Layered computer design follows the top-down approach to design and also to improve computers. A computer is designed layer by layer, starting with the top conceptual layer and ending with the bottom layer, the complete design. A layer of a computer consists of operations, operands and components. A layer is implemented by the operations, operands and components of the layer below. There are many possibilities to implement a layer. To decide, design goals are used: speed, cost, size, weight, power consumption, reliability, expendability, flexibility and compatibility.

Applications to Transistors

<table>
<thead>
<tr>
<th>Application Level</th>
<th>Computational Method Level</th>
<th>Architecture Level (Machine Language Level)</th>
<th>Microarchitecture Level (Organisation Level) (Register Transfer Level, RTL)</th>
<th>Logic Level</th>
<th>Transistor Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Scientist</td>
<td>Abstract machine</td>
<td>Add, subtract, multiply, load, store, jump, branch,...</td>
<td>- Fetch instruction, increment PC, calculate effective address,...</td>
<td>- AND, OR, NOT, clocked store on a flip-flop,...</td>
<td>Switch on and off</td>
</tr>
<tr>
<td>Algorithm designer</td>
<td>Abstract machine</td>
<td>- 2’s complement integers, FP numbers, vectors,...</td>
<td>- 2’s complement numbers, memory addresses,...</td>
<td>- Gates (AND, OR, NOT) and flip-flops: digital circuits</td>
<td>- Voltage levels</td>
</tr>
<tr>
<td>Operating System Level</td>
<td>Abstract machine</td>
<td>- Processes, tasks, threads, memory, I/O,...</td>
<td>- Abstract machine</td>
<td>- Abstract machine</td>
<td>- Abstract machine</td>
</tr>
<tr>
<td>Computer architect &amp; Compiler, linker, loader</td>
<td>- Simulating a plane, word processing, controlling an elevator,...</td>
<td>- Add, subtract, multiply, sort, Call, Return, Search,...</td>
<td>- Abstract machine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm Level</td>
<td>Abstract machine</td>
<td>- Processes, tasks, threads, memory, I/O,...</td>
<td>- Abstract machine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High-level Language Level</td>
<td>Abstract machine</td>
<td>- Create processes, allocate memory, switch tasks,...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td>Hardware</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Computer architecture courses cover application, architecture, organization, logic and transistor layers. But, four other layers between the application and architecture layers need to be studied as well for a comprehensive design as the figure above shows. In the figure above, who/what implements a layer is indicated on the left side. Some layers are shown with three text lines on the right side. The first text line indicates operations of the layer, the second text line indicates operands of the layer and the third text line indicates components of the layer. Finally, it must be noted that ultimately a computer computes by means of its transistors turning (switching) on and off: The transistor layer is the complete design! All nine layers shown in the figure above are summarized below. However, before we discuss the layers, we will discuss computer fundamentals, then popular computer classifications and then conclude that the nine layers above give a better view of computers.
The fundamentals: A computer processes digital information. In order to do that it runs (executes) a machine language program. As an example, when we buy software, such as the Microsoft Word, we buy the machine language program of the Word software. A machine language program manipulates data. A machine language program consists of machine language instructions. A machine language instruction is a simple command that can be immediately understood by the hardware. It commands the computer to perform a simple operation such as add, subtract, and, shift left, etc. Thus, it can be directly run by the computer (hardware).

Machine language instructions and data are in terms of 1s and 0s and are stored in the memory. It is not possible to distinguish whether a part of the memory has an instruction or a data element by just looking at it. This is a unique property of today’s computers and so are called stored-program computers. Data and programs are input from input/output (I/O) devices into the computer memory and result data are output to the I/O devices.

