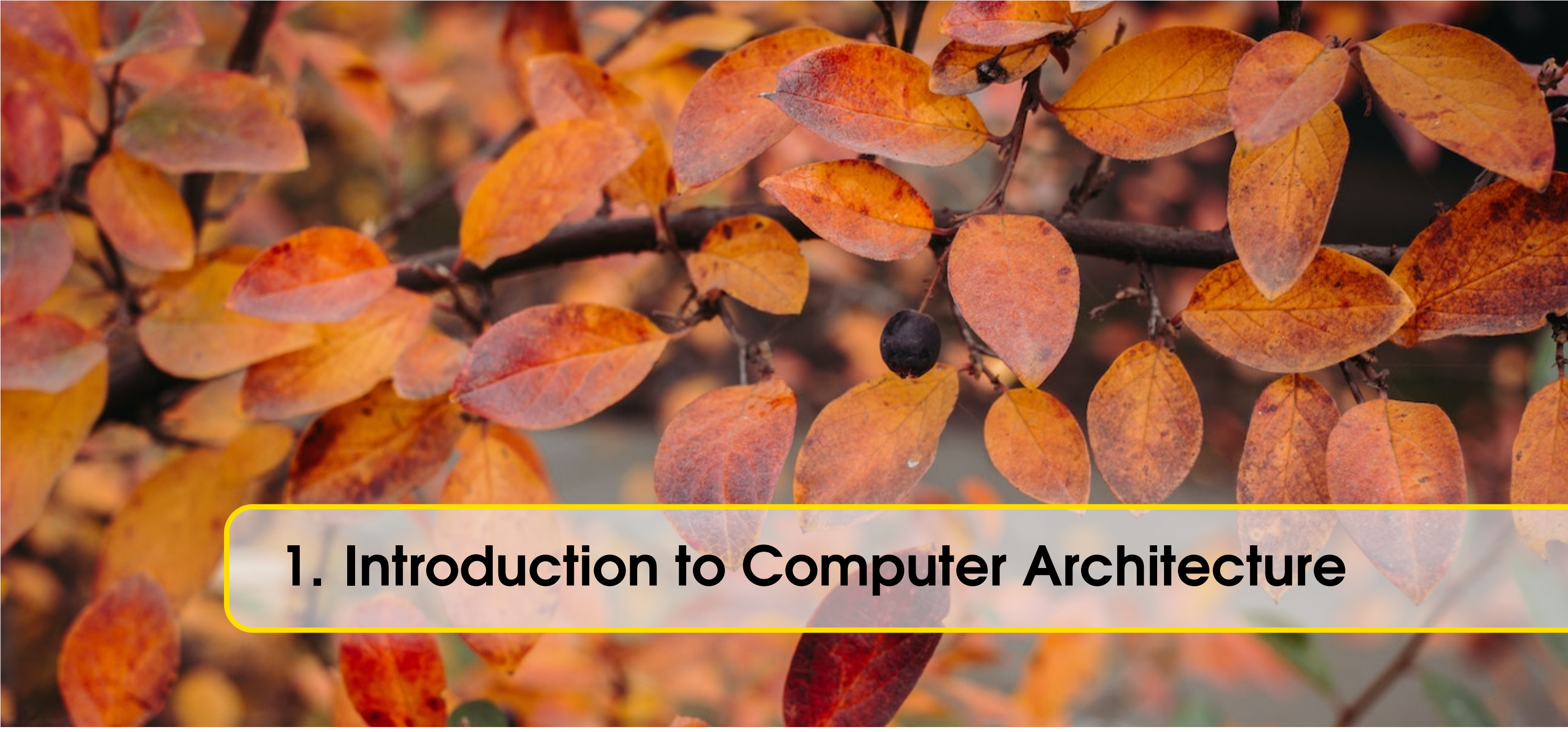


1. Introduction to Computer Architecture

Objectives: Introduction to Computer Architecture

In this chapter, we will:

1. Review briefly computer components;
2. discuss the fundamental ideas that guided computer design;
3. Learn computer performance and how to measure it;
4. Finally, describe the two families of computers and their characteristics.

1.1 Computer Components

1.1.1 Central Processing Unit (CPU)

The central processing unit (CPU), also called processor, is the active part of the computer. The processor logically comprises two main components: datapath and control (brawn and brain). Datapath Components of the processor that perform arithmetic operations. It includes the memory, registers, adders, ALU, and communication buses. It is called data path because data follows a specific path when executing an instruction. 2. Control The component of the processor that commands the datapath according to the instructions of the program. It sets up dataflow directions on communication buses and selects ALU and memory functions. Control signals are generated by a control unit of one or more finite-state machines.

