COSC 230: Computer Organization

Alex Zhang

November 10, 2022

Contents

1 Introduction to Computer Organization **2**

2 Memory **4**

3 Cache **6**

4 Binary Numbers **8**

5 Instructing the CPU **10**

6 RISC-V **11**

6.1 Architecture and Application 11

6.2 Instruction Set 12

7 Instruction Pipeline **16**

8 I/O **18**

1 Introduction to Computer Organization

The goal of computer organization is to hide the complexity of the system from the end user. While the computer itself is certainly a complicated machine, the user does not need to understand its intricacies. In contrast, computer scientists and engineers should understand such intricacies in order to create better programs.

Definition ▶ Computer Organization

Computer organization is how functional components are put together to create a computer.

Definition ▶ Abstraction

Abstraction is a way to handle complexity by hiding unnecessary details from the user.

Definition ▶ Software

Software consists of processor instructions written to perform operations.

Generally, software exists in two spaces:

1. User Space – user applications (e.g. Microsoft Word, Google Chrome)
2. System Space – operating system, system tasks

Definition ▶ Operating System

The **operating system** is the software that supports a computer's basic functions, such as scheduling tasks, executing applications, and controlling peripherals.

The operating system is a layer of **abstraction** between the user and hardware. Typically, the user space interfaces with the system space (operating system) through system calls, and the system space interfaces with hardware through I/O.

Definition ▸ Hardware

Hardware consists of the physical components in a computer which operate by routing electrons (i.e. electricity).

Definition ▸ Motherboard

The **motherboard** physically connects all components of the computer. The board is responsible for routing data and power throughout the entire system.

Most PC motherboards include a basic input/output system (BIOS) that allows you to directly interface with the motherboard. This allows you to configure things such as which hard drive to boot from.

Definition ▸ Central Processing Unit (CPU)

The **central processing unit (CPU)** acts as the brain of the computer.

- performs all basic operations of the system
- facilitates data transfer within the system's memory or peripherals
- manages the other components of the system

Most modern CPUs consist of multiple **operating cores**. Each core contains its own cache (memory), arithmetic unit, logic unit, memory management unit, integer and floating-point registers, and a floating point unit.

CPUs also contain...

- System Agent – display agent and memory controller
- Cache – onboard memory; significantly faster than RAM
- I/O Controller – connects I/O devices to the CPU

2 Memory

Definition ▸ Memory Hierarchy

Memory hierarchy tells us what kind of computer storage is the fastest.

1. **Internal:** CPU registers and cache
2. **Main:** the system RAM and controller cards
3. **On-line mass storage:** hard drives
4. **Off-line bulk storage:** flash memory, optical discs, external hard drives

The first three are also called primary, secondary, and tertiary storage.

The main difference between the speed of primary and secondary storage is caused by the physical distance between the storage and processor. Since memory operations occur upwards of billions of times a second, the time for electrons to travel between memory and processor adds up. Registers and cache are physically attached to the CPU while RAM may be inches away.

Definition ▸ Memory Controller

The **memory controller** manages the flow of data going to and from the computer's memory.

Definition ▸ Random Access Memory (RAM)

Random Access Memory (RAM) encompasses all longer term, volatile memory storage in a computer.

Definition ▸ Static RAM

Static RAM uses several transistors to statically lock in a value by feeding back the previous output as an input to a logic circuit (i.e. latch).

- Very stable
- Takes lots of space; unable to pack tightly

Definition ▶ Dynamic RAM

Dynamic RAM uses a capacitor to store a bit's value and one transistor to control the capacitor.

- Small; can be packed tightly in a grid with row and depth fields for the memory controller
- Capacitors passively leak charge, so they must be periodically refreshed ($\sim$ 64 ms)
- Capacitors discharge when reading, so if the bit is 1, we must charge the capacitor after reading
- Charging a capacitor is not instantaneous
- Capacitors store very little charge, so it must be amplified by a **sense amplifier** to be read

DRAM Timings

Because DRAM is typically packed tightly in a grid there are two main operations:

- **Row Address Strobe (RAS)** navigates the memory controller to the proper row
- **Column Address Strobe (CAS)** navigates the memory controller to the proper column

Consequently, we need to properly time these operations and account for latency:

- CL: CAS latency
- tRCD: latency between RAS and CAS

3

Cache

Definition ▸ Cache

Cache is the CPU's onboard memory.