Computers are classified with respect to their size, speed and cost as supercomputers, servers, desktop computers and embedded computers. Supercomputers are the fastest computers, costing millions of dollars and very large. They are used for scientific applications, such as airplane design, weather forecasting, molecular simulations. Government agencies and large corporations can afford them. CS6134 will concentrate on supercomputer class computers. Servers are large computers that allow multiple users to run general-purpose applications. Companies and universities are typical customers. Desktop computers are single-user machines, intended to run a small numbers applications ranging from email to word processing. Embedded computers are very small and control a system they are embedded in. They typically have one application to run which is the control of the system they are in.

Another classification is hardware vs. software. Hardware is the collection of physical components, such as chips, wires, PCBs, connectors, I/O devices, etc. that form a computer. Software is the collection of programs on a computer. Software and Hardware are equivalent in that any operation performed by the hardware can be built into software and any operation performed by software can be also directly realized by hardware. Therefore, we have the hardware/software trade-off. This equivalence is under the assumption that there is a basic set of operations implemented in hardware. Decisions on what to include in hardware and software are based on the required speed, cost, reliability, frequency of expected changes, etc.

There are two types of software today: Application and systems. The meaning of the two changes computer to computer. Since we concentrate on large computers, supercomputers, in CS6143, we define application programs as those run by ordinary users, such as email, word processing, spreadsheet, simulation programs, etc. Systems programs are used to control the hardware to make the computer easy to use, secure and more efficient. Systems software include the operating systems, language translators (compilers, assemblers), linkers, loaders, libraries. They are used by systems people who have special privileges (access rights) to use the computer. This distinction is enforced by today’s computers in the form of hardware control states: user and system states. Application programs are run in the user state and if they try to run system software in this mode an interrupt (exception) is generated. The program is terminated. System programs are run in the system state.

Even though software is in machine language, today it is often developed by first writing in a high-level language or an application-oriented language or in assembly language. High-level languages include C++, Java, C, Fortran,
Cobol, Python, PHP, etc. **Application-oriented languages** contain constructs and keywords to develop a program for a specific class of applications, such as simulating a computer network. **Assembly languages** are related to the architecture of the processor they are targeted for. That is, for a computer with an Intel Pentium processor, one would develop an assembly language program in the Intel assembly language. If the processor is an IBM Power processor, one would write an IBM Power assembly language program.

Since the computer can run only machine language programs, one needs to translate the above programs to machine language programs. To translate from a high-level language program to the machine language program, **compilers** are used: C++ compiler, Java compiler, etc. To translate from an assembly language program to the machine language program, **assemblers** are used: Intel assembler, IBM assembler, etc. To translate from an application-oriented language program to the machine language program, typically preprocessing programs are used to convert to an intermediate form in a high-level language and then they are compiled to the machine language program. Among the three types of languages, application-oriented languages are the highest level, meaning very easy to write and assembly languages are the lowest, meaning hardest to write. Although it is easier to develop application-oriented language programs, their corresponding machine code may not be efficient since preprocessors and compilers may not be sophisticated enough to generate an efficient machine code. On the other hand, developing a large assembly language program may not be practical due to the complexity of the language. The common practice today is that for embedded applications assembly and C programs are developed since embedded programs are not large. For all others high-level and application-oriented languages are used.

Another computer classification is architecture vs. organization (microarchitecture). The **architecture** is the set of resources visible to the machine language programmer: Registers, the memory, data representations, addressing modes, instructions formats, control states, I/O controllers, interrupts, etc. Although often the architecture is thought to be equivalent to the machine language set of a computer, it is more than that. Still, a major portion of the architecture coverage is devoted the machine language set. A related issue in the past was whether the machine language set should be complex (complex instruction set computer, **CISC**) or simple (reduced instruction set computer, **RISC**). The debate took place in the 1980s and first half of the 1990s. It was resolved as the RISC the winner since it allows more efficient pipelining, leads to simpler hardware and easier increase of the clock frequency. The Intel and Motorola machine language sets are CISC. The Sun is RISC. Why and how the Intel CISC architecture has kept its dominance will be clear later in the semester. But, simply, this has been possible by designing an Intel CPU that converts each Intel CISC instruction to up to three RISC instructions on the fly.