1.1.1.1 Bus Interconnection

A bus is a communication pathway connecting two or more devices (Figure 1.1). It is a set of parallel wires connecting the components of the computer. Buses are shared transmission mediums. They are shared by all the devices connected to the bus. If two devices transmit during the same time period, their signals will overlap and become garbled. This is called bus contention.

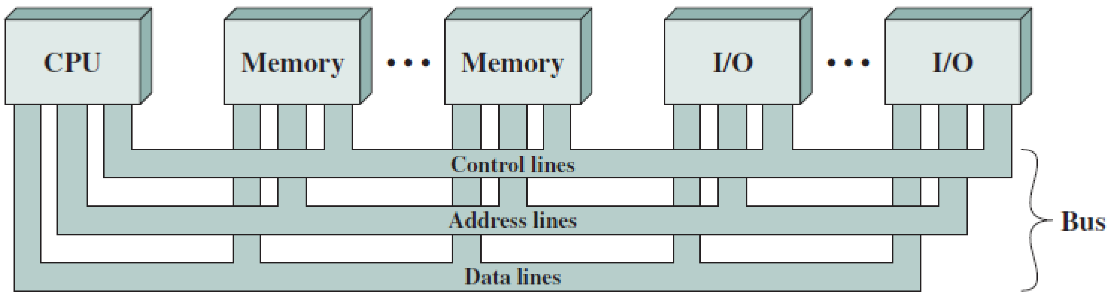


Figure 1.1: Bus

A bus is divided into three main parts: data bus, address bus, and control bus. Each part is a collection of lines (wires) that carry specific information at the same time. The number of lines is referred to as the width of the data bus. The width of the data bus is the number of bits that can be transmitted in parallel. It is a key factor in determining the performance of the computer system. The wider the data bus, the more data that can be transmitted in parallel, and the faster the computer system. The width of the data bus is measured in bits. The width of the data bus is equal to the word size.

The address bus is a set of parallel wires connecting the processor to memory. It carries the address of the memory location to be read from or written to. The width of the address bus determines the maximum number of memory locations that can be accessed. The higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port within the module.The address size is a key factor in determining the performance of the computer system too. The wider the address bus, the more memory that can be accessed, and the faster the computer system.

The control lines are used to control the access to and the use of the data and address lines. The control bus carries signals that control the operations of the computer. They transmit both command and timing information among system modules. Timing signals indicate the validity of data and address information. while command signals specify operations to be performed. The control bus is used to coordinate the activities of the processor, memory, and I/O devices.

To illustrate the role of buses in a computer system, consider the following example. When the processor wants to execute a simple instruction like a = b + c. The processor sends the address of the memory location of b to the memory through the address bus and a signal through the control bus to read the data from the memory. The memory reads the data from the memory location of b and sends it to the processor through the data bus. It then sends the address of the memory location of c to the memory through the address bus and a signal through the control bus to read the data from the memory. The memory reads the data from the memory location of c and sends it to the processor through the data bus. The processor adds the two data and stores the result in the memory location of a. For that, it sends the address of the memory location of a to the memory through the address bus and a signal through the control bus to write the data into the memory. The memory writes the result to the memory location of a.

1.1.1.2 Arithmetic and Logic Unit (ALU)

Arithmetic and Logic Unit (ALU) is a digital circuit that performs arithmetic and logical operations (Figure 1.2). It is the core of the processor. Operands for arithmetic and logic operations are presented to the ALU in registers. The ALU apply the proper operation and stores the results in a register. It also has input control lines to select the operation to be performed. Finally, if anything goes wrong, the ALU may also set flags as the result of an operation. For example, an overflow flag is set to 1 if the result of a computation exceeds the length of the register into which it is to be stored. The flag values are also stored in registers within the processor.