Overall, cache is faster but smaller than RAM, and it is slower but larger than CPU registers. Cache itself is split into multiple **levels** based on their speed, with level 1 being the fastest. The amount of cache levels depends on the CPU.

Whenever we read from cache, we need to search through the cache to find the value. A **cache hit** occurs when we find our value. A **cache miss** occurs when we cannot find our value. Then, the CPU has to ask the memory controller to find the value in RAM.

1. Compulsory Miss: nothing yet in cache; just bring in random values from RAM and hope they are useful
2. Capacity Miss: no more room in the cache
3. Conflict Miss: two memory addresses mapped to the same location

Because cache is fairly small, we only want to store commonly used values in it. There are more intricate ways to make memory reads from cache more efficient. These depend on using a program's past behavior to predict future behavior.

Principle of Spatial Locality

Instead of grabbing just one value at a time, we can grab multiple values around it.

Principle of Temporal Locality

The more often a variable is moved, the lower level of cache it stays at.

There are three kinds of cache:

Types of Cache

1.	Direct-mapped Cache	Byte offset matches byte block size
2.	Set-associative Cache	asdf
3.	Fully-associative Cache	wow

We implement two policies:

1. Write Back: Read/write inside of cache; writes to backing store when eviction occurs
2. Write Through: Write through all levels of cache and RAM; decreased speed

If we run into a capacity miss, we need to **evict** an entry to make room for the new one.

Eviction Policies

1. First in First out (FIFO): evict the oldest entry
2. Least Recently Used (LRU): evict entries that haven't been used in a while
3. Least Frequently Used (LFU): evict entries that aren't frequently used
4. Random

4 Binary Numbers

We are all familiar with the decimal number system where there are ten digits, 0 through 9. This is called a **base 10** number system. Binary uses a **base 2** number system where the only digits are 0 and 1.

Example 4.0.1 ▶ Binary Numbers

Consider the base 10 number 13.

$$\begin{aligned} 1 \cdot 10^1 + 3 \cdot 10^0 &= 10 + 3 \\ &= 13 \end{aligned}$$

The binary representation would be $1101_{(2)}$.

$$\begin{aligned} 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 &= 2^3 + 2^2 + 2^0 \\ &= 8 + 4 + 1 \\ &= 13 \end{aligned}$$

Definition ▶ IEEE–754

- The standard for storing floating point numbers
- `fadd.s`: add single-precision floats
- `fadd.d`: add double-precision floats

Definition ▶ Signed Integer

A **signed integer** uses the most significant bit to denote the number's sign (positive or negative).

- Leftmost bit is 0 if positive or 1 if negative

Definition ▶ Two's Complement (2_c)

To convert any number from positive to negative or vice versa, we use **two's complement**.

1. Flip all bits
2. Add 1

Definition ▶ Logical Shifting

A **logical shift** is a bitwise operation that shifts all bits of a number. In many programming languages, left shift is denoted by $\ll$ and right shift is denoted by $\gg$.

In programming, logical shift can be used a shorthand for multiplying and dividing by powers of two.

$$n \ll k = n \cdot 2^k$$

$$n \gg k = n \div 2^k$$

Example 4.0.2 ▶ Multiplication and Division

$0001\,0100_{(2)}$ is the binary representation for 20.

$$\begin{aligned} 0001\,0100_{(2)} \ll 1 &= 0010\,1000_{(2)} \\ 20 \ll 1 &= 20 \cdot 2^1 = 40 \\ 0001\,0100_{(2)} \ll 2 &= 0101\,0000_{(2)} \\ 20 \ll 2 &= 20 \cdot 2^2 = 80 \\ 0001\,0100_{(2)} \gg 1 &= 0000\,1010_{(2)} \\ 20 \gg 1 &= 20 \div 2^1 = 10 \\ 0001\,0100_{(2)} \gg 2 &= 0000\,0101_{(2)} \\ 20 \gg 2 &= 20 \div 2^2 = 5 \end{aligned}$$

5 Instructing the CPU

This section focuses on **assembly**, the lowest-level language that's still easily readable. In particular, we'll be looking at RISC-V, one of many assembly languages. It sports a reduced instruction set, making it ideal for beginners.