Studying the architecture implies working on machine language programs. But, this is not practical when we design a computer since machine language programs have 1s and 0s. Therefore, in CS6143, we will work on mnemonic machine language programs. They are easier to write and in one-to-one correspondence with machine language programs. That is, if one has a mnemonic machine language program, it is very straightforward to obtain the corresponding machine language program. Note that there is often no one-to-one correspondence between assembly language programs and machine language programs.

The **organization** is the set of resources that realizes the architecture which include the CPU, the memory and I/O controllers. These are digital systems with registers, buses, ALUs, sequencers, etc. The CPU is responsible for running machine language programs: It runs machine language instructions. Running a machine language instruction is performing a simple operation (command) on data. The memory keeps the programs and data, leading to the stored-program concept of today’s computers. I/O controllers interface the I/O devices to the memory and CPU. An I/O controller can control one or more I/O devices. Often the number of I/O devices connected to an I/O controller depends on the speed of I/O devices. A high speed I/O device can be controlled by a single I/O controller while a few slow speed I/O devices can be controlled by a single I/O controller. The stored-program concept and the generic view of a computer organization with at least three digital systems (the CPU, memory and I/O controller) are often attributed to mathematician **John Von Neumann**. However, there is considerable debate on that.

A microprocessor contains at the least the CPU which was the case in the 1970s and early 1980s. Today they include cache memories, bus interfaces, memory management units. High-performance microprocessors from Intel, AMD,
Sun, IBM have these functional units. Some other chips in the market today contain memory and even I/O controllers. These are used for embedded applications and called microcontrollers, not microprocessors. The reason why the memory and I/O controllers are added is that embedded computers are often required to occupy a small space in the system they are housed in. To reduce the chip count, hence the physical space, this approach is needed.

As the above discussion indicates looking at a computer from different points of view can be at least distractive, if not confusing for beginners of computer design: hardware vs. software, different programming languages, operating systems, compilers, assemblers, architecture vs. organization, etc. That is why the concept of computer layers is used to give a comprehensive view of computers at different complexities or abstraction. Abstraction allows reducing the number of details of a layer with a simpler view. In the computer layers figure on the first page, a layer is abstracted by the layer just above it.

1. Computer Layers

The Application, Computational Method, High-Level Language, Operating Systems and Architecture layers constitute the software layers. The Architecture, Microarchitecture, Logic and Transistor layers constitute the hardware layers. Each layer, except the Application layer, implements the layer above, following the concept of abstraction. Clearly, the Architecture is the hardware/software interface. A computer architect needs to handle both hardware and software and keep track of advances in both.

Application Layer:
This layer indicates the set of applications intended for the computer! Ideally, all applications can be run on a computer. However, in practice the computer is designed to “efficiently” run a subset of them. For example, a computer runs scientific applications, a different computer runs business applications, etc. Our MIPS computer will target scientific applications. Some of the applications mentioned in the textbook are benchmark suites such as Linpack, Livermore Loops, Whetstone, Dhrystone, SPEC CPU 2000, SPEC CPU2006, SPECWeb, SPECFS, TPC-C and EDN EEMBC (Embedded Microprocessor Benchmark Consortium benchmark of five classes of applications).

Computational Methods Layer:
This layer is highly theoretical and abstract. The computational method (i) determines characteristics of items (data and other) and work (operations), ii) describes how operations initiate each other during execution, i.e. which operation is followed by which or determining the order of performing operations, and (iii) implicitly determines the amount of parallelism among the operations. Three types of computational methods are frequently covered in the discussion of this topic: control flow, data flow and demand driven.