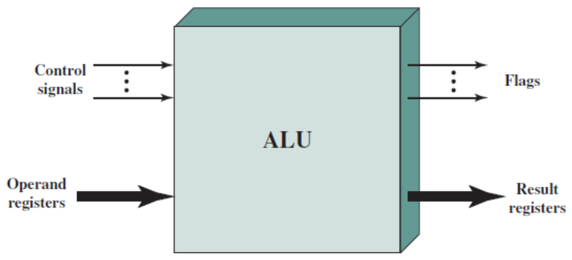


Figure 1.2: ALU

1.1.2 The Memory Hierarchy

The Memory Hierarchy is a pyramid of storage devices with different speeds and sizes (Figure 1.3). In general, the closer memory is to the CPU, the faster and more expensive it is. In contrast, the further memory is from the CPU, the slower and cheaper it is. The memory hierarchy is used to speed up the execution of programs by providing fast access to the most frequently used data. The memory hierarchy is composed of several levels: registers, cache, main memory, and secondary storage.

Registers are the fastest and smallest memory in the hierarchy. They are used to store the operands and results of the ALU. The registers are located inside the processor and typically provide about 1KB of storage and are accessed at the same speed as the CPU. The cache is the second level of the memory hierarchy. It is used to store the most frequently used data and instructions. The cache is located between the processor and the main memory. The main memory is the third level of the memory hierarchy. It is used to store the data and instructions that are not stored in the cache. The main memory is located outside the processor. The secondary storage is the fourth level of the memory hierarchy. It is used to store the data and instructions that are not stored in the main memory.

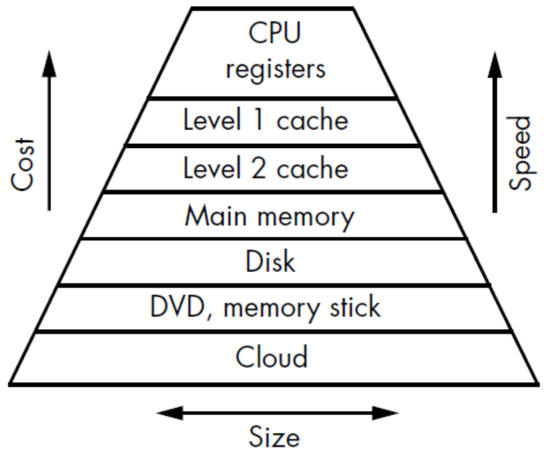


Figure 1.3: The Memory Hierarchy

1.1.2.1 Registers

Many registers are directly accessible by the programmer, while others are hidden. Some are used in the hardware that interfaces between the CPU and I/O devices. The organization of registers in the CPU is very specific to the particular CPU architecture, meaning that the number of registers, their size, and their use are different from one CPU to another. The registers that are directly accessible by the programmer are called general-purpose registers.

The most commonly used registers are:

1. **Program Counter** A program counter (PC) is a CPU register in the computer processor that has the address of the next instruction to be executed from memory. As each instruction gets fetched, the program counter increases its stored value by 1. It is a digital counter needed for faster execution of tasks as well as for tracking the current execution point.
2. **Instruction Register (IR)** is the part of a CPU's control unit that holds the instruction currently being executed or decoded. The instruction register specifically holds the instruction and provides it to the instruction decoder circuit.
3. **Memory Address Register (MAR)** is the CPU register that either stores the memory address from which data will be fetched from the CPU, or the address to which data will be sent and stored. It is a temporary storage component in the CPU that temporarily stores the address (location) of the data sent by the memory unit until the instruction for the particular data is executed.
4. **Memory Data Register (MDR)**, also known as memory buffer register (MBR), is the register that stores the data being transferred to and from the immediate access storage.
5. **General Purpose Register** General-purpose registers are used to store temporary data within the microprocessor. They can be used either by a programmer or by a user.

The registers in the CPU that are used for similar operations are grouped into a register file. We connect an 8-to-3 multiplexer to each corresponding bit of the eight registers. The inputs to the multiplexer, $r0_i\breve{}r7_i$, are the i^{th} bits from each of eight registers (Figure 1.4), $r0\breve{}r7$. For instance, a four-bit register would need four of these multiplexer output circuits. The same *RegSel* would be applied to all four multiplexers simultaneously to output all four bits of the same register.

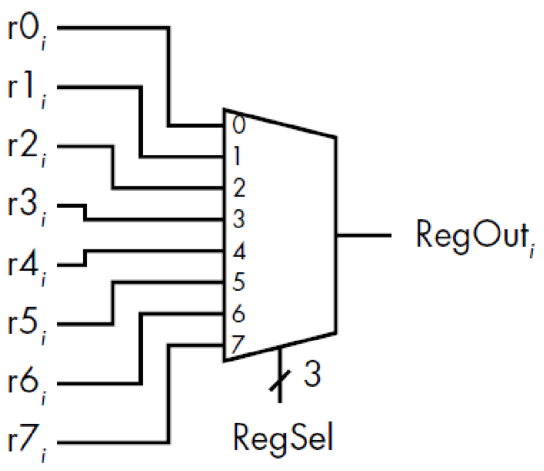


Figure 1.4: Register File

1.1.2.2 Cache Memory

Most modern computers include very fast **cache memory** between the main memory and the CPU. It provides a much faster location for the instructions and variables currently being processed by the program. Cache memory is organized in levels, with Level 1 being the closest to the CPU, and the smallest (Figure 1.5). The amount of memory copied into a cache at a time is called a line.

