Hierarchy of Computer Architecture
Final Project, CIS-242-AA-CRN94413

Leigh Johnson
johnsonl6@my.smccd.edu
Cañada College

December 2, 2023

Contents

1 Hierarchy of Computer Architecture

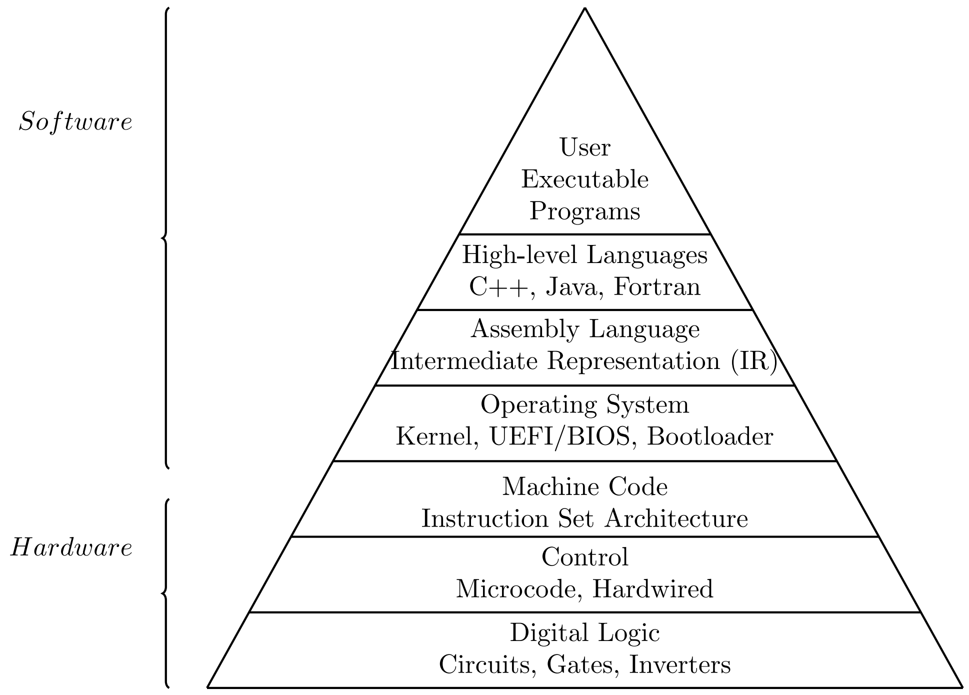

Figure 1: **Abstract Levels of a Computer System**

Computer architecture hierarchy refers to the structured organization of components within a computer system. This hierarchy typically starts with the smallest, simplest elements like transistors and builds up to more complex structures such as logic gates, microprocessors, and finally an entire computer operating system capable of running user programs. The components become more integrated and complex at each level of this hierarchy.

2 Digital Logic

The **Digital Logic** layer includes the physical components of the computer, like circuits, gates, and wires. Logic gates are the fundamental building block and implement operations using **Boolean Logic**.

2.1 Boolean Algebra

Named after mathematician George Boole, Boolean laws and identities can be used to reduce/simplify a logical statement. In the context of computer science, simpler circuits are preferred because they consume fewer resources (energy, money) and are simpler to build.

Name	AND form	OR form
$Identity$	$1x = 1$	$0 + x = x$
$Null$	$0x = 0$	$1 + x = 1$
$Idempotent$	$xx = x$	$x + x = x$
$Inverse$	$xx' = 0$	$x + x' = 1$
$Null$	$0x = 0$	$1 + x = 1$
$Commutative$	$xy = yx$	$x + y = y + x$
$Associative$	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
$Distributive$	$x + yz = (x + y)(x + z)$	$x(y + z) = xy + xz$
$Absorption$	$x(x + y) = x$	$x + xy = x$
$DeMorgan's$	$(xy)' = x' + y'$	$(x + y)' = x'y'$
$DoubleComplement$	$(x)'' = x$	$(x)'' = x$