Today’s computers use the control flow computational method where the order of operations is specified by the order of instructions in the program. The order implies the execution order and so next instruction to perform is the one that follows the current instruction in the program. If one wants to change the order of execution, explicit control instructions (branch, jump, etc.) must be used, hence the name control flow. This explicit sequence of operations obscures parallelism. Thus, the control-flow is inherently sequential, hindering parallelism and higher speeds. This is the reason why today’s supercomputers are very expensive as they need complex compilers, operating systems, hardware and highly trained parallel algorithm designers and programmers to extract parallelism from sequential pro-
grams. In **data flow**, an operation starts its execution when all of its operands are available. Since the operand availability determines the order of operations, this method is also called data driven. Many operations can have their operands ready at the same and so they can start execution at the same. Thus, data flow does not hinder parallelism. In fact, the parallelism is explicit to the fullest extend. In **demand driven**, an operation starts when its result is demanded. Many operation results can be demanded at the same and so they can all start execution in parallel. Demand driven computation also has parallelism explicit. Overall, data-flow and demand driven methods are inherently parallel. However, to implement them in full scale today is not efficient given the current technology.

**Algorithm Layer :**
The algorithm for an application specifies major steps to generate the output. The algorithm follows the computational method chosen. An algorithm is abstract and short. It is independent of high-level languages. Today, for a single-processor (uniprocessor) computer such as the MIPS processor, we write a sequential algorithm in the control-flow method. However, if we have a computer with multiple processors (cores), we write a parallel algorithm but still use the control flow method. Below, we discuss three common scientific applications and the **sequential** algorithms:

<table>
<thead>
<tr>
<th>Application: <strong>Dot Product</strong></th>
</tr>
</thead>
</table>
| \[
\text{dot} = \sum_{i=1}^{n} A[i] \times B[i] \]
| \[
A, B \text{ are vectors with } n \text{ elements}
\]

**Algorithm:**

- \[
\text{dot} = 0 \\
\text{for } (1 \leq i \leq n) \text{ do} \\
\text{dot} = \text{dot} + (A[i] \times B[i])
\]

- **Sequential time complexity:** \(O(n)\), linear
- **We will develop its parallel version!**

<table>
<thead>
<tr>
<th>Application: <strong>SAXPY/DAXPY, a step in Gaussian elimination to solve linear equations</strong></th>
</tr>
</thead>
</table>
| \[
Y = a \times X + Y \implies Y[i] = a \times X[i] + Y[i]
\]
| \[
X, Y \text{ are vectors with } n \text{ elements } \& a \text{ is a scalar}
\]

**Algorithm:**

- \[
\text{for } (1 \leq i \leq n) \text{ do} \\
Y[i] = a \times X[i] + Y[i]
\]

- **Sequential time complexity:** \(O(n)\), linear
- **We will develop its parallel version!**

<table>
<thead>
<tr>
<th>Application: <strong>Matrix Multiply</strong></th>
</tr>
</thead>
</table>
| \[
A = B \times C \implies A \text{ is an } m \times p \text{ matrix} \\
B \text{ is an } m \times n \text{ matrix} \\
C \text{ is an } n \times p \text{ matrix}
\]

**Algorithm:**

- \[
\text{for } (1 \leq i \leq m) \text{ do} \\
\text{for } (1 \leq j \leq p) \text{ do} \\
A[i,j] = 0 \\
\text{for } (1 \leq k \leq n) \text{ do} \\
\]

- **Sequential time complexity:** \(O(mp)\), \(O(n^3)\), polynomial
- **We will develop its parallel version!**
High-Level Language Layer:
The algorithm developed for an application is coded in a high-level language, such as Fortran, C, C++, Java, etc. Fortran is still the choice of scientific computing, while C is gaining ground. Note that for an algorithm there are different programs possible as each can be in a different high-level language.

Operating Systems Layer:
This layer interfaces with hardware. That is, it hides hardware details from the programmer and provides, security, stability and fairness in the computing system. Thus, this layer adds more code to run on the behalf of the application. The layer also handles interrupts and input/output operations.

Architecture Layer:
The architecture layer is the hardware/software interface. Its elements include the machine language instruction set, register sets, the memory and Input/Output structures among others. CS6143 discusses this level considerably, and so its description here is kept short.