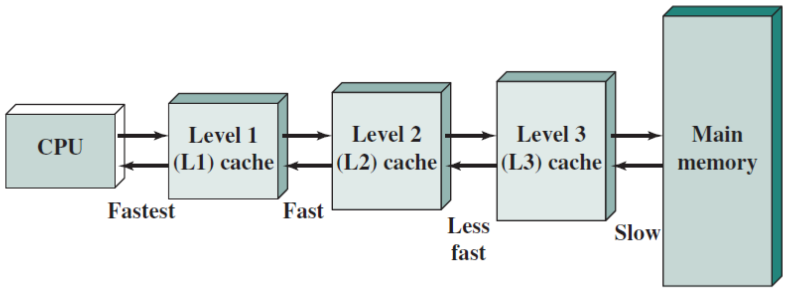


Figure 1.5: Cache Memory

Figure 1.6 depicts the steps taken to retrieve data from memory going through the different levels of cache memory.When a program needs to access an instruction or data item, the hardware first checks if it is located in the L1 cache. If it is, the hardware reads the data from the L1 cache and executes the instruction. If the data is not in the L1 cache, the hardware checks if it is located in the L2 cache. If it is, the hardware reads the data from the L2 cache and copies it to the L1 cache. If the data is not in the L2 cache, the hardware checks if it is located in the L3. If it is, the hardware reads the data from the L3 cache and copies it to the L2 cache and then to the L1 cache. If the data is not in the L3 cache, the hardware checks if it is located in the main memory. If it is, the hardware reads the data from the main memory and copies it to the L3 cache, then to the L2 cache, and finally to the L1 cache. If the data is not in the main memory, the hardware checks if it is located in the secondary storage. If it is, the hardware reads the data from the secondary storage and copies it to the main memory, then to the L3 cache, then to the L2 cache, and finally to the L1 cache. If the data is not in the secondary storage, the hardware generates an exception and stops the execution of the program.

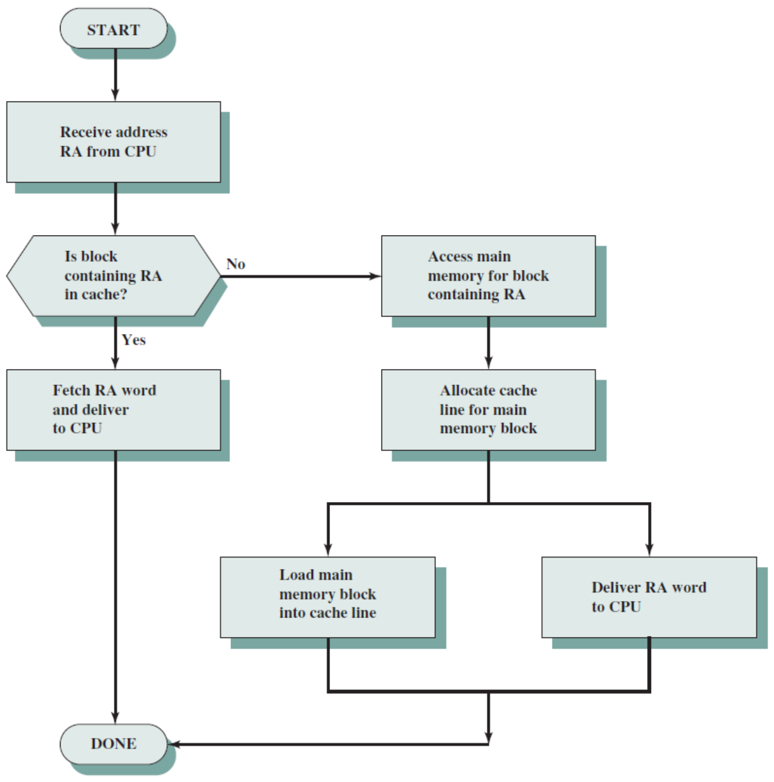

START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

Figure 1.6: Information retrieval from memory

1.1.2.3 Main Memory

Main memory is a large, slow, and inexpensive memory that stores the data and instructions that are not stored in the cache. Usually, the entire program can not be loaded into main memory. Instead, only the portion currently being worked on is loaded by the operating system from mass storage into main memory.