Table 1: Boolean Algebra Identities and Laws

2.2 Truth Tables

Also known as **Karnaugh Maps (K-Maps)**, a **Truth Table** can be represented in two formats:

- **Sum-of-Products form** - collection of ANDed variables (product terms) that are ORed together (sum of all product terms)
- **Product-of-Sums form** - collection of ORed variables (sum terms) that are ANDed together (product all summed terms)

Consider the following example of a function *F(x,y,z) = x'yz + xy'z + xyz*, which outputs true (1) if the majority of inputs are true:

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 2: Truth Table representation for a majority function:
F(x,y,z) = x'yz + xy'z + xyz.

2.3 Logic Gates

Using **Boolean Algebra**, we can construct the next basic building block of digital circuitry: a **Logic Gate**. The most basic logic gates are:

- **AND Gate -** true (1) if both inputs are true, otherwise false (0).
- **OR Gate -** true (1) if either input is true, false (0) if both inputs are false.
- **NOT Gate -** true (1) if the input is false (0), false if the input is true. Also known as an **Inverter**

Two other gates, **NAND** and **NOR**, produce complementary output to AND and OR gates. The NAND gate is called a **universal gate** because any electronic circuit can be constructed using only NAND gates.

3 Hardware Control

Logic Gates are combined to build different types of circuitry and modules. The simplest units are **Integrated Circuits (ICs)**, a chip consisting of the necessary transistors, resistors, and capacitors to implement various gates.

3.1 Hardwired Control

3.1.1 Combinational Circuit

Combinational Circuits use a stateless circuit design, which accepts input values and (almost) instantly produces an output. One of the simplest examples is the **Half Adder**, which outputs the sum of two bits.

3.1.2 Sequential Circuit

The output of **Seqential Circuits** varies depending on the internal state (memory) of the circuit. The state changes are governed by a **Clock**, which produces a regular periodic electrical wave. Sequential circuits typically incorporate a **Feedback Loop**, which loops output back to input terminals.

A **Finite State Machine (FSM)** depicts the relationship between the state of **Flip-Flop circuity** and the system clock. FSMs can only be in one state at a time.

3.2 Microcode

Microcode is a characteristic of **Complex Instruction Set Computers (CISC)**, which are a layer of low-level instructions needed by higher-level machine code instructions. Machine code is an intermediary between the machine language and the hardware, allowing more flexible control of the processor's operations. A **Microprogram** is a sequence of microcode instructions.

3.3 Hardware Components of a Computer

3.3.1 Central Processing Unit (CPU)

The **Central Processing Unit (CPU)** is responsible for fetching program instructions, decoding each instruction, and executing the decoded operations. These steps are known as the **Fetch-Decode-Execute Cycle** or **Instruction Cycle**.

3.3.2 Memory

Memory refers to the devices that store data temporarily or permanently.

- **Random Access Memory (RAM)** - a volatile (temporary) form of memory.
- **Read-Only Memory (ROM)** - a nonvolatile (permanent) form of memory that is read-only.
- **Flash Memory (NAND, NOR)** - a non-volatile (permanent) form of memory that can be erased and rewritten.

3.3.3 Storage

A storage device is used to store data. There are many different kinds of storage mediums available for computers:

- **Magnetic Tape** - a thin strip of plastic coated with a magnetizable material.
- **Optical Disks** - reflective disk read using a laser.

- **Hard Disk Drives (HDD)** - rotating ”platter” disk with a magnetic head and seek arm.
- **Solid-State Drives (SSD)** - non-volatile flash memory.