Definition ▸ Assembly (Language)

Assembly refers to any low-level programming language with very strong correspondence between the language's instructions and the architecture's machine code instructions.

Definition ▸ Instruction Set Architecture (ISA)

The **Instruction Set Architecture** is an abstract model of a computer. It defines the support instructions, data types, registers, and other fundamental features.

Definition ▸ Data Segment

A **data segment** is a portion of a program that contains static global variables and static local variables. The size is determined by the size of the values in the source code, thus does not change at runtime.

Definition ▸ Instruction

An **instruction** is a primitive operation which is executed by the CPU. Individual instructions exist as memory which is parsed by the CPU.

RISC-V

Definition ▸ RISC-V

RISC-V is an assembly language. More specifically, RISC-V is an open standard instruction set architecture.

RISC-V source code contains four distinct things:

- **Comment**: notes in the source code that aren't parsed
- **Directive**: tells assembler to take some action or change a setting
- **Label**: denotes different sections of the code; allows ability to jump between labels
- **Instruction**: performs a basic operation

6.1 Architecture and Application

Definition ▸ Stack

The **stack** is a piece of random-access memory used to store **local** variables.

Register **sp** (stack pointer) points to the bottom of the stack.

- Allocate — add sp, sp, -16
-

Definition ▶ Register

Registers are fast memory locations found onboard the CPU. They're easier to access and faster than RAM, making them ideal for storing temporary and constantly accessed data.

RISC-V has 32 integer registers (x0–x31) which can be addressed by their special names:

- **zero**: **always** set to zero
- **ra**: function return address; stores the address of the instruction after a call
- **sp**: stack pointer
- **gp**: global data pointer
- **t0–t6**: temporary storage
- **fp**: frame pointer for function-local stack data
- **s0–s11**: saved registers (if frame pointer is not in use, x8 becomes s0)
- **a0–a1**: arguments passed to functions and function return values
- **a2-a7**: arguments passed to functions

RISC-V also has 32 floating point registers which follow the same scheme.

6.2 Instruction Set

The RISC-V instruction set only has 47 instructions. Despite its size, it is still a turing-complete language.

Computational Instructions

All computational instructions except `lui` and `auipc` take three arguments: destination register, source register, and a second source register or immediate value. Instructions using an immediate value typically end with the letter `i`.

- `add,addi,sub`: addition and subtraction
- `sll,slli,srl,srli,sra,srai`: logical shift
- `and,andi,or,ori,xor,xori`: logical bitwise operation
- `slt,slti,sltu,sltui`: *set if less than*; set destination register to 1 if the first source operand is less than the second, else 0
- `lui`: *load upper immediate*; loads bits 12–31 of destination register with a 20-bit immediate value
- `auipc`: *add upper immediate* to PC and store the result in the destination register (adds 20-bit immediate to the upper 20 bits of the program counter, enabling PC-relative addressing in RISC-V)

Note: immediate values are data stored as part of the instruction itself
Note: `lui` and `auipc` can be used to achieve 32-bit operations by (1) setting bits 12–31 to the upper 20 bits of the value and (2) using `addi` to the lower 12 bits for the complete 32-bit result

Control Flow Instructions

These consist mainly of conditional branching instructions: compare two registers, and based on the result, jump to a label

- `beq,bne,blt,bltu,bge,bgeu` – branch if equal/not equal/less than/less than unsigned/greater than/greater than unsigned
- `jal`: *jump and link*; set destination register to program counter (return address) then jump to statement at target address
- `jalr`: *jump and link*; jump to statement at label and set return address to **ra**

Memory Access Instructions

These instructions transfer data between a register and a memory location. These instructions involve either **b**yte, **h**alfword, or **w**ord.