When a needed instruction or data item is loaded into main memory, the computer loads the whole block of instructions or data that includes the needed item into memory. Chances are good that the nearby parts of the program (instructions or data) will be needed soon. Since they're already in main memory, the operating system doesn't need to access the mass storage device again, thus speeding up program execution.

There are main two technologies used to implement the main memory: Dynamic Random Access Memory (DRAM) and Static Random Access Memory (SRAM). DRAM is a type of memory that stores each bit of data in a separate capacitor within an integrated circuit. The capacitor can hold a charge, which represents a 1, or no charge, which represents a 0. The charge in the capacitor leaks away over time, so the data must be refreshed periodically. DRAM is slower and less expensive than SRAM. SRAM is a type of memory that stores each bit of data in a flip-flop within an integrated circuit. The flip-flop can hold a charge, which represents a 1, or no charge, which represents a 0. The charge in the flip-flop does not leak away over time, so the data does not need to be refreshed. SRAM is faster and more expensive than DRAM.

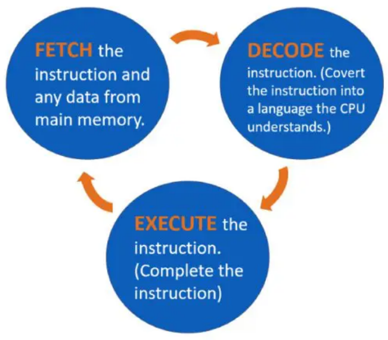
1.1.2.4 Mass Storage

Mass storage is a large, slow, and inexpensive memory that stores the data and instructions that are not stored in the main memory. Their contents are nonvolatile. Mass storage is used to store the entire program and the data that the program operates on. The data and instructions are loaded from mass storage into main memory by the operating system. This includes hard disks, solid-state drives, memory sticks, optical disks, etc.

Figure 1.7: Mass Storage

1.2 Instruction Execution Cycle

The processor executes instructions in a sequence. The instruction cycle is the basic operation of the processor. It is composed of three main steps (Figure 1.8). The first step is to fetch the instruction. Where the process retrieves the instruction from memory. The second step is to decode the instruction. Where the process determines the operation specified by the instruction. The third step is to execute the instruction. Where the process performs the operation specified by the instruction. The processor executes the instruction cycle repeatedly until the program is finished.


FETCH the instruction and any data from main memory.
DECODE the instruction. (Covert the instruction into a language the CPU understands.)
EXECUTE the instruction. (Complete the instruction)
Figure 1.8: Instruction Execution Cycle

1.3 Seven great ideas in computer architecture

Computer designers used some fundamental ideas to improve the performance of computer systems. These ideas are the foundation of computer architecture. The seven great ideas are:

Use Abstraction to Simplify Design: The basic idea is to hide the complexity of the underlying hardware by providing the programmer with a clean and simple interface. This is the essence of abstraction. Make the Common Case Fast: The basic idea is to design the hardware to optimize the execution of the most common instructions and to provide hardware support for the most common data types. This is the essence of optimization. The goal is to make the common case fast and the rare case correct. Performance via Parallelism: performance can be improved by doing more than one thing at a time. Computer architects have developed a number of techniques to exploit parallelism in both hardware and software.

Performance via Pipelining: A particular pattern of parallelism. The basic idea is to divide the execution of an instruction into a series of smaller steps, each of which can be performed concurrently with the others.

Performance via Prediction: ”It can be better to ask for forgiveness than to ask for permission”. The basic idea is to predict the outcome of an operation and to proceed as if the prediction is correct. If the prediction turns out to be wrong, the processor must undo the effects of the incorrect predictions. Prediction may seem like a strange way to improve performance, but it is a powerful technique that is used in many aspects of computer architecture.

Hierarchy of Memories: The basic idea is to use a hierarchy of memories to store programs and data. The fastest, smallest, and most expensive memory is located closest to the processor. Successively larger, slower, and less expensive memories are located further away. The goal is to provide the processor with the illusion of a large, fast, and inexpensive memory. The memory hierarchy is a fundamental aspect of computer architecture. It is used to bridge the gap between the high-speed processor and the slower, cheaper memory systems.