3.3.4 Bus (Communication)

3.3.5 MARIE Reference Architecture

The CPU in the MARIE reference architecture (an example of **Von Neumann architecture**) consists of the following components:

- **Arithmetic Logic Unit (ALU)** - performs logic operations (e.g., less-than or equals comparisons) and arithmetic (add, subtract).
- **Accumulator (AC)** - a specialized register used for storing the output of the last executed operation.
- **Input / Output Registers** - specialized registers used to store user input and program output.
- **Memory Buffer Register (MBR)** - specialized register used to store data being transferred to/from main memory.
- **Memory Address Register (MAR)** - specialized register used to store the memory address of data to be fetched from main memory.
- **Control Unit (CU)** - responsible for sequencing operations, manipulating the **Program Counter**.
- **Program Counter (PC)** - specialized register used to keep track of the next instruction to be executed.
- **Instruction Register (IR)** - specialized register used to hold the instruction currently being decoded/executed.
- **Main Memory** - also known as the datapath. Stores program data.

4 Machine Code

Machine code is the lowest-level computer language, consisting of binary or hexadecimal instructions.

4.1 Instruction Set Architecture (ISA)

An Instruction Set Architecture (ISA) is an agreed-upon interface between *software* running on a machine and the *hardware* that executes it.

The ISA provides the specification for:

- Word size (e.g., 32-bit or 64-bit)

- Instruction opcodes and max operands
- Instruction length
- Addressing modes (e.g., byte-addressable or word-addressable)
- Supported data types and operations (e.g., IEEE 754 floating point arithmetic)
- Endianness (e.g. Big Endian or Little Endian)
- Branch evaluation
- Instruction Format

4.2 Instruction Formats

- **Stack Architecture** - instructions and operands are popped off the stack. A stack architecture uses **postfix notation** (also known as Polish notation) to express operations.
- **Accumulator Architecture** - one operand of a binary operation is stored in the accumulator, one in memory.
- **General Purpose Register (GPR) Architecture** - operands stored in general-purpose registers.

4.3 Reduced Instruction Set Architecture (RISC) vs. Complex Instruction Set Architecture (CISC)

RISC	**CISC**
Multiple register sets	Single register sets
Many registers (256+)	Few registers (6-16)
Parameter passing via on-chip register windows	Parameter passing via off-chip memory (less efficient)
Single-cycle instructions	Multi-cycle instructions
Hardwired control	Microcode Control
Highly pipelined	Less Pipelined
Few Simple instructions	Many Complex instructions
Fixed-length instructions	Variable-length instructions
Complexity in compiler	Complexity in microcode
Memory access is limited to load/store instructions	Many instructions can access memory

Table 3: Comparison of RISC and CISC attributes.

5 Operating System

An operating system (OS) is the intermediary system software between users and the computer hardware. The OS facilitates the execution of application programs and efficiently manages hardware resources like CPU, memory, and I/O systems.

Operating System code is often referred to as **the Kernel**, especially to delineate between applications that run in **Kernel Space** vs. **User Space**

5.1 Boot Firmware

The following firmware is used during the computer boot stage. This firmware is responsible for initializing and testing system hardware components and loading the Bootloader or operating system.

5.1.1 BIOS

Basic Input/Output System (BIOS) is a legacy firmware interface initially developed by IBM. The UEFI standard has replaced BIOS, however the term "BIOS" is often used to refer to boot firmware in a generic sense.

5.1.2 UEFI

The Unified Extensible Firmware Interface (UEFI) is a specification that defines a software interface between an operating system and platform firmware, replacing the legacy Basic Input/Output System (BIOS) firmware interface. UEFI provides graphical menus, secure boot, and network capabilities before loading an operating system.

6 Assembly Language

Assembly is a human-readable language that compiles to **Machine Code**. Higher-order programming languages in the next section are compiled into Assembly Code or an **Intermediate Representation (IR)**.

Intermediate Representations (IRs) is an internal stage used by a compiler (e.g., LLVM) or virtual machine (e.g., Java Virtual Machine), which allows for optimizations and language/machine-agnostic conversion between source code and machine code.

7 High-level Language

A high-level language abstracts the implementation details of the machine, hardware, and instruction set. Examples of high-level languages include:

- C/C++

- Rust
- Java
- Python

8 User Space

When a user writes and executes a program, this program typically runs in "user space" or "userland." This delineates between system applications that run in "kernel space" or "kernel land" and user programs.