Microarchitecture Layer:
This layer consists of digital systems. A computer which is a digital system consists of at least three smaller digital systems: the processor (CPU), the memory and Input/Output controller. A digital system consists of registers, buses, ALUs, sequencers, etc. Other names used for this layer are organization and register transfer level (RTL). The microarchitecture layer is also discussed in depth in CS6143. Handouts will be distributed during the semester to ensure students understand fundamental microarchitecture concepts.

Logic Layer:
This layer consists of digital circuits. Digital circuits form digital systems of the microarchitecture level. Digital circuits use gates and flip-flops. Most common gates used are AND, OR and NOT gates. A flip-flop stores a single bit. To store the bit (1 or 0), a clock signal is used. Most common flip-flops used are D and JK flip-flops. Note that a flip-flop is not a memory. The memory chip design is different from the flip-flop design. There are two types of digital circuits. A combinational circuit contains gates. Combinational circuits cannot store information. Sequential circuits contain gates and flip-flops. They store past inputs: The output now is a function of inputs now and past inputs.

Transistor Layer:
This layer consists of digital electronic circuits. Digital electronic circuits are used to build digital circuits. That is, digital electronic circuits implement gates (also flip-flops). Digital electronic circuits consist of transistors, resistors, capacitors, diodes, etc. Transistors are the main component and so this level is often called the transistor level. Transistors in these circuits are used as on-off switches. The switches are turned on and off by control inputs. The figure below shows on-off switches and how these switches are used to implement an AND gate as an example.

A switch is a “device” with two conditions:

<table>
<thead>
<tr>
<th>Switch</th>
<th>Control Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>0</td>
</tr>
<tr>
<td>Closed</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Open** when the control input is 0
- **Closed** when the control input is 1
The figure below gives an example of a gate implementation where the gate is a 2-input **TTL NAND gate**. The implementation of the NAND gate by transistors, resistors, etc. is shown next to the gate. TTL is a chip technology and is described below. Today, digital electronic circuits, i.e. circuits with transistors, resistors, capacitors, etc., are on chips. That is, gates and flip-flops are on chips.

We use **semiconductor** substances to implement transistors. That is, today’s chips are semiconductor chips. Silicon and Gallium Arsenide are examples of semiconductor substances. Each substance has its own speed, cost, power consumption figures. Today, the most common substance is Silicon which is found in sea sand. This is why Silicon chip prices are so **low**.

Chip design is constrained by design goals: speed, cost, power consumption, size, weight, reliability, etc. Before the design is started, we determine these constraints and then design the product. We try not to exceed the constraints, by using the right number of gates and flip-flops and right digital electronic implementations. However, it is not easy to satisfy them as they conflict with each other. For example, the higher the speed, the higher the cost and power consumption. Hence, a study of spectrum of choices from semiconductor substances to chip densities is needed. The figure below shows the spectrum of substances and their relative speed for today’s digital electronic circuits.
Unipolar/bipolar transistors and other electronic components (resistors, capacitors, diodes,...) are used to implement transistor circuits, such as CMOS, TTL, ECL and BiCMOS. By using a transistor circuit, we implement a single gate. For example, a CMOS AND gate, a TTL AND gate, etc. The reason for using resistors, capacitors, diodes besides transistors for a gate is first for the correct usage of transistors and second to maintain the signal integrity, hence operational stability of the gate.

The number of electronic components on a chip depends on the intended functionality: the more functionality, the more components. A widely used classification of integration of components on chips is given on Table 1 below. The earliest chips from the 1960s were SSI chips and some of them are still used today. The current state of the art microprocessors have more than 200 million components. The integration level for these high-density chips is beyond ULSI but no new name is agreed upon it yet.