Dependability via Redundancy: we make systems dependable by including redundant components that can take over when a failure occurs and to help detect failures. The basic idea is to use redundancy to detect and correct errors.

1.4 How Processors Are Made

The chip manufacture begins with silicon, a substance found in sand. Because silicon does not conduct electricity well, it is called a semiconductor. With a special chemical process, it is possible to add materials to silicon that allow tiny areas to transform into one of three devices: Excellent conductors of electricity (using either microscopic copper or aluminum wire); Excellent insulators from electricity (like plastic sheathing or glass); Areas that can conduct or insulate under specific conditions (transistors) The transistors are the most important part of the chip. They are the tiny switches that control the flow of electricity. A chip can have billions of transistors. The transistors are connected by microscopic wires that carry the electrical signals. The chip is made by a process called photolithography. The process uses light to transfer the pattern of the chip design onto a silicon wafer. The result is a Very Large Scale Integration (VLSI) circuit. A VLSI circuit is billions of combinations of conductors, insulators, and switches manufactured in a single small package.

Silicon is first melted and then cooled into a solid form. We call this solid silicon crystal ingot (Figure 1.9). Silicon crystal ingot is a rod composed of a silicon crystal that is between 8 and 12 inches in diameter and about 12 to 24 inches long. The ingot is then sliced into thin wafers. Wafer is a slice from a silicon ingot no more than 0.1 inches thick used to create chips. After that a patterns of chemicals are placed on each wafer, creating the transistors, conductors, and insulators. Then we tesrt the wafer to make sure it does not have any defects. Defect is a microscopic flaw in a wafer or in patterning steps that can result in the failure of the die containing that defect. The wafer is then cut into individual dies. Die is the individual rectangular sections that are cut from a wafer, more informally known as chips. We then test the dies to make sure they are working properly. The good dies are connected to the input/output pins of a package. This process is called Bonding. The package is then sealed to protect the dies from the environment. This process is called Packaging. The packaged chip is then tested to make sure it is working properly. Finally, the chip is shipped to the customer.

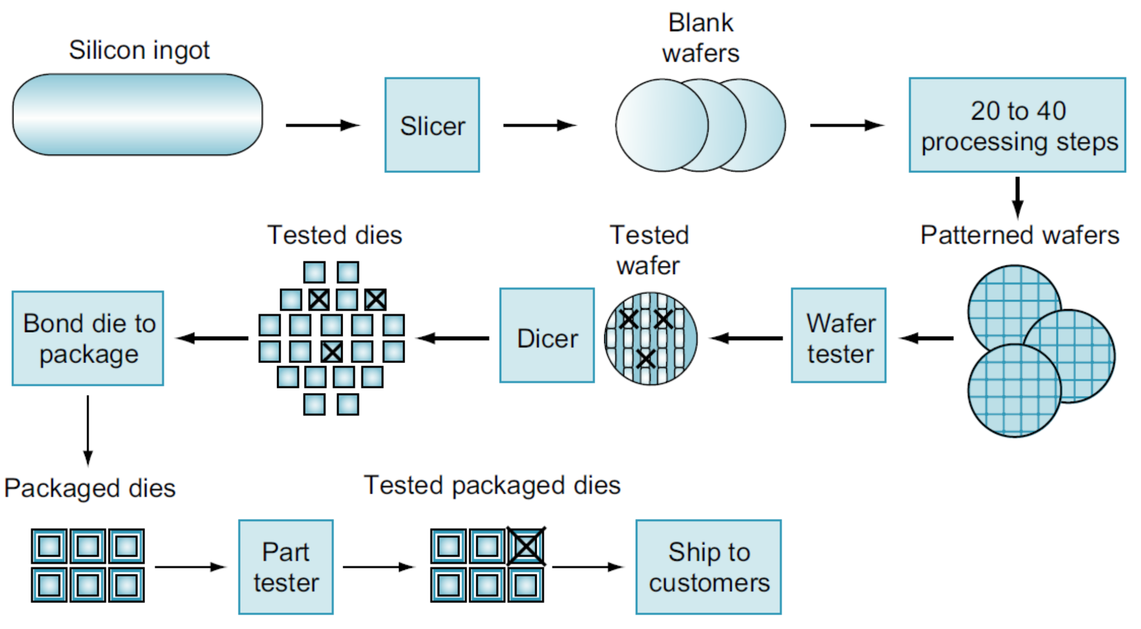


Figure 1.9: Making Chips

Here you can see that we can't use all the area of the wafer because of its circular shape and the rectangular shape of the dies. The area of the wafer that can be used to create dies is called the die area. Another factor that affects the number of dies that can be created on a wafer is the defect. The percentage of good dies from the total number of dies on the wafer is called the yield. Yield The percentage of good dies from the total number of dies on the wafer.

