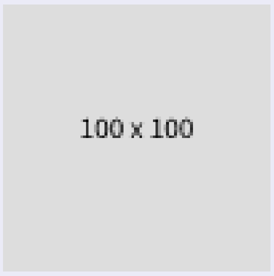
- **RISC ISA:** All instructions have the same size
  - **load/store architecture** (only *load* and *store* instructions can access memory).
  - Other instructions use registers
- **Data word:** 32 bits (4 bytes)
- **Instruction word:** 32 bits (4 bytes)
- **Registers** 32 general purpose 32-bit registers
- **Size of Addressable Memory:** \$ 2^32 \$ (each memory address is 32 bits)
- **Operands:** Signed, unsigned, immediate

To make the implementation simpler and faster, the number of instructions in the ISA should be reduced as much as possible
ALU operations are performed on: - data from registers, or - immediate value (data encoded into the instruction word)

ALU operations

- **addition**
  Both input data from registers (signed) add *add, Rd, Rs, Rt*
  Both input data from registers (unsigned) addu *addu, Rd, Rs, Rt*
  One input data as immediate value addi *addi, Rt, Rs, immed.*
- **subtraction**
  Both input data from registers (signed) sub *sub, Rd, Rs, Rt*
  Both input data from registers (unsigned) subu *subu, Rd, Rs, Rt*
- **AND**
  Both input data from registers (signed) and *and, Rd, Rs, Rt*
  One input data as immediate value andi *andi, Rt, Rs, immed.*
- **OR or, ori**

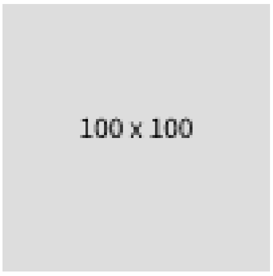
Shift operations



Example

Assume that the contents of register R1 is as follows:
1000 0000 1100 1010 0001 0111 0001 1110
After shifting two times to the left the following number is in register R1:
0000 0011 0010 1000 0101 1100 0111 1000

sll, sllv shift left logical ***sllv Rd, Rs, Rt***
srl, srlv shift right logical ***srl Rd, Rs, immediate***
sra, srav shift right arithmetic ***srav Rd, Rs, Rt***



Shift instructions can be used in arithmetic operations (multiplying or dividing by a number which is a power of 2). The number of bits to be shifted can be given as a constant value or the contents of the second source register can be used to determine how many bits to be shifted.

A few points to remember:

- In the instruction representation (in our case), always the *destination register* is written first and then the *source registers* (this convention may be different for other processors).
- Registers in the instructions are represented using $reg_num. (for example, register 1 is $1 and register 15 is $15).
- In our ISA (MIPS ISA) the contents of Register 0 ($0) is 0000 0000 0000 0000 0000 0000 0000 0000 and register $ cannot be written. (It is hardwired to 0)

4.5 *Examples of the MIPS ALU Instructions:*