### Table 1: Chip densities for various scales of integration

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Scale Integration (SSI)</td>
<td>&lt; 10 gates</td>
<td>&lt; 64 components</td>
</tr>
<tr>
<td>Medium Scale Integration (MSI)</td>
<td>&lt; 100 gates</td>
<td>&lt; 2K components</td>
</tr>
<tr>
<td>Large Scale Integration (LSI)</td>
<td>&lt; 10,000 gates</td>
<td>&lt; 64K components</td>
</tr>
<tr>
<td>Very Large Scale Integration (VLSI)</td>
<td>&lt; 100,000 gates</td>
<td>&lt; 2M components</td>
</tr>
<tr>
<td>Ultra Large Scale Integration (ULSI)</td>
<td>&gt; 100,000 gates</td>
<td>&gt; 2M components</td>
</tr>
</tbody>
</table>

Silicon is the most commonly used substance and used by high-speed microprocessors and high-density memory chips. Silicon is expected to be around 10 to 15 years into the future at least. Table 2 below presents the state of the silicon technology. Silicon transistor circuits (CMOS, TTL,...) have different speed, cost, power consumption figures. A brief description of TTL and CMOS circuits is given below. TTL circuits are used for high-speed, low-cost applications while CMOS is for high-density chips, such as microprocessors and memories (DRAM, SRAM). CMOS circuits are also used for portable applications that require low-power consumption (space, embedded applications).

### Table 2: The state of the silicon technology

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Silicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Densest chip transistor circuit</td>
<td>CMOS</td>
</tr>
<tr>
<td>Transistors/chip (density)</td>
<td>4,000,000,000</td>
</tr>
<tr>
<td>Gate delay</td>
<td>50 - 500 ps</td>
</tr>
<tr>
<td>Process</td>
<td>22 nanometer</td>
</tr>
</tbody>
</table>

Table 3 compares three most commonly used transistor circuits. CMOS is the preferred transistor circuit to implement microprocessors and high-density memory chips. This is because CMOS circuits consume the least amount of power among the three. TTL is the most widely available and cheapest one, while ECL is the fastest one. Finally, we list a number of properties of TTL and CMOS technologies below.
2. The Big Picture: Transistors to Computers

Today, digital electronic circuits (transistor circuits) are on chips. That is, those transistors, resistors, capacitors, etc. are on chips. Chips are on printed circuit boards (PCBs) also known as cards. A PCB can contain tens of chips. The main PCB of a computer is called motherboard which contains the microprocessor and the memory chips. Typically, how many PCBs a non-embedded computer can have in a single cabinet depends on the size of the PCB together with the power and cooling arrangements of the cabinet and the room the cabinet is in. For example, a desktop computer can have two to six PCBs.

Table 3: Summary of characteristics for three commonly used IC logic families

<table>
<thead>
<tr>
<th>Parameter</th>
<th>TTL</th>
<th>CMOS</th>
<th>ECL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>Medium</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Power consumption</td>
<td>Medium</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Chip density</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Cost</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>

Transistor-Transistor Logic (TTL) features

- TTL families: 7400 series: 74 (Standard), 74H (High speed), 74L (Low-power), 74S (Schottky), 74LS (Low-power Schottky), 74AS (Advanced Schottky), 74ALS (Advanced Low-power Schottky), 74F (Fast)
- Unused gate inputs: can be left unconnected (floating), but should be tied to a used input to be safe. Also, can connect to 1 or 0 depending on the input characteristic, via a pull-up resistor or pull-down resistor, respectively
- Gate output circuits:
  - Totem-pole (do not short circuit gate outputs)
  - Tri-state (gate outputs can be short circuited if only one gate is enabled)
  - Open-collector (an external pull-up resistor needed. Gate outputs can be short circuited)