The cost of an integrated circuit can be expressed in three simple equations:

$$Cost\ per\ die = \frac{Cost\ per\ wafer}{Dies\ per\ wafer\ \times\ yield} \tag{1.1}$$

$$Dies\ per\ wafer \approx \frac{Wafer\ area}{Die\ area}, \tag{1.2}$$

since it does not subtract the area near the border of the round wafer that cannot accommodate the rectangular dies.

$$Yeld = \frac{1}{(1+(Defects\ per\ area \times Die\ area))^N}, \tag{1.3}$$

based on empirical observations of yields at integrated circuit factories, with the exponent related to the number of critical processing steps.

Figure 1.10 shows a 300mm by 10nm wafer contains 10^{th} Gen $IntelI7Core^{TM}$ processors, code-named **"Ice Lake"**. The number of dies on this wafer at 100% yield is 506.

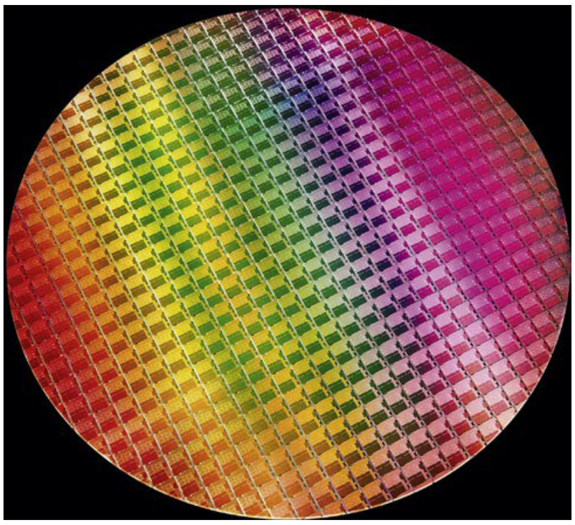


Figure 1.10: Ice Lake Wafer

1.5 Measuring The Performance

1.5.1 What do we mean by performance?

What do we mean by performance? The performance of a computer system is the amount of useful work accomplished by the in a given amount of time. In other words, the performance of a computer system is measured in terms of the time it takes to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, etc. The time it takes to complete a task is called the response time Also called execution time, wall clock time, or elapsed time. The execution time is measured in seconds.

$$Performance = \frac{1}{Execution\ time} \tag{1.4}$$

We can then compare the performance of two computers as follows:

$$Performance_X > Performance_Y$$

$$\frac{1}{Execution\ time_X} > \frac{1}{Execution\ time_Y} \tag{1.5}$$

$$Execution\ time_Y > Execution\ time_X$$

Another way to measure the performance of a computer system is to measure the number of tasks that can be completed in a given amount of time. This is called throughput or bandwidth. Throughput is the number of tasks completed per unit of time. The throughput is measured in tasks per second. A processor executes several programs simultaneously. Therefore, the system may optimize throughput rather than the elapsed time for one program.

For example, Personal mobile devices are more focused on response time than servers, which are more focused on throughput.

1.5.2 Relative Performance

If X is n times as fast as Y, then the execution time on Y is n times as long as it is on X:

$$\frac{Performance_X}{Performance_Y} = \frac{Execution_Y}{Execution_X} \tag{1.6}$$

If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

$$\begin{aligned} \frac{Performance_A}{Performance_B} = \frac{Execution_B}{Execution_A} = n \\ \frac{15}{10} = 1.5 \end{aligned} \tag{1.7}$$

A is 1.5 times as fast as B, or B is 1.5 times slower than A.

1.5.3 Measuring Performance

CPU execution time also called CPU time is the actual time the CPU spends computing for a specific task and does not include time spent waiting for I/O or running other programs. It is the most important factor in determining the performance of a computer system. The CPU execution time is the product of the number of instructions executed and the average time it takes to execute each instruction. The CPU execution time is measured in seconds.

User CPU time is the CPU time spent in a program itself. It does not include time spent in the operating system or other programs. System CPU time is the CPU time spent in the operating system performing tasks on behalf of the program. It does not include time spent in the program itself, such as waiting for I/O. The response time is the sum of the user CPU time and the system CPU time.