Load instructions copy data to a destination register. The letter **u** denotes load using zero-extend.

- `lb,lbu,lh,lhu,lw`

Store instructions copy data to a memory address.

- `sb,sh,sw`

Fence: TODO

System Instructions

- `ecall`: invoke a system call by invoking the service specified in register `a7`
- `ebreak`: initiate a debugger breakpoint
- `csrrw,csrrwi,csrrc,csrrci,csrrs,csrrsi`: TODO

Pseudo-Instructions

Pseudo-instructions don't exist on the machine but can be instead expanded by the assembler.

Definition ▶ Machine Code

- A format of 1s and 0s that instruct the CPU
- Every instruction in RISC-V is 32 bits (makes this simple to encode and decode)
- These 32-bits must encode everything (opcode – instruction group's unique number) (parameters – registers and/or immediate values)

Definition ▸ Branches and Jumps

- All branches and jumps are PC-relative
- Branches are encoded with a 12-bit immediate, but only bits [12:1] are encoded
- The 12-bit immediate describes how many bytes away the label is (remember: each instruction is typically 4 bytes)
- Initially a 13-bit signed number, but right-most bit is chopped off since it must be a multiple of 2
-

Instruction Pipeline

For simplicity, we will be looking at a basic example of an instruction pipeline.

RISC-V Pipeline

1. Instruction Fetch (IF)
2. Instruction Decode (ID)
3. Execute (EXE)
4. Memory (MEM)
5. Write-back (WB)

Definition ▸ Data Hazard

Data Hazards are problems that may occur during a pipeline.

Definition ▸ RAW (Read After Write)

Problem: An instruction needs a past instruction's output, so we need to wait until writeback before going to the next instruction.

Solution: Operand Forwarding – directly forward the output of the instruction to the next instruction (immediately after ALU outputs)

Definition ▸ Branch Prediction

Predict which way a branch instruction will go.

- At first, use random guess
- Afterwards, use past data to make accurate predictions
- If right more times than wrong, this will significantly speed up branch instructions
- If wrong, we have to flush pipeline

One bit branch predictor flips prediction as soon as prediction is wrong Two bit branch predictor flips prediction after two wrong predictions

Definition ▶ Scalar Machine

A **Scalar machine** is a machine that can do atmost one instruction per CPU clock cycle.

8 I/O

I/O stands for general input/output. In terms of low-level computing, we will be focusing on devices and components communicating with one another.

Definition ▶ Poll

A device needs to continuously check or **poll** the device's register for any change that might occur.

A device needs to be actively listening to properly receive information from another device. The **polling rate** signifies how often a device checks for inputs.

Definition ▶ Port I/O (PIO)

Port I/O uses special assembly instructions to communicate with a dedicated I/O bus.

Definition ▶ Memory-mapped I/O (MMIO)

Memory-mapped I/O maps the device's registers to RAM's address space. It uses the memory controller to arbitrate.

MMIO is the preferred method for simpler, embedded systems since a dedicated I/O bus is not necessary.

Definition ▶ Character Oriented I/O

Character Oriented I/O uses two status registers to receive and transmit information.

We will typically only read and write one character at a time, usually the size of one byte. The size of the character can vary depending on the system.

Definition ▶ Block Oriented I/O

Block Oriented I/O uses RAM to communicate blocks of data at a time. Status registers are still used to:

- Communicate which memory address to look at
- Notify the hardware device that something happened

Typically, large I/O systems will use block-oriented I/O.

Transferring data over a wire is simply send voltage across or not and then wait until the clock on the other side has had a chance to sample it. However, as we know with wires, they can be subjected to interference. This mainly occurs when some other sort of electric device is nearby. Furthermore, there is resistance in a wire. Therefore, the strong voltage sent on one end is not “felt” on the other end.

To shield from signal interference, we have two techniques:

1. Differential Signaling: Use two intertwined wires to send the same information, but one has inverted information. That way, if there is interference, it will affect both wires, so we can still recover data.
2. Shielding: Wrap the wire with material that intercepts interference and drives it to ground

For long-distance cables, we usually use both techniques, or we use fiber-optic cables instead.