Complementary Metal Oxide Semiconductor (CMOS) features

- CMOS families: 4000 series; 7400 series: 74HC (High-speed CMOS), 74HCT (High-speed CMOS, TTL compatible), 74AC (Advanced CMOS), 74ACT (Advanced CMOS, TTL compatible), 74FCT (Fast CMOS, TTL compatible), 74FCT-T (Fast CMOS, TTL compatible with TTL $V_{OH}$)
- Unused gate inputs: do not leave them unconnected (floating). Tie them to a used input. Also, can connect to 1 or 0 depending on the input characteristic, via a pull-up resistor or pull-down resistor, respectively
- Gate output circuits:
  - Regular (do not short circuit gate outputs)
  - Tri-state (gate outputs can be short circuited, if only one gate is enabled)
  - Open-drain (an external pull-up resistor needed. Gate outputs can be short circuited)
- Electrostatic discharge can damage CMOS chips. Unless properly grounded, one should not touch CMOS chips
Transistors and electronic components are placed in the center of the chip the area called **die**. That is, the digital circuits implemented by transistors are on the die. Pins (terminals) of the chip allow the components on the die to be accessible from the external world. The die is connected to the pins by means of wires. Dice are placed on a wafer. The number of dice per wafer depends on sizes of the wafer and die. The size of the die depends on the complexity (functionality) of the digital circuit!


Just as a chip design is constrained by the speed, cost, power consumption, size, weight, reliability, etc., the PCB design is also constrained by the same factors. Before the PCB design is started, we determine these constraints! Based on them, we go ahead and design the PCB. We keep speed, cost, power consumption, size, weight, etc. of the PCB in mind, by using the right number of chips, chip implementations and wiring. The hierarchy of circuits from chips to the whole system is exemplified by one of the world’s fastest supercomputer, the IBM Blue Gene/L, below.
A microprocessor chip today contains several processors (cores, central processing units, CPUs), cache memories, memory management units (MMUs) and bus interfaces. These are implemented by registers, buses, arithmetic-logic units (ALUs), sequencers and other digital circuits. All of these digital circuits are implemented by gates and flip-flops. Finally, all the gates and flip-flops are implemented by transistor circuits which are on a single die. Therefore, transistor circuits on a die implement a microprocessor. Of course, transistor circuits also implement media processors, embedded processors, I/O processors, digital signal processors, memory controllers, the memory, etc. Below the Intel Pentium 4 die is shown.

---

**Moore's Law**

Moore's Law states that every two years the number of transistors on a chip doubles. This is due to our ability to shrink the size of the transistor and control its power consumption. What we call the process on Table 2 above is a measure to figure out the size of a transistor on a chip. The process is 22 nanometer today and is reduced by one-third every two years, “leading” the Moore’s Law curve and the shrinking transistor size. Currently, we have chips with up to 7 billion transistors. For example, the Intel 60-core Phi microprocessor chip has 5 billion transistors. Other Intel, AMD, Sun, IBM microprocessor chips typically have several billion transistors. On the left, a typical MIPS-based microprocessor organization, the MIPS R10000 die is shown.
Recently, power consumption has become a major concern. This is because with so many transistors on the chip, the power consumption becomes large. Power is also related to the clock frequency: **the higher the clock frequency, the higher the power consumption.** When the power consumption is high, the temperature of the chip increases. If a hot chip is not cooled quickly, it will **burn out.** Thus, one has to use **heat sinks, fans or liquids** to cool the chip. However, cooling adds to the size, weight and cost of the chip and the PCB.

The recent shift in microprocessor design from one processor (core) to multiple processors (cores) on the chip is due to the increased power consumption. Simply put, engineers cannot keep the microprocessor chip at low temperatures with simple cooling techniques when they increased the clock frequency. They had to lower the clock frequency. But, that increased the execution time (CPU time), meaning slower speeds. The solution to keep the execution time low was by using multiple processors. All the processors execute instructions of the same application, performing more operations per clock period, compensating for the reduced clock rate. Note that a multi-core microprocessor is not a uniprocessor. It is a parallel processing system! A multi-core chip requires a new CPU time equation. It cannot use the one given in the textbook. What can it be? Below, two multi-core dice are shown. One is the 9-core IBM Cell die and the other is the 2-core Intel Itanium-2 with 1.72 billion transistors.