Another important concept is the clock cycle, also called tick, clock tick, clock period, clock, or cycle. It is the time for one clock period, usually of the processor clock, which runs at a constant rate. The clock cycle is measured in seconds. The clock rate is the inverse of the clock cycle. It is the number of clock cycles per second and is measured in Hertz. The clock rate is also called the clock frequency. The clock rate is the speed of the processor.

We calculate the CPU execution time for a program as follows:

$$CPU\ execution\ time = CPU\ clock\ cycles \times Clock\ cycle\ time \tag{1.8}$$

, Or

$$CPU\ execution\ time = \frac{CPU\ clock\ cycles}{Clock\ rate} \tag{1.9}$$

We improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle.

1.5.4 Instruction Performance

CPU clock cycles are different from one instruction to another. The number of clock cycles required to execute an instruction is called the clock cycles per instruction (CPI). The average number of clock cycles per instruction is called the average CPI. The average CPI is measured in clock cycles per instruction.

Similarly, the number of instructions executed is different from one program to another. The number of instructions executed is called the instruction count. Thus, the CPU time is the product of the number of instructions executed, the average CPI, and the clock cycle time.

$$CPU\ time = IC \times CPI \times Clock\ Cycle\ Time \tag{1.10}$$

Or,

$$CPU\ time = \frac{IC \times CPI}{Clock\ rate} \tag{1.11}$$

Now that we have a clear idea on how to measure the performance of a computer, we can understand the effect of the algorithm design, programming language, compiler, and instruction set architecture on the performance of a computer system (Table 1.1). First of all, the algorithm design will affect the number of instructions executed and the average CPI. The better the algorithm design, the fewer the number of instructions executed and the average CPI. For example, if the algorithm uses more divides, it will tend to have a higher CPI. Similarly, The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions. The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in varied ways. Finally, the instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

Hardware/ software component	Affects what?	How?
Algorithm	IC, CPI	**IC**: # program instructions -> # processor instructions **CPI**: Complex instruction -> High CPI
Language	IC, CPI	Abstraction level: High level Vs Low level languages
Compiler	IC, CPI	Translation efficiency of different compilers
Instruction Set Architecture	IC, CPI, Clock rate	See above

Table 1.1: The affect of Hardware/ software component on performance

1.6 CISC Vs RISC

There are two main families of microprocessor architecture, namely: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). The CISC architecture is characterized by a large number of complex instructions, which can take multiple clock cycles to execute. The RISC architecture, on the other hand, is characterized by a smaller number of simple instructions, which can be executed in a single clock cycle. The CISC architecture was developed in the 1970s and was designed to make it easier for programmers to write code. The idea was that by providing a large number of complex instructions, the programmer would have to write less code to perform a given task. However, the downside of this approach is that the processor has to work harder to decode and execute the complex instructions, which can result in slower performance. The RISC architecture was developed in the 1980s as a response to the limitations of the CISC architecture. The idea was to simplify the instruction set to make it easier for the processor to decode and execute instructions. This approach has the advantage of faster performance, as the processor can execute instructions more quickly. However, the downside is that the programmer has to write more code to perform a given task.

We say that CISC emphasis is on hardware while RISC emphasis is on software. This is because the CISC architecture make the job of the compiler designer easier since the translation of the high-level language to a complex machine language is easier. While letting the hardware designer to design a complex hardware to execute the complex instructions. On the other hand, the RISC architecture make the job of the hardware designer easier since the hardware instruction set is simple. While letting the compiler designer to design a complex compiler to translate the high-level language to the simple machine language.

The instruction formats of CISC and RISC architectures are different. The CISC architecture uses variable length instructions, which can be of different lengths depending on the instruction. The RISC architecture, on the other hand, uses fixed length instructions, which are all the same length. The benefit of fixed length instructions is that they are easier for the processor to decode, as the processor can read the instruction in a single clock cycle, without having to wait for the end of the instruction to determine its length. With CISC, the processor has to read the instruction and then determine its length before it can start decoding it, which can result in slower performance.

Commercial processors like Intel and AMD use the CISC architecture, while academic and research processors like RISC-V use the RISC architecture. The trend in nowadays is to use the RISC architecture because of its faster performance and lower power consumption. However, the CISC architecture is still widely used in commercial processors because of its ease of use for programmers.

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instruction of the same size with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of micro-programming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

Table 1.2: Comparison between CISC and RISC

1.7 Exercises